Operating System Directed Power Management

A thesis submitted to the School of Computer Science and Engineering at The University of New South Wales in fulfilment of the requirements for the degree of Doctor of Philosophy.

David Snowdon

March 4, 2010

I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, or substantial proportions of material which have been accepted for the award of any other degree or diploma at UNSW or any other educational institution, except where due acknowledgement is made in the thesis. Any contribution made to the research by others, with whom I have worked at UNSW or elsewhere, is explicitly acknowledged in the thesis. I also declare that the intellectual content of this thesis is the product of my own work, except to the extent that assistance from others in the project's design and conception or in style, presentation and linguistic expression is acknowledged.

Signed:

Date:

To my sister Carolyn

ABSTRACT

Energy is a critical resource in all types of computing systems from servers, where energy costs dominate data centre expenses and carbon footprints, to embedded systems, where the system's battery life limits the device's functionality. In their efforts to reduce the energy use of these system's hardware manufacturers have implemented features which allow a reduced energy consumption under software control.

This thesis shows that managing these settings is a more complex problem than previously considered. Where much (but not all) of the previous academic research investigates unrealistic scenarios, this thesis presents a solution to managing the power on varying hardware.

Instead of making unrealistic assumptions, we extract a model from empirical data and characterise that model. Our models estimate the effect of different power management settings on the behaviour of the hardware platform, taking into account the workload, platform and environmental characteristics, but without any kind of a-priori knowledge of the specific workloads being run. These models encapsulate a system's knowledge of the platform.

We also developed a *generalised energy-delay* policy which allows us to quickly express the instantaneous importance of both performance and energy to the system. It allows us to select a power management strategy from a number of options.

This thesis shows, by evaluation on a number of platforms, that our implementation, Koala, can accurately meet energy and performance goals. In some cases, our system saves 26% of the system-level energy required for a task, while losing only 1% performance. This is nearly 46% of the dynamic energy.

Taking advantage of all energy-saving opportunities requires detailed platform, workload and environmental information. Given this knowledge, we reach the exciting conclusion that near optimal power management is possible on real operating systems, with real platforms and real workloads.

ACKNOWLEDGEMENTS

As with most theses, this could not have been produced without the effort of many. The most directly involved have been my supervisors, particularly Gernot Heiser and Stefan Petters, from whom I have learned an enormous amount. Gernot is a man who demands the best from his students. I have watched him build a brilliant and invigorating research lab at NICTA during my tenure as a PhD student. Stefan always provided as much time as I need or wanted, displaying his vigorous approach and infectious enthusiasm. Sergio Ruocco provided me with guidance during the earlier phases of this thesis.

While they were never official supervisors, there are two other people who provided an enormous support during my thesis. In one afternoon, Mothy Roscoe, while visiting NICTA, was able to help re-focus and further enthuse me about this research. He asked the question ”What makes what you are doing really hard?”, and wouldn't accept my initially fluffy responses. Thinking about these issues, and the processes which he suggested, eventually brought me to focus my PhD on model-based power management. David Johnson, who was a member of the technical staff at CSE, taught me more than any other about electronics and how to build real electronic systems. He designed much of PLEB 2, and it was his influence that allowed me to design the I-Box and Echidna among many other electronic projects. He's not a member of the teaching staff, but he taught me an enormous amount.

I've been fortunate enough to have a wonderful group of people who I've worked with over the last few years. Etienne Le Sueur was first a summer intern, then a research assistant, and now a master's student continuing this work. He has worked hard to help prove that my system works, has run many of the experiments which I discuss herein, and has been a vital bouncing board for various ideas. Many thanks go to him. In addition to that direct involvement, it has been a privilege to work with the incredible team at ERTOS and OK Labs: AB, Chuck, Benno, Luke, Adam, Leonid, Godfrey, Dan, Carl, and all the rest of you.

My family is an inspiration to me and others. They are always welcoming, always positive, and always there for me. It was my Dad who pointed me toward a PhD, and it is his constant enthusiasm and positivity that I try to emulate. Mum is always the calming and sensible influence, always there for a hug, or there to hear my complaints when the models just wouldn't fit the data. My sister, who was struck with a serious illness during the course of this thesis, is a true inspiration — she remains happy and positive, despite her hardships. My brother has also been there each Monday for "family night in" to hear the academic war stories.

Many friends have helped me through the inevitable low periods, drunk coffee in the high periods, and provided a wonderful counter to the academic life. Too many to mention individually, but Lisa and Beth were coffee and movie fiends; Fiona loved the occasional slurpee; Serin and Haydn lapped up the beers, sun and snow; Mitch provides intellect and grace via a whole lot of craziness; Em provides fierce competition on the tennis court; and Natalie has been a caring and reliable friend, sharing the PhD highs and blues. Rani has been wonderful to have in my life for the last little while, showing amazing interest and enthusiasm about the intricate details of computers' operation for an English teacher.

Lastly, the solar car project at UNSW has to be granted thanks for many distractions, learning experiences, thrills and heartbreaks.

PUBLICATIONS

Koala: A platform for OS-level power management David C. Snowdon, Etienne Le Sueur, Stefan M. Petters and Gernot Heiser *Proceedings of the 4th EuroSys Conference, Nuremberg, Germany, April, 2009*

Integrating real time and power management in a real system Martin P. Lawitzky, David C. Snowdon and Stefan M. Petters *Proceedings of the 4th Workshop on Operating System Platforms for Embedded Real-Time Applications, Prague, Czech Republic, July, 2008*

Accurate on-line prediction of processor and memory energy usage under voltage scaling David C. Snowdon, Stefan M. Petters and Gernot Heiser *Proceedings of the 7th International Conference on Embedded Software, Salzburg, Austria, October, 2007*

Accurate run-time prediction of performance degradation under frequency scaling David C. Snowdon, Godfrey van der Linden, Stefan M. Petters and Gernot Heiser *Proceedings of the 3rd Workshop on Operating System Platforms for Embedded Real-Time Applications, Pisa, Italy, July, 2007*

Power management and dynamic voltage scaling: Myths and facts David C. Snowdon, Sergio Ruocco and Gernot Heiser *Proceedings of the 2005 Workshop on Power Aware Real-time Computing, New Jersey, USA, September, 2005*

Power measurement as the basis for power management David C. Snowdon, Stefan M. Petters and Gernot Heiser *Proceedings of the 1st Workshop on Operating System Platforms for Embedded Real-Time Applications, Palma, Mallorca, Spain, July, 2005*

CONTENTS

LIST OF FIGURES

LIST OF TABLES

SOURCE CODE LISTINGS

CHAPTER 1

INTRODUCTION

> *"We choose to go to the moon in this decade and do the other things, not because they are easy, but because they are hard"* – John F Kennedy, 1962

Energy is a critical consideration in all areas of engineering. None more so than in the development of computing systems. Global warming has seen a focus on energy efficiency for environmental reasons but the recent focus on minimising the energy use in computers has also been driven equally by the rise of mobile (battery powered) computing and the thermal problems associated with increased performance.

Energy efficiency is increasingly important for servers and data centres due to the cost of power for computation and cooling [128]. This is not only a monetary concern, but an environmental one: in 2006, each server had a yearly carbon footprint similar to 1.5 cars [162], and those servers used 1.5% of the total U.S.A. electricity production [88] at a cost of about $4.5 billion [156].

On the other hand, the rise in popularity of mobile, battery-powered devices has also driven an interest in energy efficiency: reducing power consumption while maintaining performance [98]. Since the charge stored in a battery is finite, the utility of mobile devices is limited by the energy used. In this case, an increased energy efficiency results in a more useful device with the same battery lifetime (or a smaller battery, for the same utility).

Power management is a critical component of any energy-efficient computer system design. It is the process of managing a computer's hardware to achieve power-related goals. This is intrinsic in hardware design, but must be dealt with explicitly at a software level. In nearly all medium and high-performance systems, an operating system manages the hardware, including its power management. At

a cluster level, the operating systems running on cluster nodes execute decisions made higher in the software stack.

Modern hardware provides many features for reducing power and energy that require software control. A study of these features forms part of Chapter 2. These can generally be classified as either active power management (where the performance of a system can be traded for a reduced power), or idle power management (where the system is put into a sleep state while idle). This thesis focuses on active power management, and in particular deals with *dynamic frequency and voltage scaling* (DVFS).

Making optimal power-management decisions is a difficult problem, since a system's performance must inevitably be traded for energy savings, and it is often difficult to predict the effect of choosing one setting over another. Worse still, the interaction between the different hardware controls is not obvious. As an example consider the question of whether to race-to-halt when frequency scaling: should the CPU be throttled to its minimum performance, minimising the time spent idle? Or should the CPU run as quickly as possible, consuming more power, but allowing more time to be spent in a low-power mode? The answer is completely dependent on both the platform and the workload. In fact there is often an optimum setting which is neither the highest or lowest, and that setting is highly dependent on workload. Chapter 3 presents an experimental study of these difficulties and others in active power management.

To deal with this complexity, both operating-system designers and power-management researchers make assumptions about, and simplifications of, the system behaviour. These assumptions lead to the development of heuristics which can be used to guide power-management decisions. One clear example is in active power management, where the assumption is often made that the lowest performance setting will always be the most energy efficient. This thesis shows that this is false, and counter-intuitively, can lead to an *increased* energy use. The ineffectiveness of the heuristics is also discussed in Chapter 3. In an alternate formulation: previously proposed and implemented power-management schemes are largely unaware of the effect of their actions. The result of this heuristic approach is that power-management policies can at best expect reasonable, rather than optimal, results, and are highly unpredictable when used with platforms, workloads or

combinations of workloads for which the system has not been tuned. Moreover, the performance effects of power-management decisions can not be predicted or managed.

Instead, this thesis proposes that systems should be made aware of the effect of their power-management decisions. Instead of hiding the real hardware behaviour behind often-erroneous heuristics, we suggest that power-management schemes need an accurate model of the running system to make their decisions. This allows the power-management policy to evaluate different strategies for a predicted workload, trading performance for energy savings, while the details of the platform's behaviour are abstracted and encapsulated by the model. This allows high-level policies to deal with interfaces which are platform and workload independent.

To do this, we first studied ten different platforms' response to running different workloads. This experimental work conclusively shows that the majority of the platforms did not behave the way the academic literature would expect. For example, some platforms had multiple frequency control knobs which make choosing a combination of settings difficult. Other platforms have DC-DC converters with non-linear efficiency-power curves. In all platforms examined, the performance of the memory subsystem does not scale linearly with the CPU frequency. There are several more of these irregularities, which we term quirks.

This thesis develops models which represent these effects based on experimental data gathered off-line. The models are use a locality argument, and estimate the workload's future behaviour based on statistics gathered during its recent execution, rather than using any kind of workload pre-characterisation. Based on these inputs, the models predict what the execution time and energy use would be for the same workload running at a different setpoint. The models are sufficiently light-weight so as to be used at run-time without a significant overhead. As part of this thesis work we developed models for several platforms on multiple architectures. These can predict many of the quirks which we discovered.

The models can then be used to accurately trade the benefits of the performance hit at a given setpoint against the energy savings by running at that setpoint. In Chapter 5 I discuss the ways in which this can be done, first examining the effects of running some simple policies, and then developing a new policy, the

generalised energy-delay policy. This policy forms a middle layer in the power-management stack, taking the energy and performance results for a range of candidate settings and selecting one based on a single tuning knob. This tunable forms an abstract interface with higher-level policies, allowing a system to be explicitly adjusted for maximum performance, minimum energy use or minimum power (and all of the settings in between). This can adjust the system based on QoS information such as system utilisation or battery state of charge.

This thesis conducts a thorough evaluation by developing an operating-system which implements the above concepts. It uses a simple prediction technique to regularly estimate characteristics of the upcoming workload, allowing our models to calculate the expected performance and energy use required for each candidate setting. Our policy can then select the appropriate setting. We conducted a substantial experimental validation characterising and testing the models and policies, and validating the operation of the final implementation.

The contributions of this thesis are:

- an experimental study of the effect of using traditional heuristic-based active power management on various platforms (and a record of the quirks found);
- the most complete models for a system's energy consumption for a variety of workloads and power management settings, including:
  - the first models of both the system's performance and its power which are sufficiently accurate for estimating the energy consumption of a variety of workloads under varying conditions;
  - a discussion of the effect of often-ignored variables such as temperature and power supply efficiency;
  - methods for developing and characterising on-line models of an unknown platform based on experimental data gathered off-line;
- policies and mechanisms for using these models to accurately trade energy against performance at run-time;
- an implementation of the ideas in Linux, called Koala, resulting in a dynamic, specific and efficient trade-off between the system's performance

and power;

- a thorough evaluation of this implementation.

The rest of this document is organised as follows: Chapter 2 provides background material and related work pertaining to active power management. Chapter 3 presents a study of a large number of platforms, showing the ways in which these platforms behave in non-intuitive ways which complicate active power management (quirks). Models which provide estimates for the performance and energy use for a workload on a given platform are presented in Chapter 4. I also discuss ways in which the model parameters can be chosen, and the models characterised, based on empirical data (collected off-line). Chapter 5 looks at how power-management decisions can be made based on knowledge of the performance and energy savings information provided by the models, and presents the *generalised energy-delay policy*. Higher-level policies such as those based on battery state of charge and utilisation are also considered here. The details of *Koala*, an implementation of these ideas in Linux is documented in Chapter 6. The accuracy of the models and Koala's effectiveness is evaluated in Chapter 7. Chapter 8 provides conclusions, detailed contributions and suggested future work.

This thesis contains material published in peer-reviewed workshops and conferences. At the time of writing I am the primary author of all but one of these publications [91, 144–148].

CHAPTER 2

BACKGROUND AND RELATED WORK

> *"Opportunity is missed by most people because it is dressed in overalls and looks like work"* – Thomas Edison,

Energy, having such an enormous impact on the utility of both mobile and fixed computing systems, has seen significant attention in both the academic literature and industrial research laboratories. This chapter explains the concepts necessary for a computer scientist to understand this thesis, and examines the excellent work done by academics and industry.

The rest of this chapter is structured as follows. Section 2.1 discusses the basic physics of how computers use power. Section 2.2 discusses ways of evaluating the power consumption of computer systems, in particular the energy accounting and modelling techniques which we build on in Chapter 4. Section 2.3 looks at the difficulties for comparing power management policies and compares the techniques. Sections 2.4 and 2.5 describe policies for dynamic frequency and voltage scaling, and dynamic power management, respectively. Here we include real-world schemes, a description of the hardware support available, and many of the policies which inspired this work. The last two sections, 2.7 and 2.8, look at workloads and the ways in which our system can be supported by the applications themselves.

2.1 How computers use power

Computer systems consist of a group of interconnected electronic, electro-mechanical, electro-chemical, etc, devices. These systems use energy at a rate that is called power. Reduced power implies a reduction in the amount of energy that is used in a given time period, so

$$E = Pt. \tag{2.1}$$

Power in an electrical system is the product of current and voltage:

$$P = IV. \tag{2.2}$$

Capacitors store a charge, and the charge (q) is proportional to the voltage across the capacitor.

$$q \;=\; CV. \tag{2.3}$$

Current is the rate at which charge flows. The current into a capacitor is proportional to the rate of change of its voltage as

$$I \;=\; C\frac{dV}{dt}. \tag{2.4}$$

and the stored energy is

$$E = \frac{1}{2}CV^2. \tag{2.5}$$

2.1.1 Integrated circuits

Integrated circuits combine a many electrical components in a single package. One important component is the *field effect transistor* (FET), which consists of a capacitive *gate* which controls a channel between the *source* and *drain*. If the capacitor is charged beyond its *threshold voltage* (V_{th}), the channel conducts, if not, it has a high impedance. There are two different types of FET – n-channel FETs will switch on when the voltage at its gate is positively charged (relative to the source/substrate) and p-channel FETs will switch on when the gate is negatively charged.

Many circuits within computing systems are based on *complementary metal-oxide-silicon* (CMOS) fabrication. The circuits implemented use two complemen-

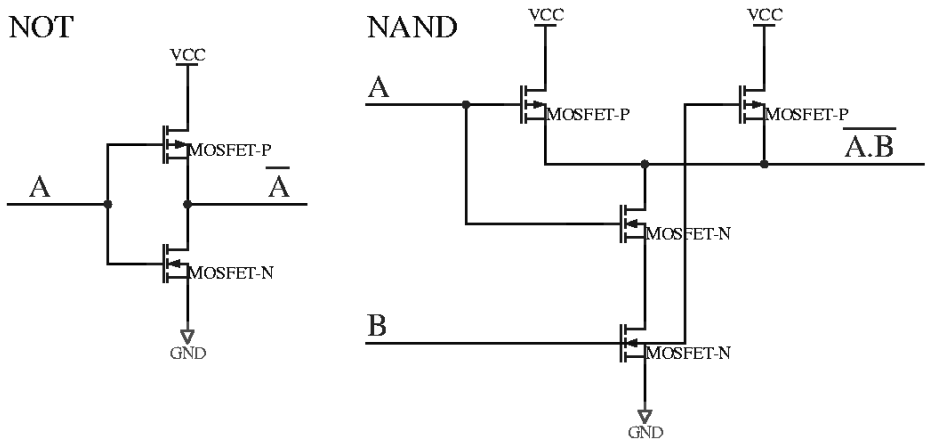


Figure 2.1: CMOS logic gate configuration

tary devices to implement various functions. Figure 2.1 shows two common logic gates as implemented in CMOS.

These transistors are connected in networks and form integrated integrated circuits. Power is dissipated by these devices due to:

- Gate capacitance: when the gate capacitor is charged and discharged, that charge is lost (see Equation 2.5).

- Short circuit power: as the gate capacitor charges, a FET will be partially on. Since both FETs in a CMOS pair switch concurrently, there will be a short period of time where current will flow through both.

- Leakage current: the devices are not ideal, and constantly conduct a small current between all of the terminals.

Ignoring leakage and the effect of non-CMOS components, a processor's power consumption is dependent on the number of transistors and gate capacitance of the transistors switched. For a CMOS circuit which is switched at a frequency (f), with a constant number of transistors switching, the switching losses are proportional to the energy stored in the gate capacitance and the rate at which the transistors switch

$$P = fn\frac{1}{2}CV^2. \quad (2.6)$$

Reducing the voltage on a switching circuit reduces the switching energy required for a given operation. However, the voltage can not be reduced to zero, since a reduced voltage results in a slower transistor switching speed, which limits the frequency at which clocked circuits can operated correctly. In order to improve the switching speeds, the transistors can be designed with a lower threshold voltage but this results in increased leakage.

Another way to reduce energy consumption is to reduce the number of transistors which switch. The number of gates switched can be reduced via design changes. In addition, the number of gates switched by a clock can be reduced by clock gating. This stops a clock signal from connecting to parts of the circuit which do not require it, reducing wastage in those circuits.

2.1.2 Microprocessors

The number of transistors switched in a microprocessor will be depends on the software which is executed. The use of functional units, registers, busses, and other hardware affects the power drawn. For example, the task of loading a register with very different data to what it previously contained will use more energy than loading a register with very similar data.

Clock gating is used extensively in modern microprocessors. Similarly, clock *throttling* reduces the system performance and power draw by stopping the processor's clock periodically. The clock is run for some period, and then gated for some time. Clock throttling is implemented in several processors as a simple way of implementing thermal control.

A processor's switching speed, and switching power loss, are governed by clock frequencies. When clocks are slowed, the transistors can take longer to settle, and so the voltage used to switch those transistors can be reduced. The voltage needs to be high enough to allow the processor's critical path to function correctly. The critical path is the longest chain of dependent circuits.

Dynamic frequency and voltage scaling (DVFS) allows a system to switch its frequency and voltage at run-time, and is implemented in many modern microprocessors.

Choosing the best setting is a focus of this thesis. In particular, we consider the inadequacies of Equation 2.6 for predicting the effect of a frequency change. While it is a good approximation of the switching costs of a circuit where the same number of transistors switch on a single clock cycle, it has fails to model a complex microprocessor because it assumes a single clock. It assumes no leakage or background power.

When a processor is idle it makes little sense for the circuit to continue to execute an infinite loop (*spinning*). For this reason, microprocessor and peripheral developers implement *sleep modes*. These are low-power modes where the processor retains some state, but is inactive. Because it is inactive, clocks can be disabled, voltages can be lowered, and sometimes power supplies can be physically disconnected. This allows for significant power reductions. The power saved is often related to the time required to enter and exit the sleep mode, since some state is lost and must be restored. For this reason most systems implement a range of sleep states so that the entry and exit time can be traded against the power saved. Managing these sleep states is a significant operating systems challenge.

2.1.3 Memory

In many systems, devices implementing memory consume a significant amount of energy. Itoh [85] gives a good summary of how power is dissipated in RAM.

There are two popular RAM technologies which almost all computers use at present; *static RAM* (SRAM) and *dynamic RAM* (DRAM) RAM. Static RAM is implemented using a small number of transistors per bit. Dynamic RAM uses a smaller number of transistors and a capacitor. DRAM is cheaper and denser than SRAM, but SRAM operates at a higher speed and is usually used for caches and memory where a low latency is required.

As with microprocessors, hardware designers have implemented software-controlled features which can be used to reduce the power drawn. Manufacturers such as Micron [104] have implemented low-power DRAM with features like par-

tial refresh, temperature-compensated self-refresh, and various sleep states. *Rambus DRAM* (RDRAM) offers four different modes of operation (as discussed and utilised by Lebeck et al [92]) with a tradeoff between power consumption and latency in recovery. Because of RDRAM's narrow bus, architecture individual chips can be mode-controlled independently of one another. These techniques require OS support to be effective.

2.1.4 System and other peripherals

Peripherals are the largest consumer of energy in many systems. In Lorch's survey [94], more than 40% of power was consumed by the hard drive, backlight, display and modem in a laptop. In Flinn's analysis of the Itsy pocket computer [45], he shows that the LCD interface and UART both have a significant power consumption (these devices are considered peripherals but are built into the system-on-a-chip in the Itsy). The power consumed by these peripherals can be reduced in hardware via means specific to the device in question, but usually in software via use of power-down modes. These modes can make a significant difference to the power used by the system and are discussed by various authors [29, 38, 105, 157].

2.1.5 Power supply

Computers use regulators to convert from an input voltage to that provided to the processor and other system components. These power supplies are usually switching regulators which convert relatively, but not perfectly, efficiently. The losses depend on load, voltage buck/boost ratio, space and cost constraints.

Martin [98] examined the non-ideal properties of batteries in relation to computer systems and their use. In particular, he notes that a battery's capacity is reduced when it is discharged at a higher rate. Accordingly, constant loads will give a greater battery lifetime than an intermittent load with the same average current.

This is expressed in a simple form by Peukert's equation

$$Q = \frac{k}{I^{\alpha}} \tag{2.7}$$

where Q is the available battery charge, k and α are constants specific to the battery and I is the proposed discharge current. This does not consider the properties of specific battery chemistries (such as voltage recovery in *Lithium Ion* (Li-Ion) cells. Martin uses a low-level Li-Ion specific battery model in his work on battery-aware DVFS.

2.2 Evaluating computer power

Computers' complicated use of energy make analysing how and why it is used difficult. This section discusses some methods which have been developed.

2.2.1 Simulation

Simulation re-creates the behaviour of a computer in software. Many have been augmented with models for a system's power consumption. These simulators vary in accuracy, with accuracy often being traded for performance or ease of implementation.

Hardware designers often use architecture-level simulations based on the circuit and physical properties of the system. It is difficult to get access to such simulators for real systems, and it is computationally expensive to run the simulations. Brooks et al. [22, 23] developed add-on modules for SimpleScalar [12], adding a set of models for common structures in microprocessors. SimpleScalar accounts which devices are used, and records the energy used by the workload. This gave estimates of the power with less than 10% error. This type of simulation can give detailed information about the energy used by a software. Brooks was able to examine the effect of techniques like loop unrolling on the system's power consumption. A myriad of other work has improved the speed of these simulators for the purposes of system evaluation [15, 16, 54, 152], however these improvements have not been translated to run-time low-overhead power estimation, nor can they be applied without the detailed knowledge of a system's inner workings available in a simulation environment.

System-level simulators sacrifice some accuracy in order to simulate complete systems, including peripherals, without substantial information about the internals

of a processor. For example, Simunic [139–141] developed a simulator based on the ARMulator [5] and used it to simulate the SmartBadge (a small embedded system). She shows how information from this simulator allowed her to reduce the energy consumption of an MP3 playing application by more than 75%. She claims that her results are within 5% of the measured power for the same system. Jouletrack is another system-level simulator [143]. The power models were constructed by executing synthetic microbenchmarks on a StrongARM development board to characterise specific behaviours. A similar approach was used by Gurumurthi [60] who constructed a system-level simulator using SimOS. He analysed the JVM98 benchmark as well as the IRIX operating system. Interestingly, in the test system, the disk was the major power consumer. Fan [40] modified a simulator when estimating the power consumed by an XScale-based device with power-aware SDRAM. Hellestrand [62] described the event-based power modelling techniques used in the VaST commercial virtual system prototyping environment.

Simulation is an effective evaluation technique, but does not allow for run-time power measurements, and is therefore not useful in the context of this thesis. For evaluation of power management mechanisms, we find a power measurement of the real system much more convincing, since practical simulators are based on models which are necessarily inaccurate for tractability. However, the techniques developed to for these simulators provide inspiration for run-time modelling based on energy-accounting techniques.

Various simulation work is relevant to our own via modelling techniques. Zhang et al. [177] discussed the exponential dependence of the leakage current (and therefore static power) on temperature, and the linear dependence on voltage. Bansal et al. [15] give a thorough description of the statistical techniques used to choose parameters in their power-estimation models.

2.2.2 Energy accounting

Energy accounting is used for estimating the energy used by software at run-time. Like in the simulators above, data is gathered about the system state and events, and this is used to model the energy consumed by the system. Events might

include spinning up the hard disk, executing a particular instruction, a network packet transmission, etc. The data can be gathered in a lightweight way, allowing for low-overhead run-time power estimations.

Some examples of this type of technique include PowerMeasure/StateProfiler [94] which characterised a platform using microbenchmarks, and then estimated the real-world energy for a number of identical systems at Apple Computer Inc. Neugebauer and McAuley [110] proposed extensions to the Nemesis OS's in-built resource accounting features to support energy accounting.

Event-counter based techniques, typified by Bellosa and Weissel's work [17, 166] use the data recorded by CPU performance counters as the input to a model. The counters are configured to measure events which are significant to the energy consumption (cache misses, instructions retired, etc). The accuracy of this type of system is determined by the amount and relevance of the information which can be collected. These types of hardware event counter are typically only available in the CPU and chipset, and therefore do not generally apply to peripheral power estimation. An exception to this is memory power, which can be inferred by counting cache misses and write back operations in the CPU.

Bellosa [17] demonstrated that it was possible to correlate hardware performance-monitoring counter (PMC) readings with the energy consumption of a CPU. His students Waitz [163] and Fruth [51] used this approach to implement energy accounting to processes in Linux on an Intel PXA255 processor. Fruth effectively worked around that processor's limitation of only two PMCs by switching the monitored event every 10 ms.

Isci and Martonosi [83], Bircher et al. [19] and Kumar et al. [90] used the PMC-based approach on the Pentium 4, which has many counters, and investigated the suitability of different counters. Bircher et al showed that a processor's performance counters have a good correlation with the power consumed *outside* the processor when estimating the power for a complete system [20]. Research using the IBM Power processors [37] developed a predictor of program phases in terms of power consumption, similar to dynamic branch predictors. This was later ported to the Intel PXA 255 processor [30]. Peddersen and Parameswaran [113] investigated custom CPU designs which provided PMCs counting events specifically selected for estimating the power consumption of the CPU.

More recently Powell et al. [123] developed CAMP, which measures nine statistics in order to predict run-time power consumption of many structures within a CPU to within an average of 8%. The authors suggest that the methodology is sufficiently lightweight to be implemented in hardware, like Peddersen's work. Such in-built power-estimation hardware would clearly improve the accuracy and reduce the calculation overheads of our own models.

An interesting approach to modelling was taken by Singh et al. [142], who used a piecewise approach: using different models based on the value of certain parameters. This is similar in some ways to using *categorisation and regression trees* (CART) [172].

State-based accounting techniques such as those employed in ECOSystem [176] and Cignetti et al. [29] instrument operating system software to track the state of the CPU and its peripherals (e.g. for a disk, whether it is spun up or down and whether it is active or idle). The duration spent in each state is used as the input to a model. This is an all-software solution to power estimation, but fails to capture any variation of the power within a given state. The accuracy of the technique therefore depends on the number of states (detail of the model). In ECOSystem the recorded energy-related information is used as feedback for system policies such as energy budgeting.

These energy accounting techniques have seen applications including resource partitioning in virtualised servers [150], run-time application adaptation [114], energy budgeting [176], temperature estimation [18, 28] and evening the thermal dissipation in multiprocessors [102].

Energy accounting is a lightweight way of estimating the power for a system at run-time. Our approach builds on these power estimation techniques: because energy-accounting uses a model, we are able to introduce extra parameters which allow the estimation of not only the actual system power, but what the system power would have been under alternate conditions (in our case, alternate power management settings). To our knowledge, ours is the first system to do so. This is described in Chapter 4.

2.2.3 Run-time measurement

Statistical profiling of power was developed by Jason Flinn [43, 47] to assist his work on energy-aware software adaptation. PowerScope is an energy profiler, giving feedback to a developer on the energy consumed by processes, and the energy consumed by components within that software (functions). A data collection computer uses a *data acquisition system* (DAQ) to monitor the power consumed by a computer under test. Each time the DAQ samples the power, the system under test is interrupted. The interrupt handler records the *program counter* (PC) and *process ID* (PID). This data is later processed to generate an energy profile (energy samples are attributed to the PC and PID recorded). Flinn discusses some of the arguments for and against energy accounting compared with statistical profiling. One advantage of the measured data is that it captures all of the behaviour of a system, in a way that only an infinitely (hence non-existant) complex model could. It also does away with the need for characterisation of a model. Statistical sampling can not, however, appropriately attribute the power used by background tasks, such as I/O activity, to processes. Flinn suggests that a combination between statistical profiling and energy accounting could provide for more accurate results.

Chang et al [24] improved this methodology to introduce variable-rate sampling, which reduces the profiling overhead in low-power environments. we made our own improvements using custom-designed hardware (PLEB 2) and multiple sense resistors [145]. An on-board microcontroller allowed for power to be attributed to the running process without the need for a second computer for monitoring purposes.

Lauterbach have integrated these techniques into their debugging and trace analysis tools, allowing very fine-grained measurements of a system's power consumption [149].

Without a model for the system behaviour, it is impossible to predict the power under conditions other than those being measured, and so statistical sampling alone is not useful for predicting the future power consumption of a system with changed power management settings. We investigated incorporating run-time power measurement into our models by using those measurements as one

predictor of future power consumption.

2.3 Evaluation Metrics

The effectiveness of a power management scheme may be justifiably evaluated in different ways depending on the context. The relative importance of energy and performance will change based on the available computing time. In an under-utilised system, the performance is of low importance (within limits). In a fully-utilised system, a reduction in performance is likely to cause a reduction in *Quality of Service* (QoS). Given this, no single, simple policy based on a single evaluation criteria will be the "best" in all circumstances.

One method of balancing energy performance is using the *energy-delay product* (EDP) as an evaluation metric. If T_x is the execution time, and E_x is the energy used, then the EDP is

$$T_x E_x. \tag{2.8}$$

The EDP is small when either the energy is small (a significant energy saving) or the execution time is small (a boost in performance). If the performance is reduced (an increased execution time) there must be a corresponding decrease in the energy consumed to obtain an equivalent EDP.

Saeides et al [127] point out the problems an inaccuracy when using an $energy \times delay$ metric in an interval based system. Evaluating the sum of the ET for each interval in an application is not the same as evaluating the ET for the entire application. This is dissimilar to minimising energy (where minimising the energy for each interval minimises the energy for the total), or the performance (where minimising the time to run each interval minimises the time to run for the whole application).

Penzes and Martin [115] introduced the idea of a geometric weighting for the EDP in the context of VLSI designs. They suggest that ET^n be used. Sengupta and Saleh [131] also discuss a family of evaluation metrics which they use to evaluate the performance of CMOS circuits. They consider several metrics, including *power-energy-product* (PEP).

Other interesting metrics of similar intent include *millions of instructions per Joule* (MIPJ) [164].

2.4 Dynamic voltage and frequency scaling

DVFS was first discussed by Weiser et al. [164]. Their work makes clock scaling decisions each time the OS's scheduler is invoked. The system attempts to minimise CPU idle time by lowering the CPU frequency. Energy was saved via voltage scaling. This feedback mechanism is the current state-of-the-art in many modern operating systems. They used *millions of instructions per joule* (MIPJ) as a metric when evaluating their scheme, however they ignored static power and the other complications discussed in Chapter 3. The evaluation in this system was conducted via simulation based on real traces.

Govil [56] extended Weiser's work by developing algorithms which predict future CPU utilisation based on recent system events, recognising cycles and other patterns in the workload behaviour. Pering et al. [118] simulated systems when running an MPEG decoder which only has soft QoS requirements. When simulated, this technique showed significant energy reductions.

Grunwald et al [58] evaluated these algorithms on real hardware (rather than using simulation), concluding that they did not save significant amounts of energy. One clear reason is that the algorithms attempt to minimise idle time, ignoring the DVFS complications outlined in Chapter 3. Reducing the CPU frequency increases execution time, leading to a decrease in the amount of time spent idle. If the energy benefits of running at the lower frequency do not offset the energy spent due to the reduced time in the idle mode (because of the static power), the total energy is *increased*. These effects were discussed by Miyoshi et al [107], who developed a method for choosing the minimum frequency which saves energy on a given platform. Unfortunately, Miyoshi et al. make assumptions of their own: that all workloads scale with frequency in the same way. DVFS policies designed to minimise performance loss, such as that of Weissel and Bellosa [166], implicitly avoids these issues.

Poulwese et al. [122] measured the power consumption of a LART device [14], measuring real-world power consumption of a DVS technique associated with a

video CODEC. Pouwelse called for greater OS interaction with applications in order to obtain more information about future resource demands, avoiding heuristic predictions.

PACE [95] improved the above algorithms by utilising deadline information, developing a speed schedule where the CPU frequency is increased as the deadline approaches. They found that the approach saved about 20% more energy on average than the base algorithms which they improved.

Martin [98] modified Weiser's algorithm to account for the non-linear capacity drain vs. current drain characteristics of batteries.

Several techniques which do not directly attempt to minimise the performance of the system have been developed. Off-line techniques [2, 69, 170] use a detailed static analysis of a workload by the compiler to estimate the workload characteristics. Other approaches include an *a priori* characterisation of a workload by running it at two different frequencies, in order to derive a slowdown relation [161]. These off-line results are then used by a DVFS-aware scheduler to scale the processor frequency.

Systems where no a-priori characterisation is performed generally aim to get the best energy efficiency for a given performance impact [100, 107]. Such early work was typically based on the incorrect assumption that performance was proportional to CPU frequency. The highly variable dependency has since been the subject of considerable investigation.

Many DVFS policies, ours included, use PMCs as a guide to predicting the likely performance impact of a frequency change. Process cruise control [166] used the number of instructions, memory accesses and cycles to index a precomputed table of frequency settings which lead to a constant 90% performance impact for that type of workload. They observed significant observed energy savings despite the minimal slowdown. Due to hardware constraints, they were only able to estimate the effect of voltage scaling (as opposed to pure frequency scaling). Other research groups have investigated a more flexible technique using a regression performed at run-time to calculate the ratio of off-chip (CPU frequency independent) to on-chip cycles [25, 68]. The computational overheads and response time of their technique are only briefly discussed and there is reason to believe that they are substantial (i.e. the evaluation of a regression requires signif-

icant CPU time).

A theoretical model of a classification system between memory-bound and CPU-bound applications [171] assumes the availability of a large number of concurrently-usable performance counters. Limitations of this work include a lack of a detailed justification of performance-counter selection, and an insufficient evaluation. Kotla et al also looked at this problem [89]. One interesting observation in their work is the presence of variable latency memory accesses generated by pre-fetching and other micro-architectural effects (Jack Doweck [36] gives a good overview of these). Seth et al [132] based their work in real-time DVFS on that of Martin and Siewiorek [100]. They concluded that taking advantage of the non-linear relationship between CPU frequency and performance could yield significant extra energy savings.

Others have attempted to take advantage of some of the same power management challenges that we present in Chapter 3. Herbert and Marculescu present a policy which is aware of the variation between cores [65].

Many other DVFS algorithms exist [41, 56, 59, 95, 98, 118, 120, 122, 135, 159, 164] with varying applicability. Others have investigated DVS in particular applications such as sensor networks [105, 106].

2.4.1 Quality of Service

Malik et al. [97] investigated the user response to QoS degradation brought about by a reduced frequency. They used this direct user feedback as a metric for frequency scaling: the user would press a key when they were dissatisfied with the system performance. Two algorithms were tested which used the feedback to increase and decrease the frequency. The system was tested with 20 different users and 3 applications. Two major results are apparent: users had a wide variation in what performance they found acceptable, and the mean frequency varied substantially for different applications. While this work assumed that a lower frequency resulted in a lower energy use, the idea of user-feedback as a QoS metric is complementary to our work, since it can provide information about the required performance.

2.4.2 Real-time DVS

Real-time systems, which are required to deliver results by a deadline, require knowledge of the system timing, frequently in the form of *worst case execution times* (WCET). There is a multitude of work in this area, of which we have examined a selection [3, 13, 86, 87, 120, 121, 126, 135, 173]. Pering et al. [117] obtains a-priori information about whether the application is rate-based or deadline-based. Pillai's and Shin's static scheduling techniques [120] attempt to maximise the system utilisation in hard real-time systems (where deadlines can not be missed). Some have used the CPU's memory stall rate in a feedback loop with the scheduler [121, 126].

Shin et al. [135] modified the compiler to insert frequency scaling calls. The compiler estimates the WCET of each basic block and uses this to continually re-evaluate the DVFS setting as the task progresses — keeping the system speed high enough that the remaining WCET does not exceed the time until the deadline. Kim [87] also looked at hard real-time systems.

Our own work [91] extended the RBED EDF-based scheduler [21] to throttle processes appropriately based on the available *slack* time, which is created when tasks complete in less than their WCET.

Like other traditional DVFS techniques, these real-time algorithms are based on unrealistic assumptions. However, these algorithms examine ways of calculating a minimum performance requirement for a task, which complements our work well by providing a bound on the performance loss than can be tolerated for a given task.

2.4.3 Hardware support

Most modern CPUs provide support for DVFS. Intel's mobile Pentium III introduced their SpeedStep technology [33], which allows two frequency/voltage combinations to be pre-programmed into the processor. It switches between these two states depending on whether the system is operating from a battery or from an external power source (i.e. not under software control). AMD's PowerNow! [35] technology, Intel's XScale [32], Transmeta's Crusoe and Efficeon processors [42] and many others (details of the specific algorithms we evaluated are given in Ap-

pendix A). The Crusoe was interesting in this regard, with the hardware automatically scaling the CPU frequency via an algorithm similar to Weiser's [164].

During the course of this thesis, several platforms were released with interesting DVFS capabilities. Of particular note is Intel's Core i7 processor, based on Intel's Nehalem microarchitecture. This processor has a dedicated on-die microcontroller [67], called the *power control unit* (PCU), used for making power management decisions. It can read sensors and performance counters, and can change core frequencies, sleep states, etc. The details and firmware for this is not available, but it is highly likely that the techniques presented in this thesis would be applicable in such a system, and using an on-die microcontroller would allow better sensing, finer granularity and lower overhead. A similar power management controller, Foxton, is used on Intel's Montecito processor, described by McGowan et al. [101]. The Foxton controller adjusts the processor's power to some limit, given feedback from on-die *analogue-to-digital* (ADC) converters.

ARM's *intelligent energy manager* (IEM) has a significant hardware component, which we discuss in Section 2.4.4.

The *advanced configuration and power interface specification* (ACPI) [66] exports a number of useful BIOS-level interfaces for power-aware operating systems. The specification itself is very detailed, and provides mechanisms for a wide variety of behaviours. In the context of this thesis the most interesting of these manage both low-power idle modes (C-states in ACPI terminology) and active processor modes (which is analogous to DVFS and named P-states). This is the standardised interface via which many systems expose their DVFS capability.

2.4.4 Real-world implementation

To our knowledge, all modern operating systems implement DVFS via heuristic algorithms which maximise CPU utilisation. This may be reasonable on some systems where the lowest frequency is consistently the most energy-efficient, but does not apply to all systems. Without a detailed view of these proprietary systems, it is difficult to analyse their behaviour.

Linux has support for CPU frequency scaling in the form of `cpufreq`, which can be configured to use different governors (policies). Intel's LessWatts.org

web site recommends the *ondemand* governor [112], which slowly reduces the CPU frequency based on the system's utilisation, but quickly increases it if the utilisation increases (to avoid an apparent slowdown).

FreeBSD is also has this type of support [116]. Their implementation uses a user-land interface which directly controls the CPU frequency. The `powerd` daemon which governs this explicitly cites Weiser and Govil's mid-1990s work as inspiration [168].

Despite releasing the source code for parts of their OS [4], Apple has kept the CPU power management components closed (but loadable in Darwin as a kernel extension). Anecdotal evidence from an early-2008 model Apple Powerbook running Mac OS X 10.5.8 indicates the system uses the minimum CPU frequency when the CPU utilisation allows. There are a number of alternate speed-setting extensions available for Darwin and Mac OS X [124, 158] which all attempt to achieve a given CPU utilisation minimise the CPU frequency to achieve some target CPU utilisation.

Little information could be found regarding the policies used in Microsoft Windows based systems, but the trend would indicate utilisation-based heuristics.

ARM's *Intelligent Energy Manager* (IEM) [6–10, 160] is a comprehensive solution to DVFS. It is a combined hardware and software solution, targeted at battery-powered devices, which incorporates detailed feedback from a processor to make frequency scaling decisions. It integrates a hardware component, the *intelligent energy controller* (IEC) which handles performance requests, translating these to the commands required to change the power management settings. The IEC also has three event counters, two of which can be triggered on configurable events (which are presumably similar to those which we examine in Chapter 7).

The IEM integrates a policy stack, allowing the combination of multiple policies. Similar to our method, the IEM is triggered on OS events such as context switches and interrupt handlers. The details of the policies are not discussed, but the mechanisms are in place for policies to store a history of task data which can be used to make predictions about the future behaviour of tasks. This technique would compliment this thesis as a workload prediction mechanism.

The IEM software is the work most closely related to our own. It abstracts from the details of the hardware by using performance setting indexes in a similar

way to Koala's setting numbers, uses event counters for feedback, and performs task-by-task scaling. IEM is clearly a more mature product, with areas such as workload prediction, policy interfaces and hardware integration clearly having been addressed. However, IEM does not deal with real energy and performance models in the same way as Koala: it is still unaware of the real implications of its power management decisions, whereas Koala manages these explicitly.

Freescale's *eXtreme Energy Conservation* (XEC) [27, 49] defines a software framework for power management (both DPM and DVFS) across all of their platforms. The framework involves all levels of the software stack, with a view to portability between platforms. Interestingly they consider power-aware applications, although it is unclear what this entails.

2.5 Idle mode management

All the processors we examined provided support for sleep states, where the processor halts execution for a period of time with a substantially reduced power. In fact, sleeping at some duty cycle allowed high-power processors to execute their tasks quickly, and then sleep until once again required. Decreased power is usually traded for increased wake-up time. The process of managing these sleep states is known as *dynamic power management* (DPM).

Our solution deliberately leaves idle mode management to the underlying operating system, since the appropriate idle mode is dependent on the expected time spent idle. The existing research into using processor idle modes, therefore focuses either on methods of consolidating both active and idle time, or methods of predicting the time which will be spent idle. These techniques are clearly complementary to our active power-management policies. Integrating DPM and and DVFS can bring about even more energy savings. Simunic [137], among others, shows good results for her particular application as measured for an MP3 player on a real embedded system.

Device power management also falls into this category, and future work will benefit from integrating idle-state management with our DVFS policies. An example is RAM power management. Fan et al. [40] considered DVFS in their power-aware DRAM policies, since the optimal *race-to-halt* policy conflicts with

Weiser-style DVFS. Delaluz et al. [34] examined a compiler-centric approach to analysing the memory access patterns in software.

Idle-power policies attempt to predict the future behaviour of devices based on past behaviour. This is also true of our work, where we attempt to predict future workloads. We looked at a selection of work [55, 57, 93, 136, 138, 169] looking at these types of policies. Simple timeout based policies work surprisingly well.

2.6 OS power management

Ellis [38] among others argued that power management should be *"raised to first-class status among the performance goals of OS design"*, and to some degree this has become the case, with operating systems occasionally being compared based on their power management features. Likewise, the power management features of mainstream operating systems are closely guarded secrets. For example, the open-source version of Mac OS X — Darwin [4] — does not include power management code.

There is much to be said for power management at the OS-level. Grunwald [59] argued that the operating system is the only entity with a global view of resource usage and demand — resource management and provisioning being one of any operating system's main tasks. Lu et al. [96] argue similarly: that the OS has detailed information about tasks and their requirements.

The ECOSystem project [175, 176] attempts to raise energy awareness in Linux. It performs per-task energy accounting using a simple state-based model. They introduce a fictitious unit for energy — *currentcy* — which is periodically distributed to processes. Currentcy is subtracted when a process uses energy, and when a process has no currentcy, it is not scheduled. The goal for this system is to motivate application developers to be energy-aware through feedback. ECOSystem is similar to Koala in that it uses energy accounting and modelling to attribute energy use to processes. ECOSystem's significant contribution is the provision of a unified approach to energy allocation and management.

Quanto [48] is an extension to the TinyOS operating system for wireless sensor networks. It builds linear models for these small systems using recorded data based on a low-overhead energy measurement mechanism. Quanto instruments

the OS to provide tracking of the state of each energy sink in the system, and the activity which causes that energy sink. By recording both system's total energy use and the time spent in each state, the authors build a linear model for the energy used by each energy sink in each of its states. This thesis takes the same approach. Using this instrumentation allows a user to understand where energy was used in the system and which activities caused that energy to use. By tagging network packets with an activity ID, energy use can be tracked globally in a sensor network.

2.7 Workload prediction

Estimating the behaviour of an upcoming workload has applications in many areas of an operating system, including power management, because as we show in Chapter 3, the effect of a given power management decision is highly workload dependent.

Sherwood et al. [133] showed that most programs demonstrate cyclic behaviour. They used an off-line technique to analyse the basic-block behaviour of software [134]. Using these methods they are able to identify, off-line, repetitive behaviour which could be used for workload prediction.

Canturk Isci has contributed significantly to this area. He shows that reactive predictors such as ours work well for many workloads, but not for workloads with highly variable behaviour [82]. He has developed several methods of identifying phases in workload behaviour for use with similar power management frameworks to our own. Phases are consistently behaving portions of an application. One of these uses a branch-predictor-like scheme with a *global page history table* (GHPT) to base future predictions on several previous phases [82]. Another compares a basic-block analysis technique with a performance-counter based technique for identifying and classifying phases [84]. Some of this work includes predicting the duration of a phase, as well as the type of phase [81]. This work is evaluated on real systems with real benchmarks.

Real-time systems, with their small, repetitive tasks and rigid schedules, clearly have information about future workloads. In these circumstances, reactive predictions should work well.

Nagpurkar looked at implementing phase detection in the Java run-time environment [108].

While the above is by no means an exhaustive literature review of workload prediction techniques, we expect these and other workload prediction schemes to augment our own power management framework by allowing a more accurate estimation of future highly-variable workload behaviour.

2.8 Application support

2.8.1 Adaptation

Jason Flinn used the term fidelity to describe application quality in his work regarding energy-aware adaptation [43,44,46]. By involving the applications he was able to scale the work done by the applications in order to achieve some goal (a particular battery lifetime). He notes that techniques like DVFS are complementary to adaptation and that many of the energy savings achieved via adaptation are only possible when used in conjunction with other dynamic power-management techniques. This work provides an adaptation API which allows applications to communicate their resource expectations and query the requested fidelity level.

Peddersen [114] developed a methodology for modifying an application to adapt its fidelity at run-time based on low-overhead, event-based energy measurements [113].

Other similar work includes Ellis's discussion of content specialisation for saving bandwidth and energy [38], Min's discussion of energy-scalable algorithms for sensor networks [105], and Narayanan's learning techniques for predicting the energy consumption of a workload at different fidelities [109].

2.8.2 Application hints

Much of this work has pointed to the need for accurate future predictions of energy and resource consumption. Policies requiring good workload prediction range from those managing disk spin-up to wireless network usage to processor usage. Most of the policies developed attempt to achieve transparency at the application

level. However applications may have knowledge of their future actions. Feeding this information into energy-management algorithms may improve the management of otherwise unpredictable workloads.

Flautner et al. [41] characterise application behaviour into episodes which are either interactive or periodic. This characterisation is based on analysing events associated with the operating system (e.g. system calls, inter-process communication). Knowledge of the nature of standard system components, such as the X server, is used to characterise the events. For example, communication with the X server is likely to be associated with an interactive episode. They further suggest that episodes could be accurately identified using an OS API. This option is attractive since it is not prescriptive: applications are not required to use these facilities, but may benefit if they do.

In a similar vein, Weissel [167] suggests that applications provide hints about the importance or nature of file operations. This information can be used to schedule of disk spin-up and spin-down. Applications which provide these hints are shown to use significantly less energy than the unmodified applications.

Papers regarding real-time scheduling have investigated the use of the most stringent application specifications, for example, specifying WCETs for particular operations or pieces of software. This is similar to the a-priori analysis required for hard real-time scheduling [120, 135].

Application hints are clearly relevant to our workload prediction scheme, and would help characterise erratic applications.

CHAPTER 3

MOTIVATION

> *"The most incomprehensible thing about the world is that it is comprehensible."* – Albert Einstein, 1936

DVFS has been the subject of a significant volume of academic research (as described in Chapter 2). Much of the work in this field, with several noteworthy exceptions, makes several assumptions about the way a system behaves when a single CPU frequency is changed (see Section 3.1). Following an extensive study of the behaviour of a number of platforms of different types (described in detail in Appendix A), we concluded that there were many situations where the platform's measured behaviour deviated significantly from the commonly assumed model. Consequently, many of the techniques for choosing DVFS settings that have been developed in both academic and industrial settings are less effective than would otherwise be expected. In the worst cases these schemes result in an *increased* energy use and cause a substantial loss in performance.

Instead, this thesis proposes that comprehensive models of a system are required to make effective power management decisions. A discussion of how real platforms deviate from the commonly assumed models forms the motivation for this work, and the bulk of this chapter. While we focus here on frequency and voltage scaling, this concept holds for active power management specifically, and power management in general.

3.1 The commonly assumed model

The commonly used model assumes that, for a given voltage, there is a constant constant charge consumed per clock tick. This is a reasonable model for a circuit in which exactly the same activity occurs on each cycle, under constant conditions.

When scaling voltage (in addition to scaling frequency), the charge consumed is assumed to be proportional to the voltage. Multiplying the charge per second (current) by the voltage to give power, we get

$$P \propto fV^2. \tag{3.1}$$

Previous research often assumes a constant number of cycles for a given computation. With this assumption, the performance is given by:

$$\frac{T^{max}}{T} = \frac{f_{cpu}}{f_{cpu}^{max}} \tag{3.2}$$

leading to the energy being represented as

$$E \propto V^2. \tag{3.3}$$

This implies that the most power-efficient (lowest power-per-work) and the most energy-efficient frequency (the least energy per work) are always the lowest — that slower is always more energy efficient.

The addition of a constant static power term improves this model. The static power is that which the system would use even if the clock stopped (0MHz). This power is consumed by peripherals, chipsets, and other circuits within the system which are unaffected by the CPU frequency. The system's power is then represented as

$$P = \beta_0 fV^2 + P_{static}. \tag{3.4}$$

Even an increased run-time can lead to a higher total energy, given a constant number of cycles and depending on the balance between static and active power.

3.2 Power management challenges

While the commonly assumed model can predict the power for a simple switching CMOS circuit, it can not accurately predict the power for microprocessors and, more generally, computer systems. The following sections outline the problems.

3.2.1 Workload dependence

The traditional model of a system when frequency depends only on the CPU frequency and voltage. In fact a system's response to frequency scaling is highly dependent on the running workload's characteristics. Clearly the power used by the system can change: a complicated instruction like `fdiv` will use more energy than a simple one like `nop`. Clock gating, a technique whereby sections of the processor are shut down when not in use, exacerbates this effect by saving large amounts of power in unused sections of a CPU.

Memory instructions present a compelling example: a memory subsystem's performance is often independent of the CPU frequency. Figure 3.1 shows the effect of memory latency when frequency scaling. Here we compare the responses of the CPU-bound `gzip` and the memory-bound `swim` benchmark on a Dell Latitude D600 laptop. The execution time of the CPU-bound program is proportional to the clock period (inverse of frequency), while the memory-bound program is almost independent of the CPU frequency. The result is clear: the total energy use for the CPU-bound benchmark is minimised by running at the highest frequency — the *race-to-halt* strategy — because this minimises the static energy used by the CPU [147]. In contrast, the memory-bound process uses the minimum energy a low (but not the *lowest*!) frequency. Clearly, a power management approach that does not take workload characteristics into account will not be able to deliver a reasonable result for both programs.

3.2.2 Multiple frequency and voltage domains

Several platforms which we evaluated allow the scaling of multiple frequencies. Among the systems evaluated there were variable CPU, memory and bus frequencies, as well as variable peripheral frequencies. This is most common in SoC systems such as Intel PXA processors [74, 75] and iMX-based platforms [50], but was also seen in more general purpose systems (the AMD Opteron-based server had a variable memory frequency). While no multi-core processors were trialled or modelled, these often allow independent control over the frequency of each core, and future processors are expected to provide independent voltage planes.

Different combinations of frequencies and voltages (which this thesis refers to

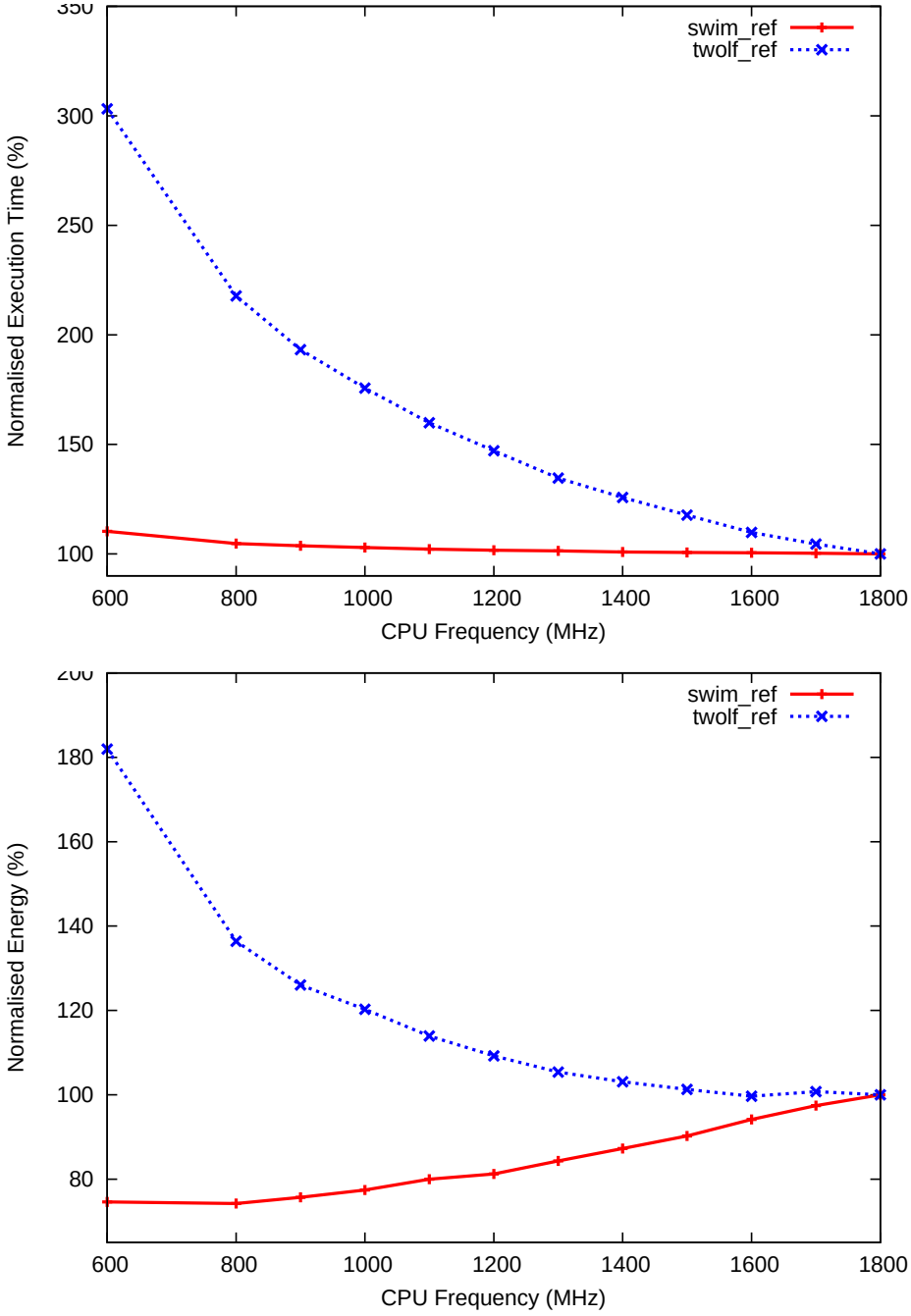


Figure 3.1: Normalised performance (top) and energy use of two benchmarks under DVFS on a Latitude laptop.

interchangeably as either *settings* or *setpoints*) lead to different results, as shown in Figure 3.3.

Often there are constraints on the allowed combinations of settings; on the PXA, for example, the system bus frequency must be half of the so-called run-mode frequency (the lowest frequency usable for turbo-mode switching), while the memory frequency is divided from the CPU core frequency with a small set of supported dividers [74].

On one platform — an AMD Opteron based server — the highest f_{mem} did not occur at the highest f_{cpu}. This can happen on platforms where the CPU frequency is locked at a multiple of the memory frequency, but the CPU's maximum frequency can not be exceeded. On this platform memory-bound applications actually ran fastest at a less-than-maximum f_{cpu}! (Coincidentally, our policies transparently handle this situation, switching between the optimal frequencies for maximum performance for each workload). This is shown in Figure 3.2, where the execution time is *minimised* at the third-highest frequency. For this system the highest memory frequency of 333MHz was achieved at a CPU frequency of 2.0GHz, whereas setting the highest possible CPU frequency of 2.4GHz resulted in a memory frequency of only 299MHz.This variation in memory frequency was not specified by the manufacturer, but determined through hardware measurement using an oscilloscope.

Figure 3.3 shows the execution time for `twolf` and `gzip` on a PXA270-based machine for different frequency settings. Note that `gzip` is memory-bound on this platform — the opposite of the same program running on a Pentium-M based Dell Latitude D600 and other machines with comparatively large caches, showing that the best power management policy is not only highly dependent on the workload, but on the platform as well!

The graph shows the impact of changing the memory and bus frequencies — lines connect points of equal memory and bus frequency. While the memory frequency has a clear impact on `gzip`, the CPU-bound application is unaffected by the memory frequency.

Scaling the memory frequency can result in energy savings. Figure 3.4 shows the energy required to run a range of workloads at the manufacturer-recommended frequencies for the PXA255-based PLEB 2 machine. These settings are docu-

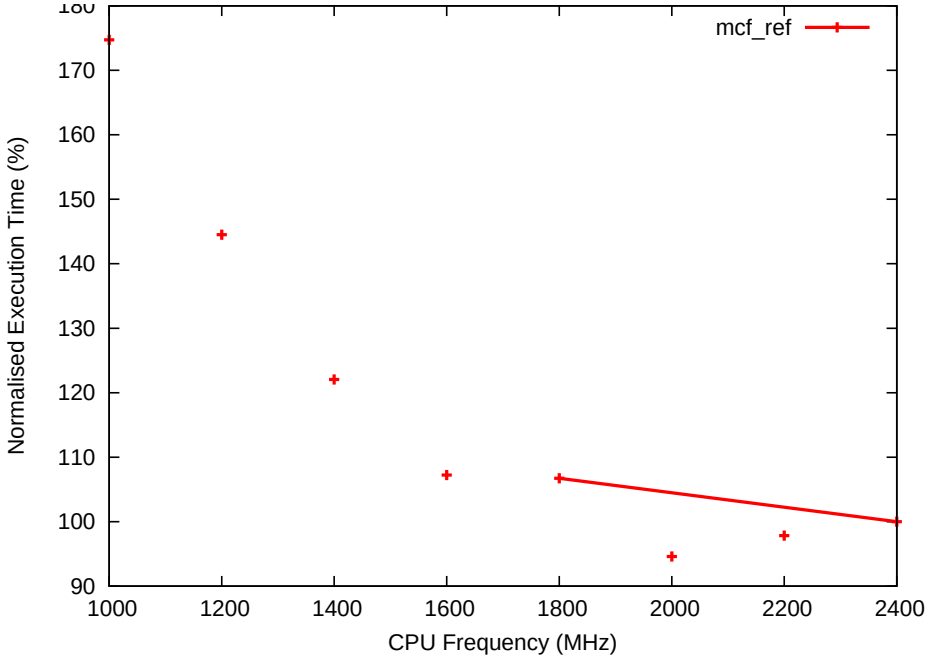


Figure 3.2: Execution time of a highly memory-bound application (`mcf`) under frequency scaling on an AMD-Opteron based server. Lines connect settings with the same memory frequency but different core frequencies.

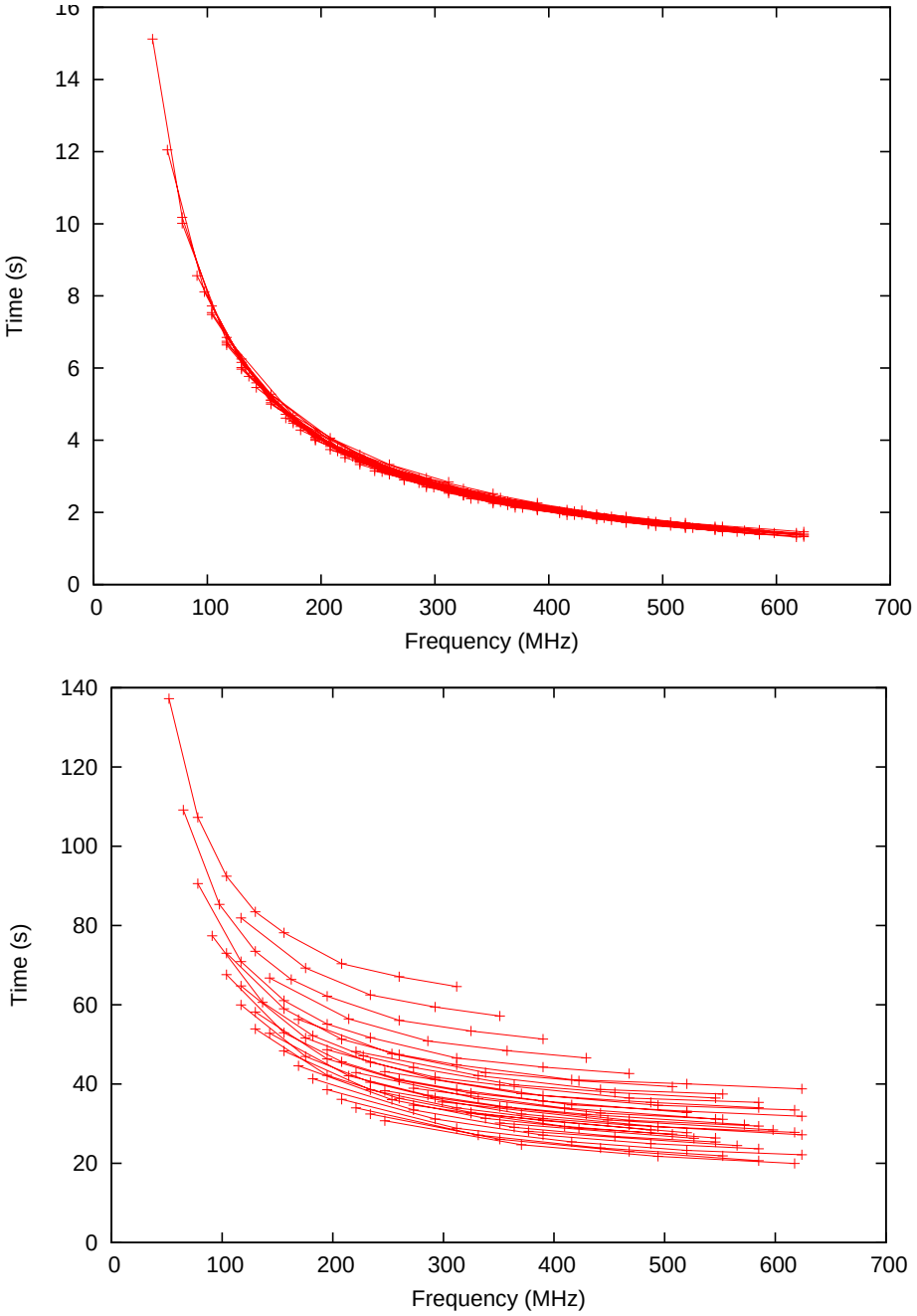


Figure 3.3: Performance of a CPU-bound (`twolf`) and memory-bound application (`gzip`) under frequency scaling on a PXA270-based platform. Lines connect settings with the same memory but different core frequencies.

V_{core} (V)	f_{cpu} (MHz)	f_{pxbus} (MHz)	f_{mem} (MHz)
1.0	99.5	50.0	99.5
1.0	199.1	99.5	99.5
1.1	298.6	99.5	99.5
1.3	398.1	99.5	99.5
1.3	398.1	199.5	99.5

Table 3.1: PLEB 2 DVFS settings

mented in Table 3.1. Lines connect samples of the same workload. At the highest CPU frequency, there are two potential bus frequencies. It can be seen that, at the highest CPU frequency, CPU-bound workloads use more energy at the higher bus frequency — presumably because the memory controller and other idle, bus-connector peripherals are using energy according to this clock — and memory-bound processes use more energy at the lower frequency — because of the effect the lower frequency has on performance. Voltage and frequency scaling schemes based on traditional heuristics ignore these opportunities and hazards.

Other hardware mechanisms for active power management reduce both performance and power in a similar way to DVFS. Examples include clock modulation and dynamically re-configurable caches. We can expect hardware designers to add new active management schemes in the future (particularly if the supporting software techniques presented by this thesis are available).

The availability of this multitude of throttling *knobs* means that active power management is not a simple single dimensional problem but a complex multi-dimensional optimisation.

3.2.3 Sleep modes

Reducing the frequency (and thus performance) of a system increases a workload's execution time and reduces idle time. Modern CPUs have low-power modes which the OS can invoke when the system is idle. Idle modes still consume power in most cases, and take time to enter and exit. Generally, the deeper, i.e. lower-power, the idle mode is, the more time it takes to enter and exit. This is usually because the system must save and resume state. Idle modes are common in pe-

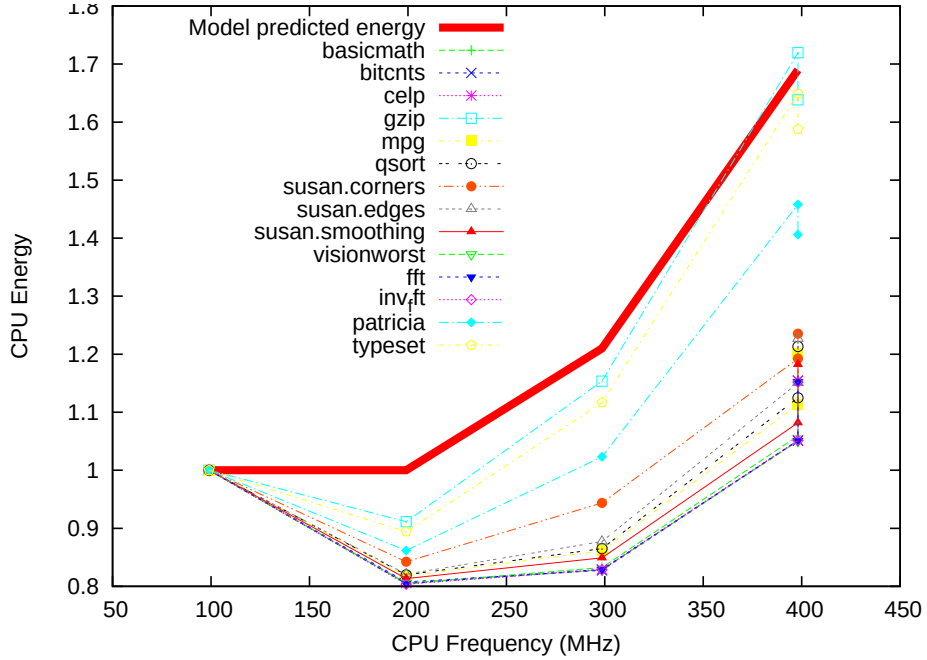


Figure 3.4: Normalised CPU energy for various workloads on a PXA255-based platform (PLEB2)

ripherals and I/O devices (e.g. disk spin-down), although the focus in this thesis has been on the CPU and memory idle power, with I/O power management left to future work.

Miyoshi and others have shown that the power consumed while idle must be taken into account to evaluate the energy-saving effects of a frequency scaling decision [107]. An analysis confirming this and is shown in Figure 3.5. It presents the total energy consumption when accounting for the idle energy used in various low-power modes on a laptop. For each benchmark, the energy consumed while actively executing is added to the energy which is used while the system is idle afterwards. The idle time is calculated as the difference between the run-time for the benchmark and the run-time for the same benchmark at the slowest setting (i.e. the setting which results in the least idle time). Besides the actual low-power modes supported in this system ("C states" in ACPI terminology), the figure also shows hypothetical 5W and 0W states (the active power is 22–30W). Again there is an obvious difference between memory-intensive and CPU-intensive processes,

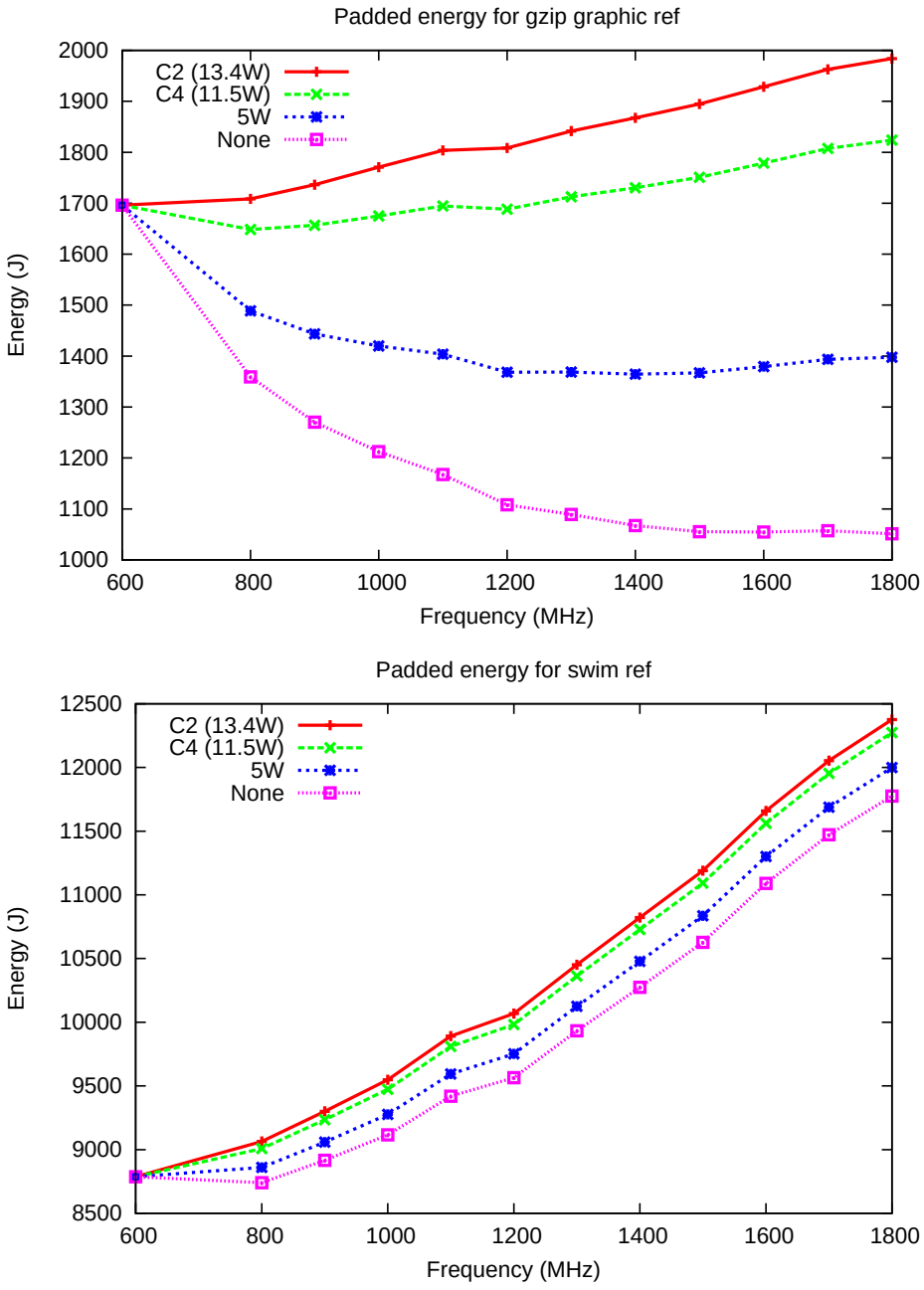


Figure 3.5: Total energy for the CPU-bound `gzip` and memory-bound `swim` applications on the Latitude, using different idle states.

although for the hardware-supported idle modes other than full shut-down, running at the lowest frequency always results in the lowest total energy use on this platform.

As the hypothetical states demonstrate, idle states of really low power consumption bias the frequency scaling decision (under a minimum-energy policy) toward the higher frequencies for a CPU-bound application, but make little difference for a memory-bound application. Note that with a zero-power idle mode, race-to-halt is sub-optimal for all but the most CPU-bound applications.

The idle modes available, and the ratio between the idle and active power, are highly platform specific. It depends on the CPU, system and peripherals, drivers, and how they interact. Embedded systems are designed for much lower power idle power than high performance computers and as a result are biased toward race-to-halt DVFS decisions. The Phycore iMX31-based system we investigated demonstrated this, with the minimum dynamic energy always occurring at some intermediate frequency for all workloads when idle modes are considered (Figure 3.6). On the AMD64 server the lowest frequency was always the most energy-efficient. Details of these platforms are available in Appendix A.

Estimating which idle mode the system will enter, if any, is a challenging task. In some circumstances, such as when the system is fully utilised, the system is never idle. In these cases, the extra CPU time mad available by running at a high frequency can be used usefully to do work, and so there is no penalty for running at a high-performance setting. This is not unreasonable in applications such as data centres, where consolidation can be used to maintain a high CPU utilisation, or high-performance computing, where, if there is no work to execute, the system will be shut down entirely.

3.2.4 Frequency-switching overhead

The time during which the CPU is unavailable during frequency switches varies considerably between platforms. Of the ones we tested it ranged from 10μs (a Pentium-M based Latitude, not including the voltage change which happens asynchronously) through 140μs worst-case for an Opteron to 500μ for PXA based platforms. This is pure overhead, since the machine consumes energy without

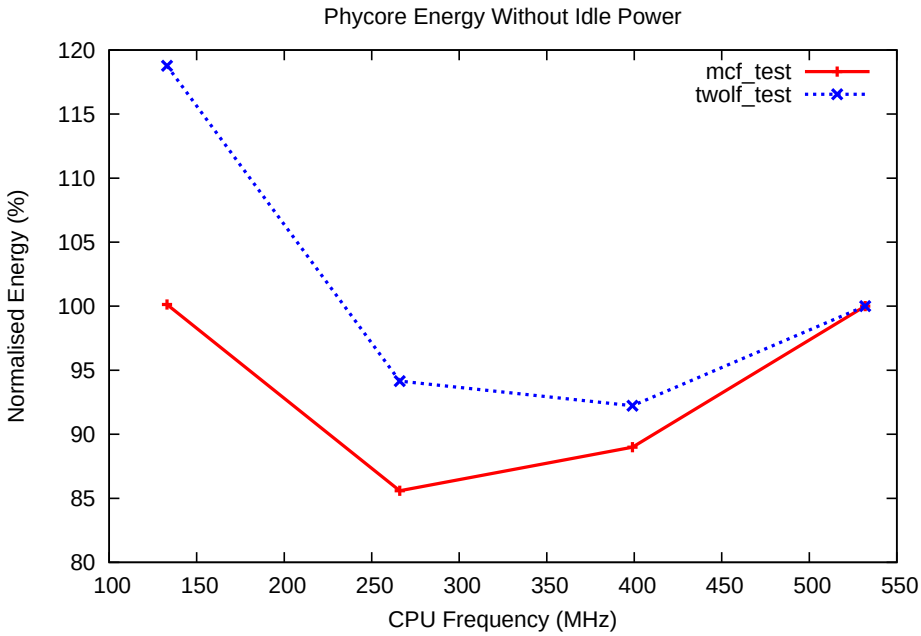


Figure 3.6: Dynamic energy for the CPU-bound `twolf` and memory-bound `mcf` workloads on the Phycore iMX31-based system.

doing any useful processing. Some other platforms examined exhibited interesting features: the PXAs take 500μs for a full frequency switch, compared with $\sim$20 cycles for a so-called "turbo mode" switch (the turbo-mode switch changes a divider on the CPU frequency, whereas a full switch requires a full PLL re-lock).

The overhead is due to two operations — the voltage and the frequency change. The overhead involved in these operations is highly hardware-specific.

On the Opteron, software controls the voltage ramps, and so the CPU is unavailable for the duration of the voltage switch. The Pentium-M is fully hardware-sequenced, and therefore the only CPU downtime is during the frequency change. Both of these platforms have a high-speed interface with the voltage regulator. The PXAs use an I2C bus to interface with the voltage regulator chip, and that bus is clocked at either 40kHz or 400kHz, depending on the particular implementation. The duration of communication with the regulator for a voltage change depends on the particular regulator chosen, since there is no standard command set. On PLEB 2 this communication takes about 100μs.

The frequency switch procedure also varies widely between platforms further varying the overhead. On the Opteron the voltage must be scaled to the maximum before the frequency switch can be performed and the voltage can be re-set to its final target. The period of CPU unavailability is then dependent on both the previous and next frequencies. For experimental purposes, the Opteron's DVFS driver was tuned to run faster than the specification, allowing the better-than-specified worst-case performance of 140μs. The worst case when run within the specification was $\sim$2ms.

To summarise, the overhead to switch between settings is highly platform dependent, and in many platforms it varies depending on the present setting and the target setting.

3.2.5 Temperature and cooling

Temperature affects the power used by electronic circuits. Resistors, capacitors, inductors and other components all change their physical characteristics with changing temperature. In computers, where *integrated circuits* (IC) are significant power users, the effect was noticeable in two ways: leakage current is related to the temperature of the silicon [177], and the power required for active cooling (a fan) is related to the speed at which the fan rotates (usually automatically controlled to prevent a device from overheating).

For a microprocessor, this implies that consistently running at higher performance setpoints (which cause the system to run at a higher temperature) will use more energy than predicted by the commonly assumed model. The relative power at different setpoints changes depending on the system's temperature.

This is shown in Figure 3.7, which plots the average power during many iterations of `gzip` running on a Dell Latitude D600 laptop. The system started from idle (i.e. at a low temperature). As the system increased in temperature due to the CPU activity associated with running `gzip`, the average power for each iteration also increased. Above a CPU temperature of 65 degrees the system automatically increased the speed of the cooling fan from its lowest speed to its medium speed. After further iterations of the workload (further increasing the CPU temperature), the system increased the speed of the fan to its maximum (again increasing the

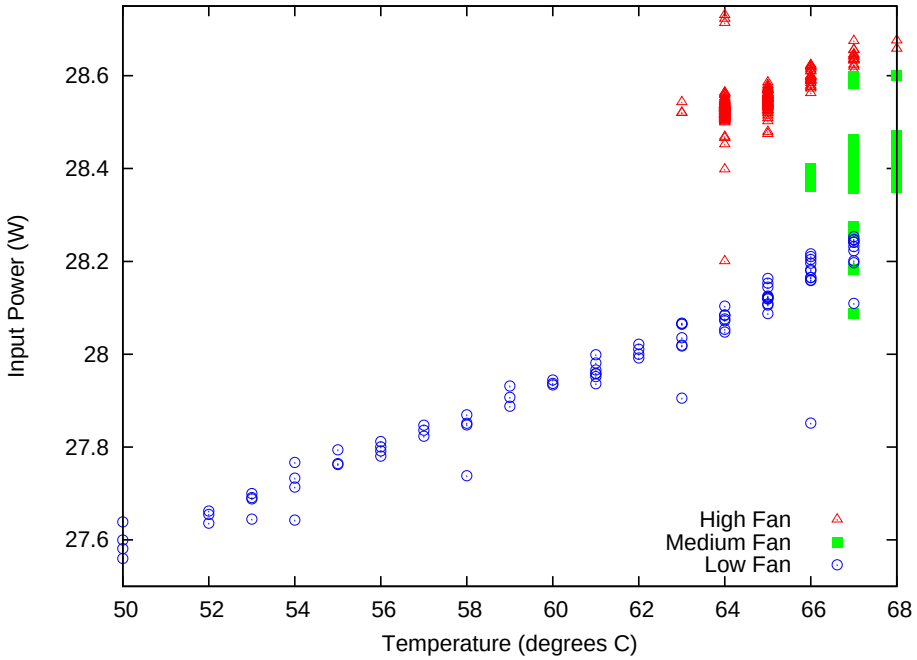


Figure 3.7: System power vs. temperature for gzip at 600MHz on a Dell Latitude D600

average power), which gradually reduced the temperature (reducing the average power). While the effect is minor (∼3.5%), the system's power is clearly effected by both the system's temperature and the fan speed, and it is conceivable that on other platforms or for the same platform in a harsher environment, such as an industrial system with wide ambient temperature variations, the effect could be more substantial.

This variation affects frequency scaling decisions, since it changes the ratio of active to static power, as well as the power consumed during idle modes.

3.2.6 Power supply efficiency

Battery efficiency

Batteries are ubiquitous in mobile and off-grid devices. They also present an interesting case of an inefficient power source. For nearly all battery types (chemistries), the available energy in the battery decreases as the rate of discharge

increases [98]. In simple cases this is because the batteries have an internal resistance which dissipates energy proportional to the square of the current drawn. In other cases, more complicated models must be used to estimate the losses, taking into account effects like battery recovery, where, if the battery is discharged in bursts, the effective capacity is greater than a single continuous discharge. In his PhD thesis and papers [98, 99], Thomas Martin examined battery models in detail, and subsequently discussed the effect of battery losses on optimal frequency selection. He showed that reducing the peak power for a system improves battery longevity more than an equivalent decrease in idle power. To achieve an optimum lifetime, the CPU frequency setting algorithms must take this into account. It suggests that running at a lower frequency improves battery lifetime *more* than the reduction in average power would imply.

Converter efficiency

Electronic circuits require well-defined and stable power supplies, but are supplied by unstable sources such as batteries (where the voltage varies with the state of charge), photovoltaics (where the maximum power available is highly dependent on the voltage at which it is drawn), or un-rectified AC transformers (which provide an alternating voltage). Switching DC-DC converters are often used to efficiently convert and regulate the power supplies. These converters switch an input supply through an inductor. The switching frequency and duty cycle are varied in order to control the output voltage to a stable value.

These regulators tend to have a high efficiency (>90%), being based on single or multi-phase buck converters. The losses for a given regulator are related to the voltages at the input and the output, as well as the current through the device. For a well-designed converter, the efficiency should be high throughout the expected power range. Therefore, in nearly all cases the power into the converter is very close to linearly proportional to the output power fed to the system and can therefore be accurately represented with a linear model such as the commonly assumed one.

However, on a Dell Latitude D600 laptop with a high CPU temperature, running a compression algorithm, *reducing* the CPU frequency would *increase* the

power drawn by the system. We traced this to the laptop's CPU core voltage regulator, which exhibited a step increase in efficiency as the we artificially increased the load power. Our experiment is shown in Figure 3.8: we added resistors in parallel with the CPU to draw a measurably load while keeping the rest of the system constant. The current to supply the entire system, and the current through the extra load resistors, were both measured. After adjusting for the usual losses in the converter, a clear step function in the efficiency is observed.

Examining the electronics, we discovered that the Latitude core supply uses an SC1476 IC [130], which is a dual phase buck converter controller. We could not obtain the datasheet, but similar ICs from the same manufacturer implement an energy saving scheme when moving from continuous to discontinuous mode, changing their switching pattern and explaining the observed step in efficiency. It is likely that the converter is designed to be efficient when running at the battery pack voltage ($\sim$11.1 V) rather than the AC adapter's ($\sim$19 V). When running from the battery the converter exhibits nearly constant, high, efficiency. For subsequent experiments we worked around this issue by supplying and measuring the current drawn from the battery rather than the AC adapter[1].

While well-designed systems will not exhibit wide variation in DC-DC converter efficiency, this type of effect must be considered in order to make accurate DVFS decisions in systems where this efficiency does change.

3.2.7 Variable memory-system performance

Microarchitectural effects

There are a large number of microarchitectural and compiler techniques which can lead to improvements in the memory subsystem's performance. Besides caching, these include out-of-order execution and pre-fetching. Compilers may use instruction scheduling to maximise the effectiveness of these features. Kotla et al. also noticed this effect [89].

The effect is to hide the latency of cache misses, since the memory accesses and subsequent cache miss are over-lapped with other instructions which do not

[1] This was complicated by the protection systems built into the battery.

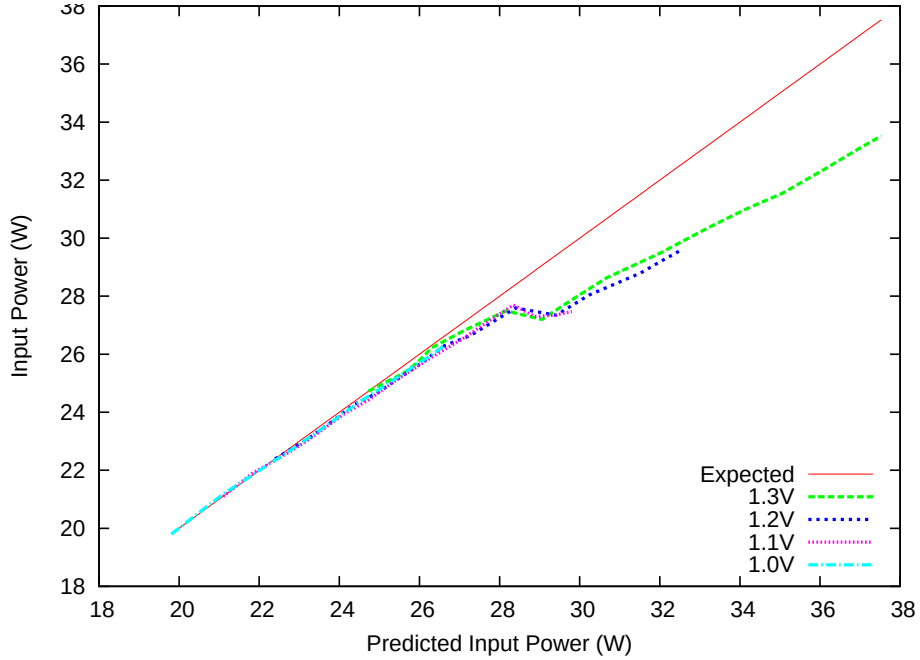


Figure 3.8: Actual vs. predicted input power for the Dell Latitude D600 laptop running from the AC adapter.

depend on the result of the access. In the case of instruction and data pre-fetching, this process happens transparently. Given these mechanisms, the execution time for a completely memory-bound workload should change minimally with CPU frequency. Any instructions which do not depend on an outstanding memory access will be executed before the end of the memory access — increasing the CPU frequency leads to a longer CPU stall while the memory operation to finishes.

However, in the case where two memory accesses are separated by many CPU instructions, it is possible that increasing the clock frequency will reduce the time between the memory accesses enough that the memory latency can no longer be hidden. In this case, these instructions will be CPU bound until the CPU frequency is high enough that the memory latency can not be hidden. When this occurs, the workload will become memory bound — further increases in the CPU frequency will not reduce the execution time. This implies that the memory latency for the workload changes with CPU speed. The same effect applies to pre-fetched data — at high CPU frequencies, the pre-fetching mechanism may not be able to pre-fetch

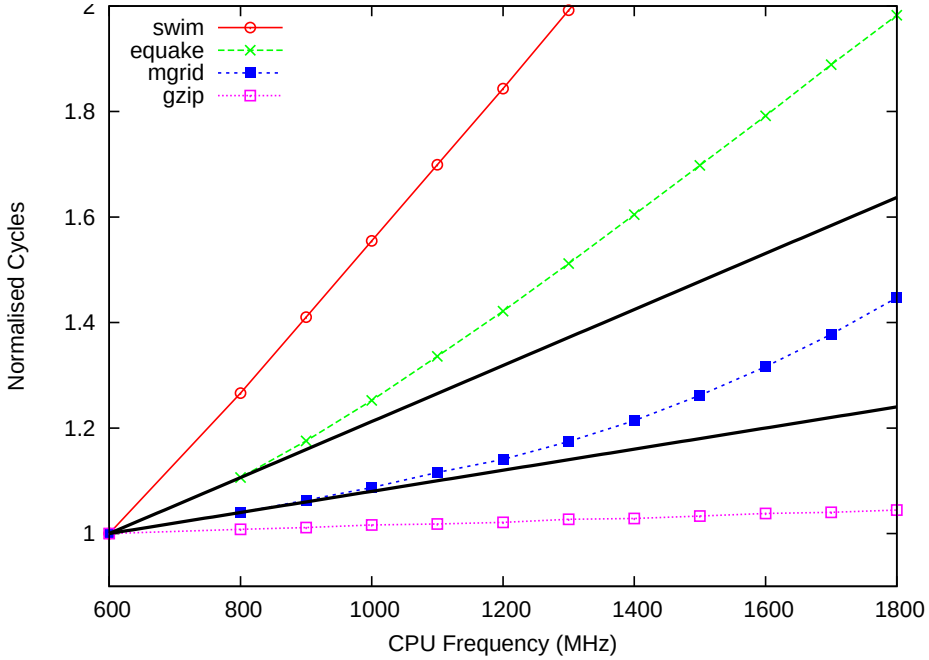


Figure 3.9: Cycles vs. Frequency for various benchmarks on a Latitude D600 laptop. Solid black lines show a linear fit to the 600MHz and 800MHz datapoints.

in time to prevent a stall.

The effect is clear in Figure 3.9. In a close-to-CPU-bound workload the number of cycles stays nearly constant, since the execution time only depends on the CPU clock rate (`gzip` in Figure 3.9, which uses a small number of memory instructions). For the close-to-memory-bound application (`swim` in Figure 3.9), the number of cycles grows linearly, since the execution time is only slightly dependent on the CPU clock frequency. However some benchmarks (`equake` and `mgrid`), grow non-linearly – they become more memory bound at higher frequencies, since the CPU can less effectively hide the memory latency by parallelising with CPU instruction execution.

This effect is clearly not considered by the commonly assumed models, and contributes substantially to the error in the models described in Chapter 4, since we could not find performance counters which could estimate the effect of frequency scaling on this parallelism. Accurate models for this effect are left to future work.

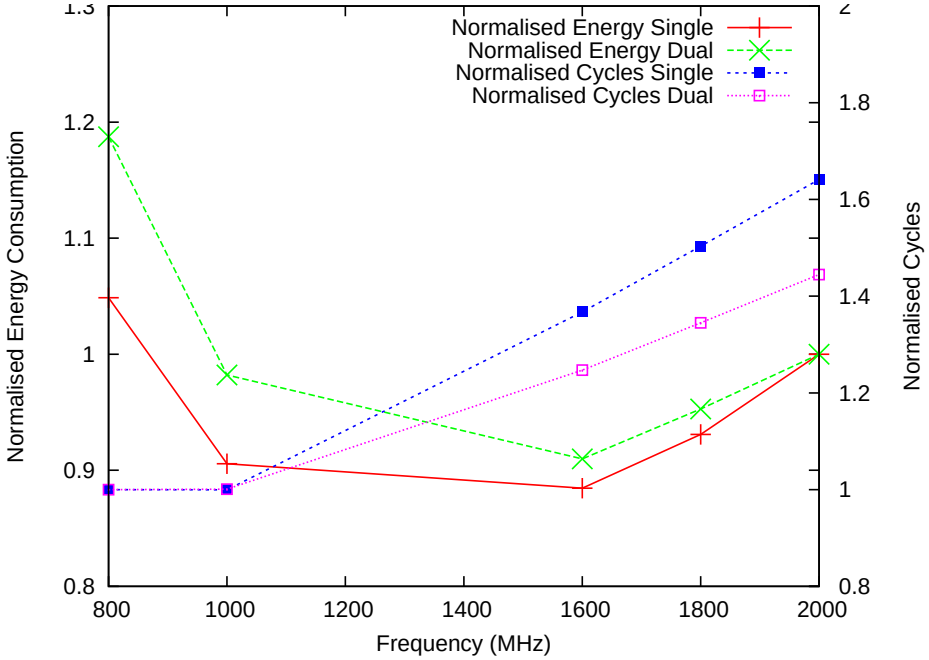


Figure 3.10: Comparison of cycles and energy use on an AMD64 Server with and without dual channel memory for `swim`.

RAM configuration

The memory performance of many machines depends on its configuration. Experiments with an AMD-Opteron based server, which uses dual channel DDR SDRAM, confirm this. Using a single DIMM in the machine gave single-channel memory operation, whereas adding a second DIMM enabled dual-channel operation. Figure 3.10 shows the effect on `swim`, a memory-bound workload. Running with the second DIMM reduces the number of cycles and decreases the energy used at the higher frequencies relative to the lower frequencies. Improving the memory performance by using dual-channel memory effectively reduces the memory-boundedness of this application. From a frequency selection standpoint, the decision is then biased toward the higher frequencies.

The irregular behaviour of the system at 800MHz is due to a reduced memory frequency at that setting (166MHz vs. 200MHz) – the memory-frequency to CPU-frequency ratio stays constant, and so the relative number of cycles also remains constant.

DMA and Multicore

Both *Direct-memory access* (DMA) and multicore have the potential to reduce the apparent memory performance via bus contention. Heavy bus contention due to either of these will result in longer memory latencies (on average), leading to more memory-bound workloads. In periods of heavy bus contention, the CPU frequency selection is then biased towards lower frequencies.

To limit the scope of this thesis, the effect of bus contention (and methods for predicting the bus contention based on predicted DMA and other off-core activity) are minimised and therefore ignored. This is considered future work.

3.2.8 Real-time dependencies

Some events in the system occur at a rate that is not related to the system's clock frequencies. Examples include the scheduler's clock ticks, I/O events such as USB polling, and other periodic tasks such as timeouts related to GUIs (e.g. a blinking cursor). In these cases the number of instructions is proportional to the time for which the system is active — if the system were to finish its task early and sleep, there would be less instructions to execute.

Scheduler clock ticks cause this behaviour: in a system with dynamic ticks, scheduler invocations do not occur while in idle modes. Therefore, running at a higher performance setting reduces both the number of scheduler invocations incurred, and the proportion of the system's active time spent processing those invocations. The number of clock ticks themselves contribute to the overall running time and energy, since each adds interrupt overhead as well as scheduler processing.

The relationship between the actual execution time (T_{tot}) and the interrupt-free execution time (T_{work}) is therefore related to the timer tick frequency f_{tick}, the CPU clock frequency (f_{cpu}) and the number of cycles taken to execute each timer tick (C_{tick}) as

$$\frac{T_{tot}}{T_{work}} = \frac{1}{1 - C_{tick} f_{tick} / f_{cpu}}. \tag{3.5}$$

A derivation of this relationship is given in Section 4.8.

3.2.9 Discrete frequency and voltage points

The hardware implementation details of DVFS mean that only discrete frequency points can be chosen. On all of the platforms examined in the context of this thesis, the various frequencies in the system were generated via one or more PLLs whose output was divided. In the case of a Dell Latitude D600, the CPU frequency could be set to 12 multiples of the memory frequency of 100MHz. On a Xeon server, only three frequencies: 2GHz, 2.33GHz and 2.66GHz were available.

The voltage settings are also limited to a discrete set. For example, the Intel Pentium-M based platforms examined define a power supply standard which allows the voltage to be scaled in 16mV steps. In most of the systems examined, the desired minimum CPU core voltage is linearly related to the frequency. In this context the discrete voltage steps lead to aliasing, since the voltage can not be set below the minimum for reliable operation. See Figure 3.11, which shows the CPU core voltage setting vs. CPU frequency for a Dell Latitude D600 laptop. Discontinuities can be seen between 1100MHz and 1200MHz, and between 1700MHz and 1800MHz. The effect can be seen in the corresponding irregularities in the relative energy of `equake_ref` between these frequencies. The same is true of our custom-designed PLEB 2 platform, where the voltage can be scaled in steps of 100mV (see Table 3.1).

3.2.10 Process variation

Manufacturing variation can cause the power to vary between otherwise identical designs. This is particularly true of microprocessors, where the small feature sizes have led to more variability. Unsal et al [155] outline a number of sources of variability in the microprocessor manufacturing process, as well as the impact which this has on the devices. Changes in the amount of energy used for computation, as well as the balance between the static, idle and active power in a microprocessor have the potential to effect frequency scaling decisions.

With only a single computer of each type, we were not able to observe this effect. A thorough study of manufacturing variation is well outside the scope of this

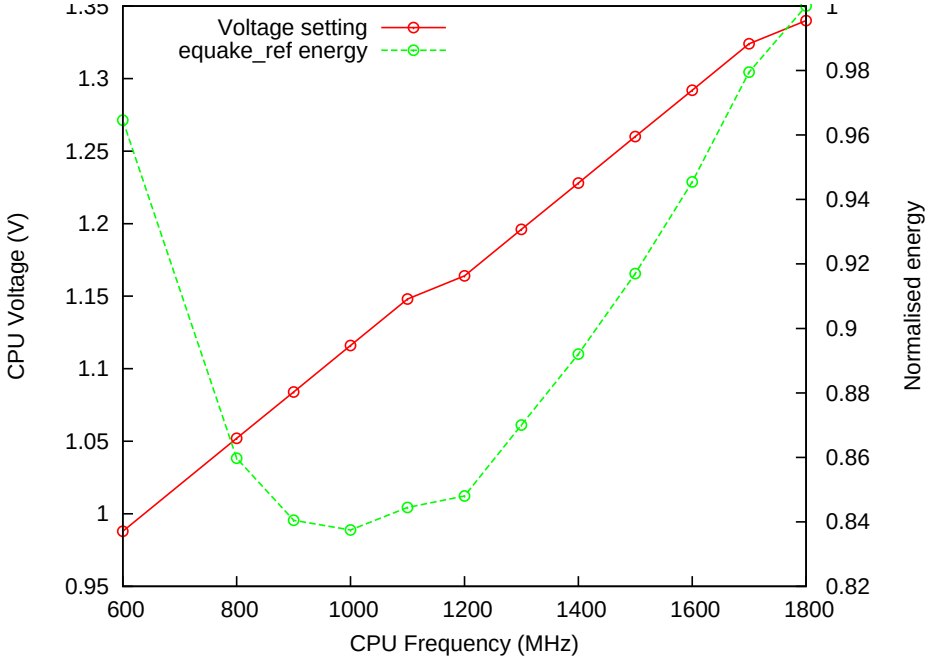


Figure 3.11: Actual voltage setting and relative energy for the frequency setpoints for `equake_ref` on a Dell Latitude D600 laptop

thesis, but the idea that a power management scheme must be customised, not just to the type of computer, but to the individual system, fits very well within it. For the remainder of this document, we consider power management schemes which are customised for individual computers, implicitly taking into account manufacturing variation. A future study might examine the difference in frequency scaling decisions between large numbers of computers of the same type to see what effect manufacturing variation has, and whether generalisations about optimal power management can be made.

CHAPTER 4

MODELLING

"It's not what you'd call a figure, is it?" – Twiggy, 1960s

As proven in Chapter 3, computer platforms do not behave according to commonly assumed models when frequency scaling. This chapter develops much more accurate models of a system's behaviour. These models predict the expected energy and execution time for a workload if it had been run at an alternate setting, taking into account the platform, workload, and environmental conditions.

The models are built using an off-line characterisation, and then used for online predictions.

While these models were developed in the context of, and for this thesis, they are largely a substantial improvement on previously-available techniques.

4.1 Terminology

Setting, Setpoint — a combination of throttling settings. For example, some particular combination of CPU voltage, CPU frequency and memory frequency on a system that supports these three hardware controls.

4.2 Assumptions

This thesis makes a number of assumptions for the sake of tractability. These assumptions limit the types of systems to which these techniques apply, and relaxing them will be core of our future work. The assumptions and limitations apply to these models — the policies described in Chapter 5 operate at a higher level, abstracted from the details of the hardware.

Firstly, I/O power is assumed to be constant, so it is just part of the static power. This assumes that I/O power is unaffected by power management, excepting the effect of idle modes (since in this case the static power is assumed to be different). This assumption is accurate for systems that perform very little physical I/O, but not for systems with power-managed I/O. In this situation, the time spent by I/O devices in their low-power modes will be effected by the speed of the system in servicing those devices. As a result, the power used by I/O systems should be taken into account. Detailed modelling and understanding of this interaction is beyond the scope of this thesis, having been addressed to some degree by prior work (for example, in ECOSystem [174]).

Secondly, the performance of the memory subsystem is assumed to be consistent. This limits us to systems where there is a constant level of bus contention. This is a major limitation, since bus contention is a very real possibility in systems with variable levels of DMA, or of those with multiple processors or cores connected to a shared bus.

We consider the efficiency of any DC-DC converters in the system to be constant for all currents. This is approximately true for a well-designed system. We also assume that there are very few interrupts, since these change our locality argument for workload prediction purposes.

4.3 Execution time model

The time taken to execute a workload is the result of waiting for several different components. These include the CPU instructions, the caches, the memory, I/O devices, and others. Ignoring the effect of some advanced architectural techniques (see Section 3.2.7, and Chapter 7 shows that this is often a reasonable simplification), these components can be considered not to overlap — i.e. in most cases the system completes an outstanding memory operation before moving on to more CPU instructions. In the case of IO, the CPU usually executes a different instruction stream while waiting for the I/O device to return.

Each of these different uses of the system's execution time (here we use CPU instructions, cache accesses, memory accesses and I/O accesses as examples) depends on different clocks. In most systems the CPU instruction execution and

some cache accesses are scaled directly with the system's CPU clock. In some (such as the IMX31 SoC), the lower levels of the cache hierarchy are connected by busses which can be scaled independently of the CPU frequency — the delay for an L2 cache access then depends on both frequencies. The delay due to memory accesses is related to the memory frequency — if the memory and memory controller both run twice as fast, the delay is halved. These *components* of the over-all execution time *do* behave according to the commonly assumed models, even though the complete system does not.

We can observe from the above that the execution time for a given set of instructions depends on the performance of the subsystems which those instructions use. Workloads with small cache footprints will be dependent on the CPU frequency alone, while those which suffer significant numbers of cache misses will depend on the performance of the memory subsystem. Those without I/O will not depend on the delays incurred waiting for I/O.

For systems with multiple frequencies, the interrupt-free execution time can be modelled as:

$$T = \frac{C_{cpu}}{f_{cpu}} + \frac{C_{bus}}{f_{bus}} + \frac{C_{mem}}{f_{mem}} + \frac{C_{io}}{f_{io}} + \dots \tag{4.1}$$

The coefficients $C_{cpu}, C_{bus}, C_{mem}, C_{io} \dots$ depend on the instruction stream of the actual workload as well as the platform. Modelling the execution-time comes down to obtaining good estimates for those parameters at run-time, without *a priori* analysis of the particular workload. Since we are considering I/O effects to be future work this thesis will focus solely on the CPU and memory subsystems. For many systems, frequencies such as f_{mem} are held fixed, and can therefore be amalgamated into some static execution time which depends on the delay incurred in those components. For generality, we consider the case where there are multiple clock frequencies.

4.3.1 Application characterisation

The coefficients C_x depend on the workload, representing the total number of cycles used for particular actions (e.g. CPU-only instructions or memory reads);

each is the product of the number of such actions and the cycle cost of such an action. The former is generally a characteristic of the workload, the latter of the system architecture[1].

To estimate the workload's characteristics, we use run-time measurements taken by *performance monitoring counters* (PMCs). Although PMCs are the most convenient source of information about a workload's characteristics, there is no reason that information from other sources could not be incorporated into these models if relevant. Each coefficient in Equation 4.1 can be represented by some linear combination of PMC readings:

$$C_{bus} = \alpha_1 PMC_1 + \alpha_2 PMC_2 + \ldots \quad (4.2)$$

$$C_{mem} = \beta_1 PMC_1 + \beta_2 PMC_2 + \ldots \quad (4.3)$$

The accuracy of the model will depend on the architecture and the suitability of the PMCs (or other measurable statistics).With specific hardware support, an exact model would be possible, and larger numbers of simultaneously monitored events may improve the system's accuracy by better measuring the workload characteristics.

The architecture-specific coefficients α_x, β_x, ... are properties of the hardware platform and can be determined once, by evaluating a suitable set of benchmarks representing the range of different workloads. Selecting the appropriate parameters is important, as the hardware typically supports the concurrent use of only a small number of PMCs, and the correct choice is not obvious. The process of parameter selection and characterisation is discussed in Section 4.10.

All of the hardware examined was able to directly measure the total number of CPU cycles, C_{tot}. It is the product of total execution time, T, and core frequency, f_{cpu}. We can then rewrite Equation 4.1 as

$$C_{cpu} = C_{tot} - \frac{f_{cpu}}{f_{bus}} C_{bus} - \frac{f_{cpu}}{f_{mem}} C_{mem} \quad (4.4)$$

Consequently, we only need to determine C_{bus} and C_{mem} from PMCs... Not C_{cpu}.

[1] One exception is in the case of a dynamically sized cache — the number of external memory accesses is then dependent not just on the workload, but the cache size setting.

Some PMCs measure time in terms of the CPU frequency. Even if the time measured is constant across frequencies, the measured value will not be. For these events, the time can be calculated as

$$\frac{PMC_x}{f_{cpu}}. \tag{4.5}$$

4.3.2 Performance prediction

Since we are able to estimate a workload's C_x values from the PMC readings at a particular setpoint (characterised by a particular combination of clock frequencies, f_x) it is possible to predict the performance of the same workload at a different setpoint with frequencies f'_x:

$$\begin{aligned} C'_{tot} = C_{tot} & \\ -\frac{f_{cpu}}{f_{bus}}C_{bus} - \frac{f_{cpu}}{f_{mem}}C_{mem} & \\ +\frac{f'_{cpu}}{f'_{bus}}C_{bus} + \frac{f'_{cpu}}{f'_{mem}}C_{mem} & \end{aligned} \tag{4.6}$$

We define the performance, s, as the execution time at a particular setpoint normalised to execution time at maximum frequency setpoint, f_x^{max}:

$$s = \frac{t^{max}}{t'} = \frac{f'_{cpu}}{f^{max}_{cpu}} \times \frac{C^{max}_{tot}}{C'_{tot}} \tag{4.7}$$

For the measured setpoint, $f'_{cpu} = f_{cpu}$ and $C'_{tot} = C_{tot}$, the latter being read directly from the CPU's cycle counter. Hence, the performance at the current setting is a linear equation of performance counter and frequency cross terms. This allows a single regression to be applied across all workloads to calculate α_x and β_x, given a pre-calculated s avoiding the intermediate step of C_x.

$$s_{cur} = \frac{f_{cpu}}{f^{max}_{cpu}} \times \frac{C^{max}_{tot}}{C_{tot}} \tag{4.8}$$

Similarly, we can calculate performance of another setting relative to the cur-

rent setpoint using C'_{tot} and f'_{cpu} in place of C^{max}_{tot} and f^{max}_{cpu}. Both of these are linear equations, since both frequencies and C_{tot} are known or measured.

4.3.3 Summary

This model addresses the challenges presented in Section 3.2.1 (workload dependence) and Section 3.2.2 (multiple frequency domains). However, with the performance counters available on the hardware evaluated in Chapter 7, we were not able to measure the effect of microarchitectural techniques (Section 3.2.7) on the execution time. The RAM configuration (Section 3.2.7) is considered to be held constant — i.e. if the RAM configuration changes, the system must be re-characterised. For systems with multiple RAM configurations, this could be taking into account by modelling how the architecture specific components (α_x, β_x, ...) change with different setups. Herein, the system is considered constant. As stated, we do not consider systems where bus contention can change the performance of the memory subsystem (Section 3.2.7). Again, this effectively varies the architecture-specific constants, and would require a prediction of the future bus contention. This is left to future work. Real-time dependencies were not observed, or have a linear relationship and are modelled by the above. Process variation does not have an effect on the system's performance.

4.4 Basic Energy model

The same reasoning can be used to predict the energy used during the execution of a program under different power management settings: individual components of a system behave according to the commonly assumed model, but the over-all system does not.

Under the assumptions in Section 4.2, Equation 2.6 holds for the *active* power in a given component of the computer executing a given workload. The active power being the frequency and voltage dependent component of the power. The *active* energy consumed during a time interval Δt is then

$$E_{active} \propto fV^2\Delta t. \tag{4.9}$$

If the time interval is expressed in CPU cycles, $cyc = f\Delta t$, this becomes

$$E_{active} \propto cyc \times V^2. \tag{4.10}$$

This energy corresponds to the energy $E = \frac{1}{2}CV^2$ of a capacitor C, it represents the energy used to charge and discharge the circuit's capacitances (such as the gate capacitance on a transistor) during each clock cycle.

We cannot assume that the active energy is workload independent in this way. In the case of a CPU, the number of transistors switched on each clock cycle depends on the instructions being executed. The level of activity external to the CPU, such as memory accesses and some I/O, is also dependent on the workload.

Instead, a system's activity can be divided into a number of *events*. The active energy associated with each of these events can be estimated using simple models similar to the commonly assumed model. Events might include different types of instructions, cache references, TLB misses, etc. These are the behaviours that make up the system's operation. There are potentially hundreds or thousands of different types of events which are impractical to account, so we must accept some inaccuracy in our model by only accounting for events which make a significant difference to the energy consumed.

In order to obtain detailed information about which events occur, we again use PMCs. Modern processors have performance counters which can be used to count a number of different types of events, some of which are correlated with power [17, 165].

The energy consumed during a time interval Δt can be modelled as a linear combination of the measurable events in the system. Each system clock generates events at some frequency (which may or may not be variable), and the number of cycles is $f_x\Delta t$. For a typical system with m available event counts (all being measured as PMCs) and a single CPU voltage domain (V_{cpu}):

$$\begin{aligned} E = {} & V_{cpu}^2(\gamma_1 f_{cpu} + \gamma_2 f_{bus} + \gamma_3 f_{mem})\Delta t + \\ & V_{cpu}^2(\alpha_0 PMC_0 + \ldots + \alpha_m PMC_m) + \\ & \gamma_4 f_{mem}\Delta t + \beta_0 PMC_0 + \ldots + \beta_m PMC_m + \\ & P_{static}\Delta t, \end{aligned} \tag{4.11}$$

where PMC_i is the event count of performance monitor i during the interval Δt, and α_i, β_i and γ_i are the coefficients of the model which are determined via off-line characterisation. Note that f_{mem} occurs twice, once with a V_{cpu}^2 factor and once without. The reason is that this frequency represents the memory bus, which interfaces with an on-chip memory controller. In this model, the bus and controller are supplied by V_{cpu}, but the memory chips are supplied by V_{mem}, which is constant.

The rate r_i of the event measured by counter i is $r_i = \frac{PMC_i}{\Delta t}$, so the system power can be expressed as

$$\begin{aligned} P = {} & V_{cpu}^2(\gamma_1 f_{cpu} + \gamma_2 f_{bus} + \gamma_3 f_{mem}) + \\ & V_{cpu}^2(\alpha_0 r_0 + \ldots + \alpha_m r_m) + \\ & \gamma_4 f_{mem} + \beta_0 r_0 + \ldots + \beta_m r_m + P_{static} \end{aligned} \tag{4.12}$$

This equation can be used to predict the power at one frequency setting from measurements obtained at a different setting, similar to Equation 4.6. Note, however, that the execution-time model is also required for either the power or energy prediction, since we must predict the changes in the performance-counter rates. Substituting one equation into the other is straightforward but the resulting formulas are unwieldy, which is why they are not presented here explicitly.

Provided that enough of the relevant events can be measured, Equation 4.12 can predict the system power for a given workload at an alternate setpoint. Combined with the model of execution time of Section 4.3, we can then predict the system's energy usage with alternate settings. Importantly, the coefficients α_i, β_i and γ_i should be independent of workload characteristics and only depend on the hardware. They can therefore be determined once for each platform.

A methodology for selecting the appropriate events to measure is discussed in Section 4.10.

4.5 Idle energy model

When the system is idle, it uses energy. In order to reduce the power drawn while idle, most hardware supports the use of *idle modes*. These shut down sections of

the system which are not useful in idle and can range from $0W$ when the system is shut down to nearly the full system power for a tight `nop` idle loop. Choosing appropriate idle modes when the system is idle is orthogonal to this thesis. However, the effect of idle time on the over-all system energy can be estimated and its effect on frequency scaling decisions taken into account. Note that, while extremely important for choosing which idle mode to use, the time and energy used for entering and exiting a mode does not affect frequency switching decisions – the time and energy used during these transitions is considered constant.

The model presented in Section 4.4 estimates the energy used to execute a given set of events, but does not take into account the static power used when the system is idle. In some circumstances (e.g. in I/O bound systems), increasing the system's performance increases the amount of time spent idle. In other circumstances (e.g. in well-consolidated data centres or high-performance computing where the system is virtually guaranteed to be fully utilised resulting in very little idle time) the system is never idle, so there is no idle power/energy [2].

Two models were considered. The first estimates the total system energy including the idle energy consumed by running at a higher-than-minimal performance setting. It models the energy used during any extra idle time and adds it to the over-all energy estimate. This can be represented as

$$E_{total} = E_{active} + (T_{max} - T)P_{idle}. \tag{4.13}$$

where E_{total} is the total energy, including the idle time, and E_{active} is the energy spent while in an active state, rather than an idle state. In the case where the system becomes fully-utilised at some non-minimum performance setting, the execution time at that performance setting should be used in place of that at the minimum performance setting (T_{max}) above. E_{total} is represented by the shaded area in Figure 4.1.

Alternately, we can calculate the *dynamic* energy. Instead of calculating the total energy required to run a workload, we calculate the extra energy required to run it, assuming that otherwise the system would be idle. Since the idle power

[2] In fact in these circumstances nodes in the systems are completely powered down when idle, which amounts to the same thing.

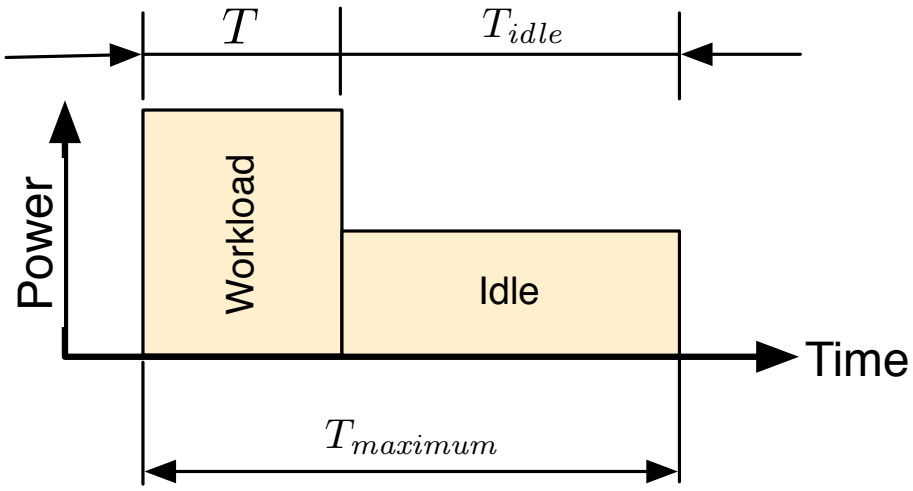


Figure 4.1: Using padding to calculate the energy used by a workload for a greater-than-zero-power idle mode.

is not changed via DVFS (idling at a particular frequency is in effect a new idle mode) that power should be subtracted from the average power for the benchmark. This can be represented as

$$E_{dyn} = E_{active} - P_{idle}T \tag{4.14}$$

where E_{dyn} is the extra energy required to run the benchmark. The dynamic energy is represented by the shaded area in Figure 4.2.

For the purposes of optimisation the two are equivalent, since

$$E_{dyn} = E_{total} - T_{max}P_{idle}, \tag{4.15}$$

and we consider T_{max} and P_{idle} to be constant for a given workload and platform. Since the second technique does not rely on knowledge of T_{max}, it is used for on-line use and evaluation in Chapter 7. Note that, in systems where idle time is present, the delay used for *energy-delay product* calculations can be considered constant, allowing the use of the dynamic energy model with this policy.

The above equations assume a knowledge of the idle power. On many systems the power and transition latency for idle modes are stored by the manufacturer in

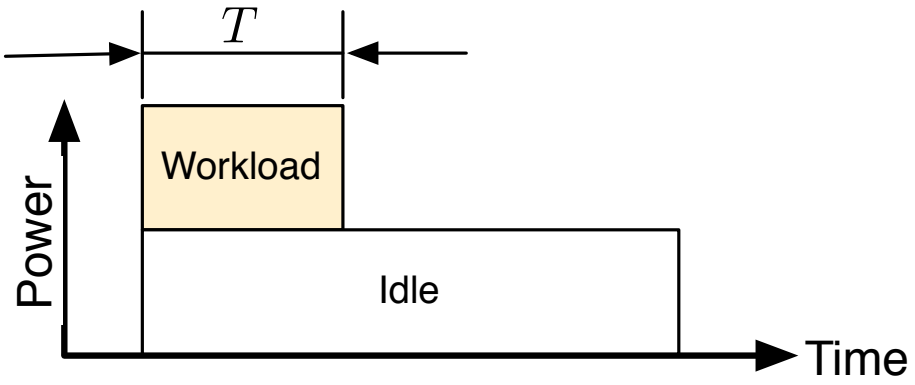


Figure 4.2: Using idle power subtraction to identify the extra energy required to run a workload.

ROM (usually accessible via ACPI [66]). When this information is not available, or the system is in a different state from that measured by the manufacturer (e.g. using a different screen brightness setting), or the manufacturer-provided information is not trustworthy, the idle powers can be easily characterised. We run benchmarks that put a system with no foreground activities to sleep for varying periods of time. The OS uses some policy for choosing idle modes as a function of sleep time. We use the OS's accounting of time spent in various sleep states, and used linear regression on the average idle power

$$P_{ave} = \sum_i \frac{T_{C_i}}{T_{total}} P_{C_i}, \tag{4.16}$$

where T_{C_i} is the time spent in idle state C_i, $T_{total} = \sum_i T_{C_i}$, and P_{C_i} the power drawn in state C_i.

Predicting how much time will be spent in each idle state when estimating the expected idle power is left to future work. This thesis considers only a constant idle power.

4.6 Temperature and fan effects

Some platforms, such as the Dell Latitude D600 (details of which are in Chapter 7), have a thermal sensor which can be read by the operating system. As shown in Section 3.2.5, this effects the power drawn by the system.

There are two main effects. The first is the leakage power, which is super-linear with respect to temperature [177] and linear with voltage [27]. This component can be approximated by a polynomial in the region of interest and modelled as

$$E_{temp} = \beta \tau_n V \Delta T + \ldots \tag{4.17}$$

where β is a constant coefficient determined via calibration, τ is the temperature as read by sensor n, and ΔT is the time over which the energy is being calculated.

The system's CPU fan consumes power when running, and consumes more power when running at a higher speed. A detailed examination of the power used by these fans is out-of-scope here, and the power was simply presumed to be modelable as a linear function of the fan speed.

$$E_{fan} = \beta F_n \Delta T + \ldots \tag{4.18}$$

Where F_n is the rotational speed of fan n and β is a calibrated constant.

4.7 Switching overheads

As discussed in Section 3.2.4, the overhead when changing settings limits the frequency at which the settings can be changed. This overhead is highly platform specific, but there are usually overheads due to both the voltage and frequency switch. Some hardware allows the processor to run while performing the switch. For all of the platforms we examined, the processor stayed in a high-power state for the duration of the switch, so there is an associated energy.

On a Dell Latitude D600 laptop, the overhead model is trivial — a constant $10\mu s$ overhead per switch, with a corresponding quantum of energy used. This

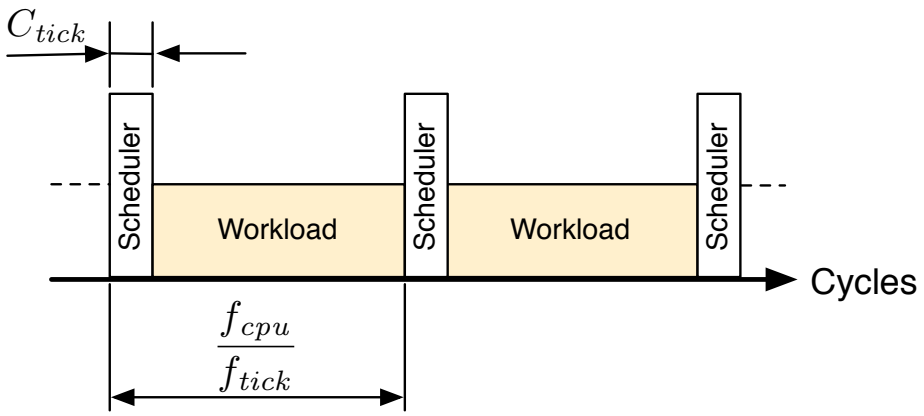


Figure 4.3: Comparing real-time events and CPU-time events

is because both the voltage and frequency change are sequenced by hardware, allowing the system to keep running during much of the the process. On the Opteron-based server we examined, a more complex model is required because more of the voltage and frequency sequencing must be done in software. In particular, the voltage must be ramped to the maximum voltage before the frequency switch occurs and the voltage is gradually ramped back down, with a generous delay while the PLL re-locks. We model this delay as

$$T_{switch} = \frac{2V_{max} - V_{cur} - V_{next}}{V_{step}} T_{step} + T_{lock}, \tag{4.19}$$

where V_{step} is the size of the step in the ramp, T_{step} is the latency incurred at each step and T_{lock} is the PLL locking delay. Again, we make the approximation that the power remains constant during the switch, and therefore that the energy used is proportional to T_{switch}.

4.8 Real-time dependencies

As discussed in Section 3.2.8, the time to execute a workload can depend on events which occur according to the wall-clock time (which proceeds at a constant rate) rather than CPU time (which is affected by the clock scaling).

An example of this effect is an operating system's regular timer tick. In this

situation the clock ticks themselves contribute to the overall running time, and the number of ticks executed depends on the real time, rather than the CPU time. This is shown in Figure 4.3, where f_{tick} is the timer tick frequency, f_{cpu} is the CPU clock frequency and C_{tick} is the number of cycles for the processing the tick (running the scheduler). The time between timer ticks is then

$$T_{tick} = \frac{1}{f_{tick}} \tag{4.20}$$

and the number of CPU cycles is

$$T_{tick} \times f_{cpu} = \frac{f_{cpu}}{f_{tick}}. \tag{4.21}$$

The number of CPU cycles available to run the workload is

$$\frac{f_{cpu}}{f_{tick}} - C_{tick} \tag{4.22}$$

and, on a system with a single adjustable CPU frequency, the interrupt-free execution time (T_{work}) will be extended to a total execution time including timer tick overhead (T_{tot}) according to

$$\begin{aligned} \frac{T_{tot}}{T_{work}} &= \frac{f_{cpu}/f_{tick}}{f_{cpu}/f_{tick} - C_{tick}} \\ &= \frac{1}{1 - C_{tick} f_{tick}/f_{cpu}}. \end{aligned} \tag{4.23}$$

Note that in this equation, T_{work} is also dependent on the CPU clock frequency and therefore affected by DVFS.

Lastly, consider the extra work done by the CPU and the energy used as a result of a longer execution run. The total number of CPU cycles for a CPU bound workload is constant, which means that as the CPU frequency is increased, the execution time decreases, and a smaller number of timer interrupts occur for that workload. The ratio between useful work and work for the scheduler is increased, and the number of cycles spent in the scheduler for that workload is decreased because the workload is executed in fewer timeslices. In a memory-bound process, the ratio of useful work also increases, but the number of cycles spent executing

the scheduler decreases. Expressed algebraically, the total number of extra cycles spent executing the scheduler ($C_{tick,tot}$) can be calculated as

$$C_{tick,tot} = C_{tick} f_{tick} T_{tot}. \tag{4.24}$$

For the operating system and platforms evaluated in this thesis, $f_{cpu} \gg C_{tick} f_{tick}$ and so the effect was negligible and ignored.

4.9 Measurement-based estimation

While the majority of these models use performance counters to estimate the system workload and power use, there is no reason that other sources of information about the system's operation can not be used. Some systems (particularly our custom-designed PLEB 2) have sensors which enable the measurement of the system's own power consumption. In the case of PLEB 2, this provides an on-line measurement of the CPU, memory and IO subsystems' power.

Real power measurements present the ability to provide a more accurate model for frequency scaling. While the measured power, by its nature, encompasses thermal variation and other difficult-to-model features of a system's behaviour, the measured power alone is insufficient to estimate the power consumed at a different setting. To perform that estimation, we propose a hybrid model where real measurements are used to estimate components of the system power which are difficult to model.

To demonstrate this, the following models consider a system with a single power measurement, and a single CPU frequency/voltage. In this system, the energy can be split into two domains: the energy consumed in the core (which scales with the CPU voltage), and the energy consumed in other circuits. This can be expressed as

$$E_{tot} = E_{core} + E_{other}. \tag{4.25}$$

If we define the energy used in the core as that which scales with the square of

the core voltage, then we can estimate what the energy is at an alternate voltage setting via

$$E'_{tot} = \frac{V'^2}{V^2}(E_{tot} - E_{other}) + E'_{other}. \tag{4.26}$$

From this equation, if we can estimate E_{other} and E'_{other} with accuracy, it is possible to estimate E'_{tot}. E_{other} can be estimated using similar techniques to those discussed above. Alternately, if an accurate estimation of the active energy E_{active} can be made, and P_{other} can is assumed to change slowly, then the reverse is possible — P_{other} can be measured, and the dynamic energy E_{active} can be estimated using performance counters. The system energy can then be estimated as

$$E'_{tot} = E'_{active} + \frac{T'}{T}(P_{tot} - P_{active}). \tag{4.27}$$

Clearly finer-grained power measurement will allow for more accurate models. For example, in the above models, the energy used in the core voltage domain alone could be measured.

We tested this model during the evaluation of this thesis, and anecdotally it improved the accuracy of the models substantially. The only platform with appropriate current measurement hardware was PLEB 2, and this was barred from a thorough investigation due to its frequency switching overhead. The implementation of this technique therefore remains future work.

4.10 Parameter selection and Characterisation

We have presented a general algebraic form for the models. This section presents a method is required for choosing and characterising models for a specific platform. The steps are:

- **gather** possible data, from a representative set of workloads running at every power management setting;
- **formulate** models including all potentially relevant terms based on this collected data;

- **select** a subset of the terms which provides the best predictive ability using an exhaustive search technique;
- **characterise** the model using further data collected using the selected parameters.

The following sections outline these steps.

4.10.1 Gather

This method assumes that the models are linear equations which can be fit using least-squares regression. If the models become non-linear (for example, if a variable power supply efficiency is significant), then alternate methods must be employed. Our evaluation in Chapter 7 shows that for the platforms we tested linear models work well.

All measurable information about a platform are considered as possible parameters for the models. All platforms investigated in the context of this thesis had performance counters which were capable of measuring a large number of different events, but only capable of measuring a small number of events simultaneously. For example, each of the PXA270's four PMCs can be configured to measure one of a set of 14 different events [76]. The Pentium M used in the Dell and IBM laptops could select from hundreds of different events, but only had two counters [71]. Some platforms, like the Phycore iMX31-based platform had more than one performance monitoring unit [50] — one internal to the core (the performance metrics unit), and one for the level 2 cache (the ARM11 Event Monitor) — each with a small number of counters able to measure a selection from a larger group of events. All of the performance counters examined could be accessed within a small number of cycles.

In addition to hardware performance counting units, other statistics were available on some platforms. For example, on our custom-designed PLEB 2, we implemented power measurement for the CPU, memory and IO power supplies. The Dell latitude laptop had a temperature sensor reading available via a BIOS call.

In order for the models to estimate accurately for a large variety of workloads, it is necessary to build them using a comprehensive set of benchmark workloads.

These benchmarks should represent the behaviours of real software. During evaluation, these exhibited a number of unexpected behaviours which were then examined using synthetic benchmarks (in particular, the effect of some microarchitectural optimisations as discussed in Section 3.2.7).

Given a set of benchmarks, the parameters to be considered for use in the model are measured at each of the different power management settings to be considered. Since it is not not possible to measure all of the parameters simultaneously, a benchmark must be executed multiple times with the same setting under identical conditions in order to build a complete set of measurements.

The data set must be sufficiently varied so as not to introduce co-linearities into the model. For example, if the ambient temperature is constant, the CPU temperature becomes a reasonable predictor for the CPU power.

4.10.2 Formulate

Given a data set including all of the potentially relevant parameters we can formulate a model which contains all of these parameters. We develop two equations — one for the relative performance and one for power (Equation 4.12).

It is important that each of these is normalised, so the weighting for each sample in a regression is naturally similar (i.e. a benchmark with a long execution time does not have more influence than a benchmark with a short execution time). Also note that each of these equations is linear in terms of parameters which are calculable from the data set.

Equation 4.7 can be combined with the total cycles model (Equation 4.6) and the external cycles models (Equation 4.2 and Equation 4.3), while ignoring the minor effect of Section 4.8 to give

$$\begin{aligned} \frac{T'}{T} &= \alpha_1 PMC_1 \frac{f_{cpu}}{f'_{cpu}}\left(\frac{f'_{cpu}}{f'_{bus}} - \frac{f_{cpu}}{f_{bus}}\right) + \ldots + \\ &\quad \beta_1 PMC_1 \frac{f_{cpu}}{f'_{cpu}}\left(\frac{f'_{cpu}}{f'_{mem}} - \frac{f_{cpu}}{f_{mem}}\right) + \ldots \end{aligned} \tag{4.28}$$

or, more simply, building the model in terms of CPU cycles

$$\frac{C'_{cpu}}{C_{cpu}} = \alpha_1 PMC_1(\frac{f'_{cpu}}{f'_{bus}} - \frac{f_{cpu}}{f_{bus}}) + \ldots + \beta_1 PMC_1(\frac{f'_{cpu}}{f'_{mem}} - \frac{f_{cpu}}{f_{mem}}) + \ldots \quad (4.29)$$

and, in the simple case where there is only one adjustable frequency

$$\frac{C'}{C} = \alpha_1 PMC_1(f'_{cpu} - f_{cpu}) + \ldots \quad (4.30)$$

In this last instance, as in the previous equations, the available data can be used to calculate both the left hand side and each of the coefficient multiplicands for a given source and target setting.

The model should be capable of predicting the performance at any setting given a sample taken when running in any other setting, so the regression matrix for this model is set up with an entry for each unique combination of source and target settings.

The effect described in Section 4.7 is only observed when dynamically switching, and therefore is not examined as part of this model.

The power model is easier to formulate, since Equation 4.12 is already a linear equation in terms of the measured data.

Importantly, unlike previous work [25, 68], neither model implicitly requires a particular parameter other than the dedicated CPU cycle count. The selection described below is therefore free to choose any event.

Lastly, for both the performance and power models, combinations of selected performance counters may be used to generate candidate terms in the model.

4.10.3 Select

Once a model has been formulated with a complete set of candidate terms we choose the minimal set of relevant terms to maximise the predictive power of the model. Comparing R^2 is not a good indication of the prediction power when the number of parameters is changed, since adding a parameter can not reduce

R^2. For this purpose the *bayesian inference criterion* (BIC), a statistical criterion which takes into account the number of parameters in the model, is a much better indicator.

We perform a parameter selection using an exhaustive search of all n-parameter models. Each n-parameter model is fit to the calibration data using least squares regression. The model with the highest BIC (although R^2 gives the same selections) is then chosen as the best model for n. While we were able to perform this operation in reasonable time for up to 50 parameters, well known search algorithms exist for narrowing the search space for larger numbers. This procedure was performed using the `regsubsets` command in `R` [125].

The parameters thus selected provide the best model for the calibration workloads used and the performance counters and other events available. Provided there exists a suitable set of events, and the calibration workload is representative enough, this method will find performance and power models that can be used to manage energy consumption.

4.10.4 Characterise

In Section 4.10.1 we introduced some inaccuracies in the data by combining the data from multiple benchmark runs. This was necessary to measure all of the selectable events, but is the correlation between those event measurements is effected by variation in each benchmark execution. To reduce the errors we generate a new dataset using new experiments. In these experiments we record only the selected parameters, and so each benchmark run records all of the appropriate data in a consistent way. This also allows for the system's environmental conditions to be varied where appropriate: for example, adding temperature. A least-square regression is conducted using this final dataset. An example of a model generated during the course of our evaluation is shown in Listing 1. It is a python script which stores the name of each event which is used in the model (the `params` dictionary). It also stores the equation used to calculate each term, and the coefficient for that term, in each of the time and power models (the `tmodel` and `pmodel` dictionaries).

Listing 1 `amd64-server.py` — a model generated for the AMD-Opteron based server.

```
### Model file automatically generated using model_builder.py
### from data.tsv
### using settings from ../../settings/amd64_original
### Written by David Snowdon, 2008
### http://www.snowdon.id.au

params = ['event0x0043016d', 'event0x0043027e',
          'event0x004300d5', 'event0x004304e0']

# Time model. R^2: 0.986175096027
tmodel = {'I(event0x004300d5 * (fcpu_t - fcpu)/tsc)': -0.00016080055125034337,
          'I(event0x004304e0 * (fcpu_t - fcpu)/tsc)': 0.037217639686974675,
          '(Intercept)': 1.0012823274516816,
          'I(event0x0043027e * (fcpu_t - fcpu)/tsc)': 0.03266298846952452,
          'I(event0x0043016d * (fcpu_t - fcpu)/tsc)': 0.0022052346778683187,
          '(SwitchLatency)' : 1}

# Power model. R^2: 0.988144777044
pmodel = {'I(vcpu_t * vcpu_t * event0x0043016d/T_t)': -0.0027479480096717608,
          'I(fcpu_t)': -0.0075424311483866377,
          'I(vcpu_t * vcpu_t * fcpu_t)': 0.013076558844243242,
          'I(event0x0043027e/T_t)': 0.35942701259072207,
          'I(event0x004304e0/T_t)': 0.28372354879294365,
          'I(vcpu_t * vcpu_t * event0x0043027e/T_t)': -0.26097675589316371,
          '(Intercept)': 72.912188254936765}
```

CHAPTER 5

POLICY

> *"I always avoid prophesying beforehand, because it is a much better policy to prophesy after the event has already taken place"* – Winston Churchill, ∼1945

The models discussed in Chapter 4 provide estimates of the energy, power and performance for a workload at any setting. These results can inform an energy management decision, but do not define the policy via which that decision is made. That policy is the subject of this chapter.

Since the platform-specific mechanisms of how to predict the effect of a decision are encapsulated in the models, the policies are platform agnostic.

5.1 Low-level policy

Given two power management strategies, along with the energy, power and time associated with those two strategies, which one is the best? If one policy leads to less energy use, less power and improved performance, the decision would be trivial. In the case where some energy is saved, but some performance is lost, this decision is less-so.

5.1.1 Trivial Policies

This thesis considers several potential fixed policies – those where the goal remains absolute and un-changing. The most trivial of these is the `maximum performance` policy, which is used to give the user the best quality of service possible, ignoring the energy effects of doing so. The system always chooses the highest performance setting — in the case of a single CPU frequency, this is

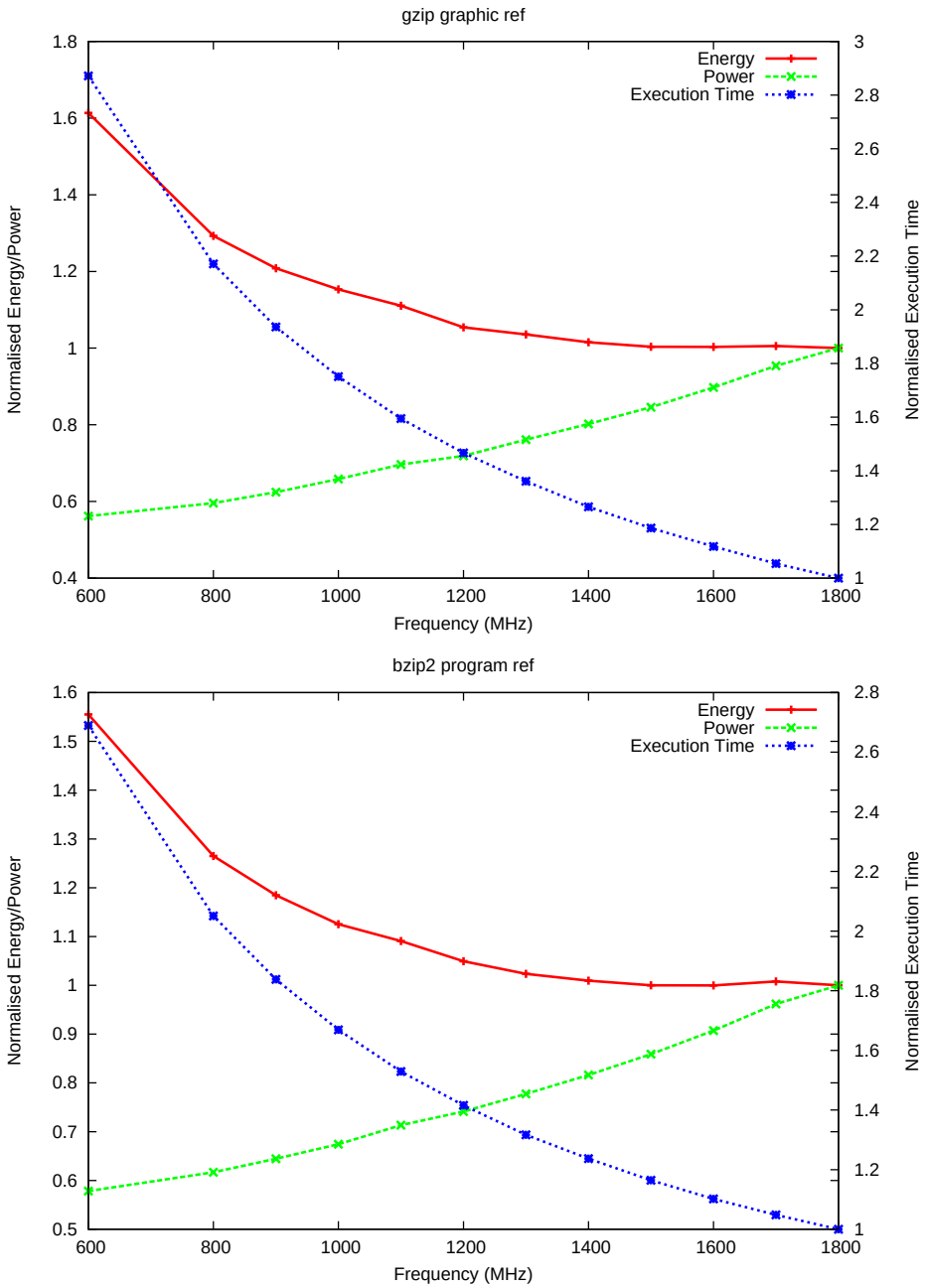


Figure 5.1: Power, Energy and Execution Time, all normalised to the maximum frequency, for CPU-bound (top) and semi memory-bound (bottom) workloads on the Latitude laptop

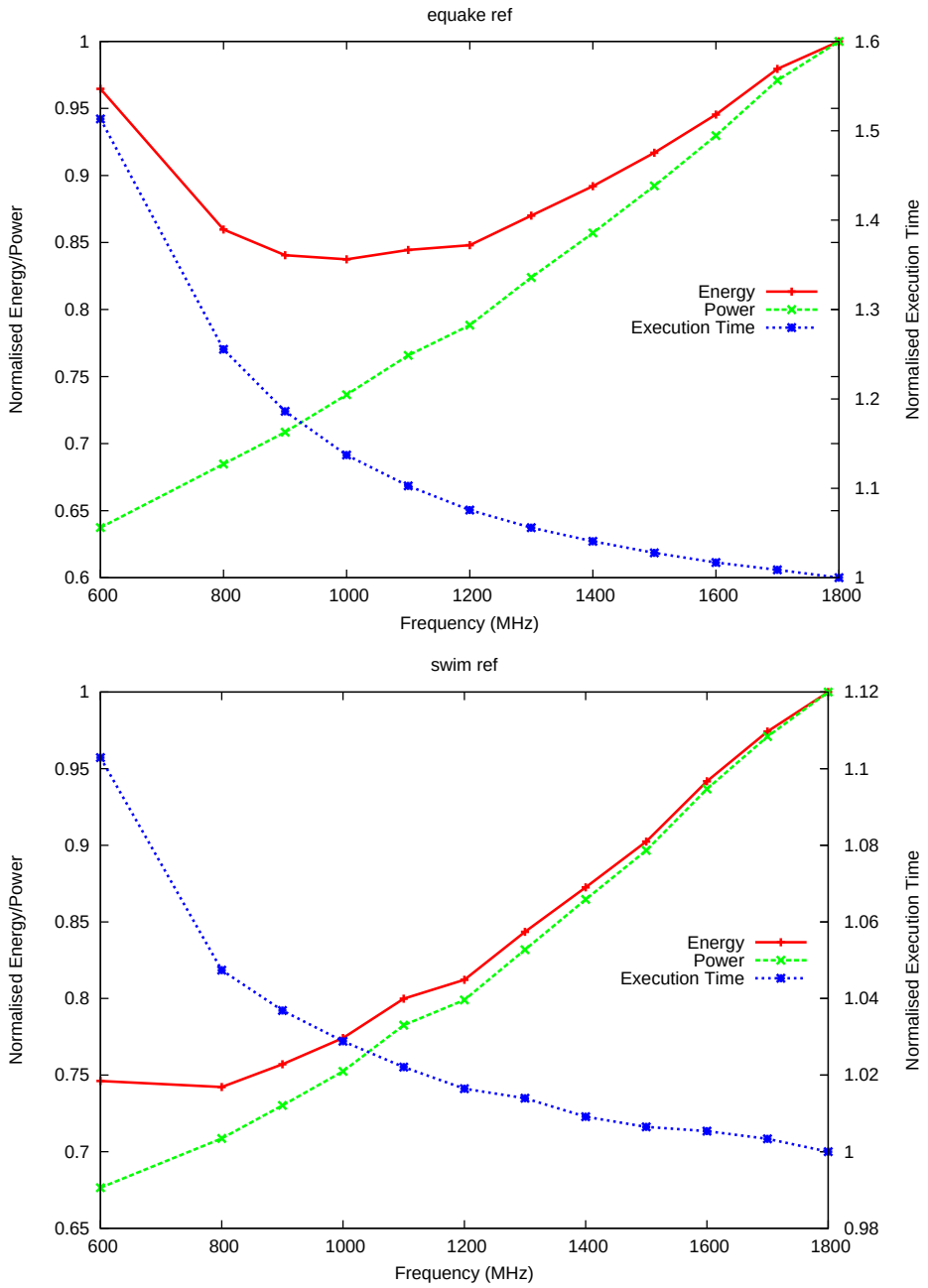


Figure 5.2: Power, Energy and Execution Time, all normalised to the maximum frequency, for semi memory-bound (top), and fully memory-bound (bottom) workloads on the Latitude laptop

always the highest available frequency. In all systems which were tested that had multiple adjustable frequencies, there was a single setting under which all of the benchmarks ran with a minimum execution time. On one platform, an Opteron-based server not surveyed as part of this thesis, we noticed that the highest f_{mem} occurred at the third-highest f_{cpu} — memory-bound benchmarks run fastest at a less-than-maximum f_{cpu} — see Section 3.2.2 for further details. In Figures 5.1 and 5.2, the maximum performance is observed at the highest frequency of 1800MHz.

Similarly trivial on all platforms tested was the *minimum power* policy, which chooses the setting which minimises the power. Again, on all systems tested there was a single configuration which gave the lowest power for all benchmarks. The intent of this policy is to reduce the thermal dissipation in the processor. In Figures 5.1 and 5.2, this is the lowest frequency of 600MHz.

The lowest power setting is almost never the lowest energy setting, thanks to the changed execution time of the workload. In energy-cost constrained systems such as data centres, and battery-lifetime constrained systems, minimising the energy used is important, leading to the *minimum energy* policy. In Figures 5.1 and 5.2, the minimum energy setting varies due to the workload. For the most CPU-bound application (`gzip`) the minimum energy is achieved at the highest frequency. For the most memory-bound benchmark (`swim`), it is at the second-lowest.

Figures 5.1 and 5.2 illustrate a serious issue with the *minimum energy* policy. For some workloads (such as bzip2 in Figure 5.1), while there are some very small energy savings available by scaling to a lower frequency (0.01% @ 1600MHz), the performance impact of doing so is comparatively high (10%). Not only may the quality of service of the scaled workload suffer, but in a fully-utilised multi-tasking system, CPU time will be taken from workloads where scaling may have a more significant energy saving effect.

The *minimum energy-delay* policy counters this effect to some degree by minimising the product of the workload's energy and execution time. When comparing two settings, if the system suffers an $n\%$ performance drop, then to be selected, the second setting must save more than $n\%$ of the energy. While this policy provides a balance between the performance and the energy, it is inflexi-

ble: it is impossible to guarantee the maximum system performance with such a policy, and impossible to obtain the maximum energy savings when performance is of no concern. Instead, we would prefer a policy which can be tuned for all circumstances.

5.1.2 Bounded performance degradation policy

One policy which has seen substantial attention from the literature [25, 26, 165] is what we term *bounded performance degradation*. The policy scales the system settings such that the performance is as close to some limit as possible. That limit is often empirically chosen as 90%, but could be dynamically adjusted depending on the performance required of the workload (based on QoS or real-time style feedback). This policy is designed to take advantage of the memory-boundedness of workloads whose performance reduces only marginally as a result of frequency scaling. A *bounded performance degradation* policy leads to memory-bound processes running at lower frequencies, where they save more energy.

Figures 5.3 and 5.4 show the effect of applying this policy when scaling each of the four workloads shown in Figures 5.1 and 5.2. The figures are generated off-line based on the measured data for the entire workload. The measurement methodology and the effect of applying this policy on-line at a fine-grained level is presented in Chapter 7.

The policy saves some energy, depending on the performance bound selection. For highly memory-bound workloads like `swim`, the policy aggressively scales the workload to achieve significant energy savings of over 25%, for a minor performance penalty (the maximum-energy savings are achieved with a performance bound of 90%, as in prior work). However, even for the highly memory-bound `swim`, the lower performance bounds do not lead to a minimal energy consumption.

For CPU bound workloads such as `gzip_graphic` and `bzip2` the optimal setting is 100% performance for all goals except for power minimisation. Reducing the performance bound increases the energy used while decreasing the actual performance of the workload. While the energy increase may only be minor, for high performance bounds, the decreased performance is guaranteed.

Workloads which are somewhat memory-bound, such as `equake`, require a different performance bound to minimise the energy use — between 85% and 87%.

This highlights both the advantage and drawbacks associated with the *bounded performance degradation* policy: the performance bound is selectable. This lets the user or a higher level policy adjust the apparent performance of the system depending on their needs (such as meeting real-time deadlines), but the performance bound must be carefully selected for each workload to achieve an energy-related goal. The policy is a better energy-saving heuristic, but is unpredictable, and must be workload-aware for good results.

5.1.3 Generalised energy-delay policy

Instead of relying on heuristics, this thesis developed the *generalised energy-delay* policy and applied it in the context of DVFS. Given an estimated power (P) and execution time (T) for a given workload at various settings, the policy attempts to minimise the quantity

$$\eta = P^{(1-\alpha)}T^{(1+\alpha)}, \tag{5.1}$$

where alpha is a parameter that can be varied between -1 and 1. Since $f(x) = x^2$ is a monotonically increasing function for positive x, special choices of α result in each of the trivial policies. Specifically:

$\alpha = 1$ maximises performance (forces highest frequency)

$\alpha = 0$ minimises energy ($E = PT$)

$\alpha = -1$ minimises power consumption

$\alpha = 1/3$ minimises the energy-delay product [1] ($ET = PT^2$).

Other values map to other policies used in the literature [115, 120]. Our approach allows the OS to easily adapt the power-management policy to changed operating conditions.

This policy decision is entirely dependent on the power and time estimates. Given accurate estimates of these quantities, the system will choose the best set-

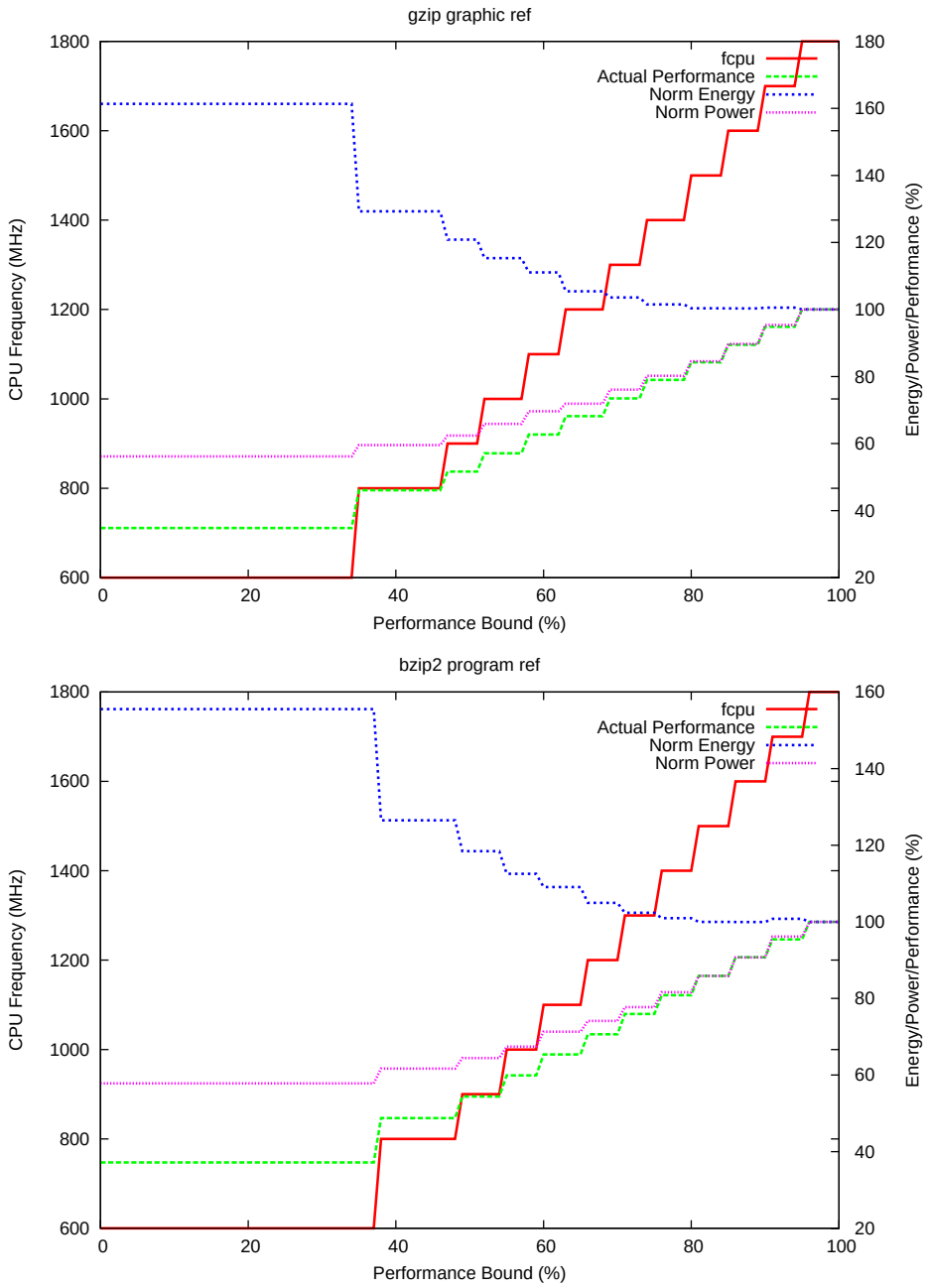


Figure 5.3: Behaviour of the *bounded performance degradation* policy for a CPU-bound (top) and lightly memory-bound (bottom) workloads on the Latitude Laptop

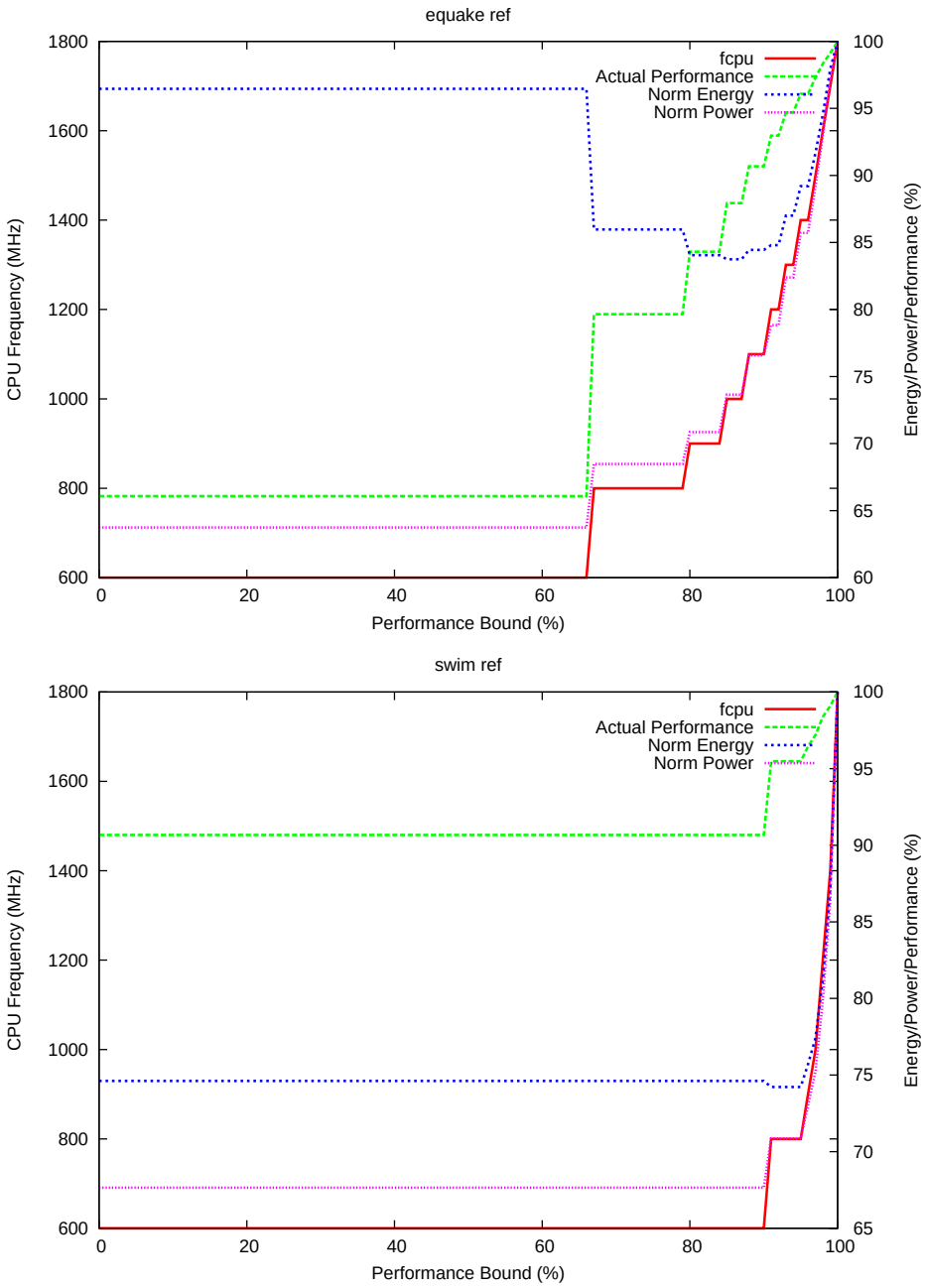


Figure 5.4: Behaviour of the *bounded performance degradation* policy for medium memory-bound (top) and heavily memory-bound (bottom) workloads on the Latitude Laptop

ting for the outcome expressed by α. This is not a heuristic — the value of α expresses a precise trade-off between performance, energy and power.

Figures 5.5 and 5.6 show the effect of applying the *generalised energy-delay* policy to the same workloads and platform shown in Figures 5.1 and 5.2.

For all four benchmarks, the goals above are achieved at their respective settings of α. For `gzip_graphic`, the performance is kept at 100% between $\alpha = 0$ and $\alpha = 1.0$, since between those values, no energy savings can be made for this workload. Below $\alpha = 0.0$, the policy chooses lower frequencies to reduce the power, at the expense of both decreased performance and increased energy. For `bzip2`, the frequency is only reduced at the minimum energy setting, and not above, since no substantial energy savings can be made. Contrast this with both `equake` and `swim`, where the frequency is reduced substantially between $\alpha = 1.0$ and $\alpha = 0.0$ in order to take advantage of the energy savings available for a small performance penalty. For all workloads the maximum performance setting is chosen at $\alpha = 1.0$, and the minimum power setting is chosen at $\alpha = -1.0$, but in-between the performance, energy and power are traded in a way that is specific to both the workload and the platform.

Contrast the use of α with the use of the performance bound discussed in Section 5.1.2. α achieves the desired goal in a workload and platform agnostic way — α means the same thing, and achieves the same goal, on each workload and platform. This is in contrast to the *bounded performance degradation* policy, which requires an intimate knowledge of the workload's behaviour on a platform to achieve these goals.

In summary, this policy provides us with the tools required to trade performance, energy and power between two or more potential settings, given a power and performance estimate for those settings. It does this in a reliable, predictable way and is abstracted from the behaviour of the benchmark or the platform, providing a single tunable which governs the balance between performance and energy.

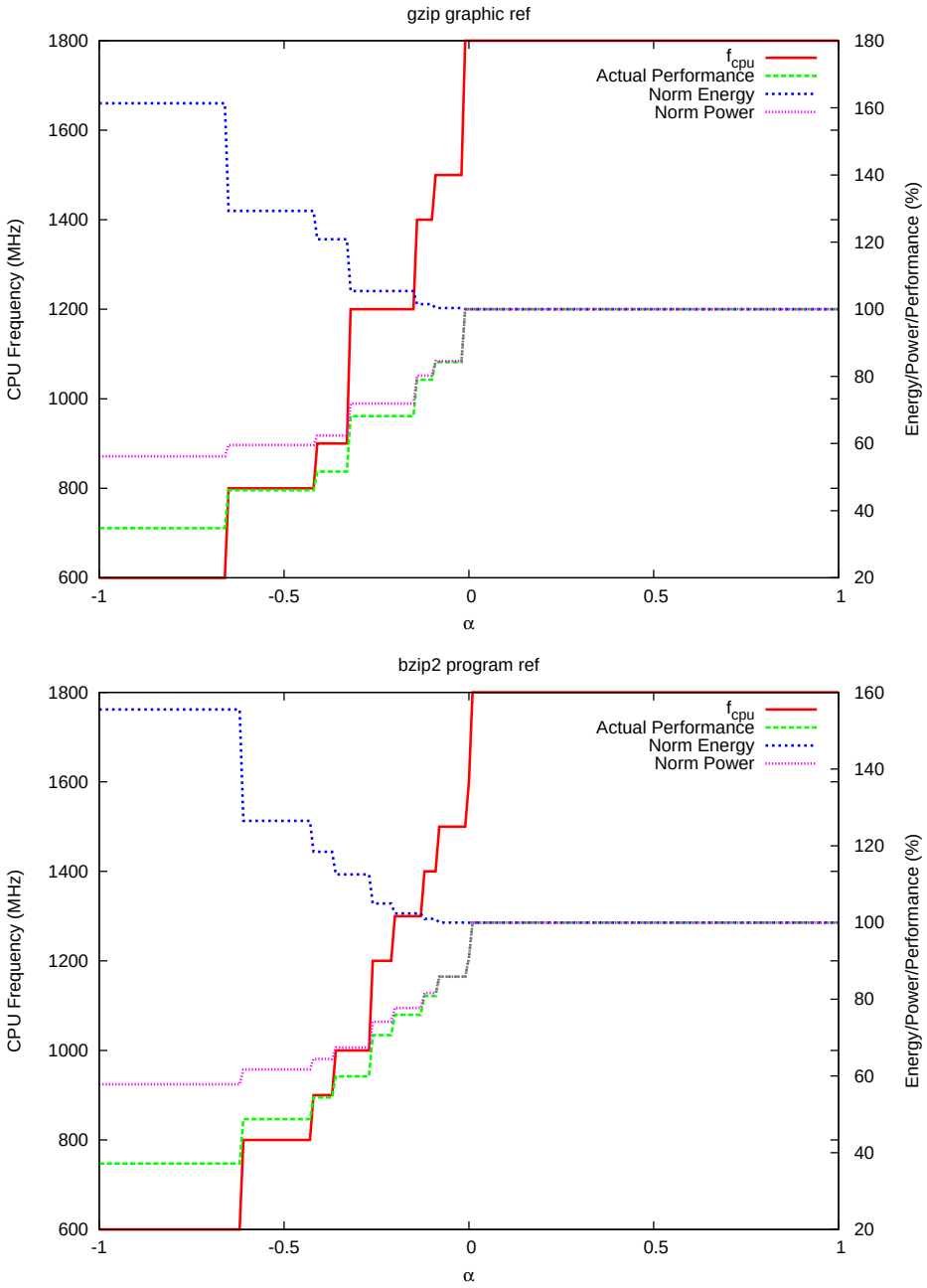


Figure 5.5: Behaviour of the *generalised energy-delay* policy for a CPU-bound (top) and lightly memory-bound (bottom) workloads on the Latitude Laptop

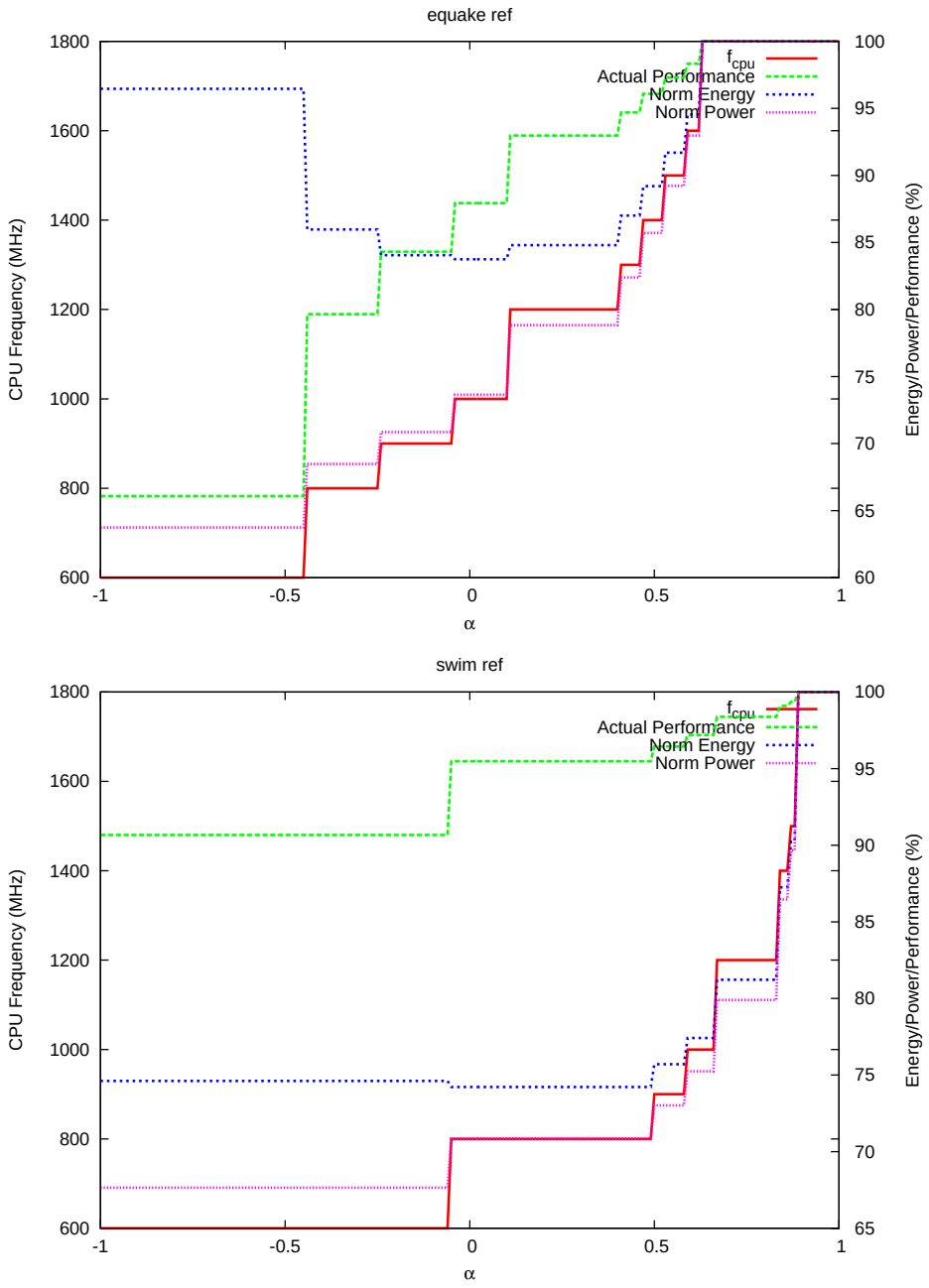


Figure 5.6: Behaviour of the *generalised energy-delay* policy for medium memory-bound (top) and heavily memory-bound (bottom) workloads on the Latitude Laptop

5.2 High-level policies

The low-level *generalised energy-delay* policy makes it possible to implement a number of high-level policies. The task of the high level policy is to choose the relative merit of performance, energy and power — to choose α. This thesis has concentrated on providing the necessary low-level mechanisms, but leaves the detailed development of automatic high-level governors to both existing and future work, presenting some proof-of-concept ideas.

Much of the previous DVFS work (including systems implementing the *bounded performance degradation* policy) aimed to minimise the system's frequency while achieving QoS targets. The α tunable can be similarly minimised (toward 0) in order to reliably save energy. For *ondemand* governor from *cpufreq* [112] in Linux does this by adjusting the frequency downward during periods of low utilisation, increasing the frequency to the maximum if the utilisation is found to be more than a given threshold. This is designed to have a minimal impact on the user's experience. Similar policies could be re-used, replacing the CPU frequency with α as the tunable. By minimising α toward 0 energy is always saved. Minimum performance guaranteed required for QoS or soft real-time reasons (such as those made by the *bounded performance degradation* policy), could be implemented alongside the generalised energy delay policy – choosing the minimum α within a given performance degradation. Since performance degradation and energy savings information will have been calculated, these higher-level policies can be implemented with low overhead.

The response of of a system to changes in α is non-linear and to some extent unpredictable. While the system will always choose the most appropriate frequency balancing energy savings against performance, the quantisation effects due to a system's discrete frequency setpoints will lead to non-linearities in the response. For this reason α will require software control to achieve QoS requirements. It is also possible to transform alpha to represent a ratio between energy savings and performance degradation – the generalised energy-delay policy will never scale the system below that ratio.

Benchmarks where energy savings are not possible are not throttled by the *generalised energy-delay* policy. The entire system might contribute to the en-

ergy savings by donating more CPU time to running processes which have been throttled. In this instance, *not* throttling a workload which can not save energy because it allows other processes to be throttled more. Since estimates for the execution time for a workload are available (thanks to Chapter 4), the scheduler can be compensate for any performance losses due to throttling.

Automatic higher-level policies can be based on entirely different criteria: A system might vary α downward for thermal throttling reasons based on a temperature sensor or other thermal emergency detection system. If the system is battery powered, the user might wish to be more conscious of their energy usage as is becomes apparent that the battery will be discharged, and α would be reduced towards 0 as the battery state of charge decreases.

CHAPTER 6

IMPLEMENTATION

> *"It is almost as if you were frantically constructing another world while the world that you live in dissolves beneath your feet, and that your survival depends on completing this construction at least one second before the old habitation collapses"* – Tennessee Williams, 1953

We implemented the ideas outlined in Chapters 3, 4 and 5 in an operating-system level power management framework. It consists of modifications to the Linux operating system, and some supporting infrastructure, and is named Koala. This chapter describes the implementation, outlines some issues which we encountered, and our solutions.

6.1 Overview

Linux 2.6.24 was chosen as a basis for the demonstrator system because it was:

- popular and well-known within the systems community;
- readily available as an open-source project;
- the latest version available at the time of implementation;
- portable and scalable: Linux is used on a wide variety of different platforms, including embedded, laptop, desktop and server systems.

The power management framework described here only considers active power management, leaving idle-mode management to the underlying OS. In the

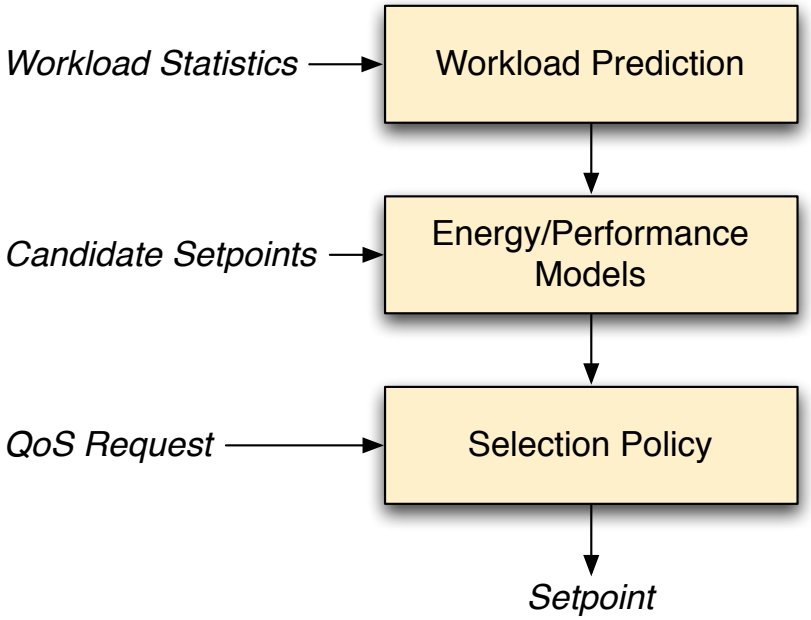


Figure 6.1: Koala high-level block diagram

case of Linux, this means that idle time is still managed by the inbuilt `cpuidle` infrastructure [111].

Figure 6.1 gives an overview of the Koala approach:

1. a prediction is made about the properties of the workload to be executed, and when those properties change;

2. the energy and time models described in Chapter 4 are used to predict the relative power and performance for the workload at a set of candidate settings;

3. the policies described in Chapter 5 are used to choose a setting for the section of workload based on the power/performance estimations, and the system's QoS requirements.

The following sections describe these three components of Koala.

6.2 Workload Prediction

We use a very simple workload prediction scheme, which works well for the benchmarks tested, and at the update rates tested (see Chapter 7). Improving this prediction scheme to handle highly variable workloads and a finer granularity is left to future work.

Like much of the prior work, this simple workload prediction scheme is based on a number of assumptions. Firstly, applications tend to execute an operation repeatedly. An MPEG player, for example, will execute a similar set of instructions for each frame of a movie. A word processor will execute similar instructions each time a key is pressed. A high-performance computing application tends to execute a tight loop many times to complete a large task. Workloads tend to be self-similar at some resolution. Like the previous work, we group these sets of similar instructions into a group known as a *phase*. For our purposes, we define a phase as a group of instructions that which should be run at a particular power management setting. Note that the granularity of the phase is arbitrary: a memory-bound phase may consist of many memory instructions interspersed with a few CPU instructions. At a fine granularity, the groups of CPU instructions could be considered CPU-bound phases, even though at a higher, macro level, the phase is memory-bound.

Threads, being an execution context, are naturally related to the workload. Changing between threads gives a high probability that the workload will be dissimilar. For this reason, Koala re-evaluates the power management setting whenever a context switch occurs. In this way, CPU-bound and a memory-bound threads can co-exist on the same processor, each being run at an appropriate frequency as chosen by the policy.

The time-slicing mechanism employed by Linux for scheduling presents an obvious opportunity for frequency scaling even if the same thread will continue to run. On each time-slice tick, and each context switch, Koala reads performance counters and any other statistics saves them in the linux `task_struct` for that thread. The values for last time-slice of the next thread, which were previously saved in its `task_struct` are then provided to the models for setting selection.

This is termed in the literature as a *last-value* predictor, and implies some

simple, if imperfect, answers to the workload prediction decisions:

Granularity The frequency scaling granularity is fixed at the time-slice size, which is unlikely to perfectly align with a thread's phase changes.

Overhead Frequency switching overheads are assumed to be low enough that switching often (potentially on each context switch) leads to an acceptable overhead. As discussed in Chapter 7, this meant that several platforms could not run Koala effectively.

Predictability Consecutive slices of the same thread are expected to be similar. Each phase change will therefore result in at least one slice running at an incorrect power management setting.

For the above reasons, academic research has developed more involved phase prediction algorithms. The simplifications made for demonstration in this thesis are not fundamental — previously used workload prediction systems could easily be incorporated into a more robust system.

For example, we considered using a system call to effect a frequency re-evaluation. Other suitable frequency scaling points could also be used by the operating system — interrupt handlers form much of the workload on some I/O bound systems. In this paradigm, Koala-aware applications make system calls which indicate the beginning or end of a phase. Koala then saves the appropriate statistics for the just-completed phase for later use via a last-value system. Since we define a phase as requiring the same power management settings, this is is a good assumption. The burden of workload prediction optimisation is then moved to the driver writer, application designer, library designer, or compiler, who must identify the phases, and determine the granularity and number of phases he wishes to indicate to the operating system. Note that this scheme is platform independent: while applications indicate the start and end of the phase, determining the properties of that phase is left to the models.

In summary, workload prediction splits a workload into small sections which can be independently managed. The smaller the sections which can be accurately predicted, the more appropriate the power management decisions will be, how-

ever the overheads and un-predictability in managing very small portions of the workload limits the granularity. For the purposes of demonstration, the operating system's in-built time-slicing and timer-tick frequency is used for workload prediction.

6.3 Modelling

Koala estimates the execution time and energy for a workload, at each of a set of potential settings. This information which is provided to the policy module.

To do this, whenever called, the system calculates the value of two linear equations, for each of the candidate settings. These candidate settings are defined in an input file in the operating system source, which is translated at build-time into the `C` code which implements the model. The characteristics of each setting (e.g. f_{cpu} and V_{cpu}), along with the information required by the hardware driver to enter that setting, are stored in arrays. An example settings file is shown in Listing 2. The index into the list of settings is called the *setting number*. The ordering of the settings does not imply anything about the setting, although it is convention that a higher setting number implies a higher-performance setting. Each row in Listing 2 defines a hardware setting with three frequencies and a voltage (although two of those frequencies remain constant). The left hand column in the listing gives the value to be written to the CPU's clock-scaling control register in order to enter that row's state.

The model calculations are the only part of the Koala system that are modified in order to port to a new platform. They must be highly optimised, since these calculations are the source of most of the computation overhead.

To generate the models we perform a linear regression (using the R statistics package [125]) and generate the model parameters as in Chapter 4. The parameters are stored in a file along with the Koala source code. At system-build time, a script translates this into C source code which performs the model calculations. The model parameters are embedded into this source code and look-up-tables are pre-calculated for speed (for example, a P_{static} for each setting).

This implementation makes a trade-off between portability and speed. It presently requires architecture-specific implementations of the basic models and

hardware drivers, but further work should eliminate the former.

The model calculations are carried out in-kernel, and are implemented as a series of fixed-point calculations (since Linux does not support floating point operations in the kernel, and several of the systems we examined do not have floating point units).

In addition to the predicted workload statistics, the models can use the predicted system conditions. At present, this includes the temperature and fan speed on systems that support it. Future systems might use the expected level of bus contention, IO power, and other system state. These statistics are presently gathered by the architecture-specific implementation.

An omission in this implementation is an idle power estimation — it is presently assumed to be constant. This is a simplification to avoid expanding the scope of this thesis. The real idle power should be the power used during any extra available CPU time, and depends on the system activity and the types of workloads which are running. For systems which are 100% utilised, the idle power is 0W, since any extra CPU time will be used to do useful work. For systems which will idle once the workload finishes, the idle power is non-zero, and depends on the idle mode being entered (see Section 3.2.3). Given an estimated idle power, the effects of idle energy can be removed by scaling based on the dynamic energy as in Equation 4.14.

As discussed in Sections 3.2.4 and 4.7, the switching overhead is non-trivial and platform dependent. Koala attributes the switching overhead to the time-slice which causes the switch, and so the overhead is added to the execution time for the upcoming time-slice. This is also a simplification, since a frequency switch may be amortized across many timeslices. It is a limitation of the present implementation which should be addressed by future work.

6.4 Policy

The policy module chooses the best setting for the predicted workload, given the time and energy estimates. It implements a function which takes an array of expected execution times and energy and returns the setting number for the selected setting. A number of policies have been implemented and the active policy can be

Listing 2 `latitude.conf` — the source file used to define settings for the Dell Latitude D600 laptop

```
#setting   fcpu     fmem    vcpu    fbus
0x0816     800.0    100.0   1.052   100.0
0x0918     900.0    100.0   1.084   100.0
0x0a1a     1000.0   100.0   1.116   100.0
0x0b1c     1100.0   100.0   1.148   100.0
0x0c1d     1200.0   100.0   1.164   100.0
0x0d1f     1300.0   100.0   1.196   100.0
0x0e21     1400.0   100.0   1.228   100.0
0x0f23     1500.0   100.0   1.260   100.0
0x1025     1600.0   100.0   1.292   100.0
0x1127     1700.0   100.0   1.324   100.0
0x1228     1800.0   100.0   1.340   100.0
```

changed by the user at run-time. The presently implemented policies are:

manual the user can directly choose the setting

max chooses the maximum setting number

min chooses the minimum setting number

mine chooses the minimum energy setting

mined chooses the setting with the minimum energy-delay product

perf chooses the setting with a performance closest to, but not less than, a performance bound (corresponding with Section 5.1.2)

alpha chooses the setting with the minimum η, where η is calculated according to the generalised energy-delay policy described in Section 5.1.3.

The active policy module can be selected via a `/proc` interface, and the policies with tunables (**manual**, **perf** and **alpha**) are controlled by the same means.

The calculations in the policy module must also occur in fixed point. The only relevant policy in that context is **alpha**, which must calculate η. In order

to minimise run-time costs and avoid floating-point arithmetic, a more suitable representation of the policy function which avoids exponents is

$$\log_2 \eta = (1 - \alpha) \log_2 E + 2\alpha \log_2 T. \tag{6.1}$$

This was implemented in fast fixed-point arithmetic using the `clz` instruction and a look-up table. Since $\log$ is a monotonic function, minimising $\log_2 \eta$ also minimises η.

We implemented higher-level policies at user-level, interfacing to the kernel-level policies via the /proc interfaces. These monitor the system at a much slower rate, updating α. Our BSOC policy is implemented as a program which reads the *battery state of charge* (BSOC) from the smart battery interface on the Dell Latitude D600 laptop, and modifies α. Above 70% BSOC, the program sets alpha to 1.0, enabling the full system performance. Below 70% the value is reduced linearly to 0.0 (minimum energy) at 30% BSOC. While this policy may not be entirely practical, it demonstrates how higher level policies can be written.

6.5 Other Details

We constructed drivers for each of the platforms. These manage the access many different types of performance counters on numerous platforms, different types of frequency scaling interface, different power management ICs, and other miscellaneous hardware (including the Dell i8k BIOS interfaces). In some cases, Linux drivers for these devices were available for reference, but in most cases the existing drivers were inadequate.

6.6 Infrastructure

Several pieces of user-level infrastructure required of a power management framework have been implemented for Koala. In particular, this includes a number of user-level interfaces.

The first of these is the statistics sub-module. It collects information about Koala's operation as well as the results from Koala's models. This information is

made available both globally and per-process, and consists of:

switchops the number of switching opportunities;

switches the number of times the setting actually changed;

setting_acc the accumulated raw value of the setting (accumulated on each switching opportunity);

energy the energy that koala estimates was used by the system;

max_speed_energy the energy koala estimates the system would have used if it were running at the maximum setting number;

time the execution time as recorded by Koala;

max_speed_time the execution time that Koala estimates for the same workload running at the maximum setting number.

In addition to the above, accumulated cycles and performance counter settings are available.

From the above, it is possible to calculate some interesting information (both globally and per-process)...

- the percentage of switch opportunities that resulted in a setting change;
- the system's performance compared to the maximum setting;
- the system's energy compared with the maximum setting;
- the average setting.

To display this information, we developed two user-land tools. These are both derived from commonly-used system administration tools. `top` is shown in Figure 6.2, and shows the energy saving and relative performance of each process in the system. A version of `gnome-system-monitor` performs the same function. The statistics are a summary generated as a part of Koala's operation, so the overhead is minimal.

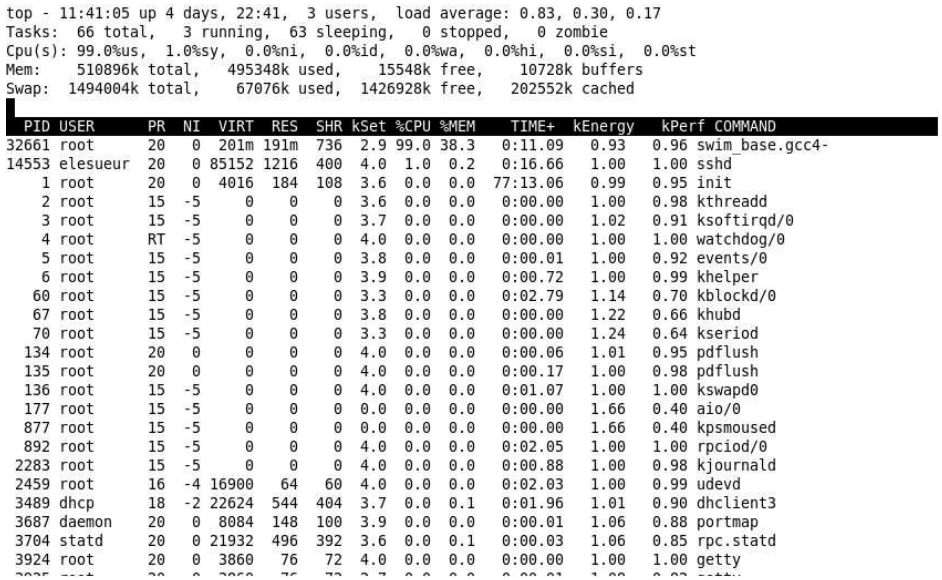


Figure 6.2: A modified version of `top` which displays statistics generated by Koala

Another user-level interface is the tracing module. This consists of an in-kernel circular buffer which stores information recorded during each switching opportunity including the statistics used by the models. The information can be used to examine Koala's behaviour in detail. The decisions be re-created, and the power savings made at each stage during a workload's execution can be estimated. The information is exported to user-land via a character device. Modifications were made to `gnome-system-monitor` to display this information in real-time (shown in Figure 6.3).

These interfaces are very useful. Technically savvy users can manually assess the effect of frequency scaling and decide whether a particular QoS impact is worth Koala's estimated energy saving. System designers can make similar assessments. The information can be used by software designers to evaluate how much energy their programs use, and whether Koala-style power management helps.

We used both of these interfaces to collect the data presented in Chapter 7.

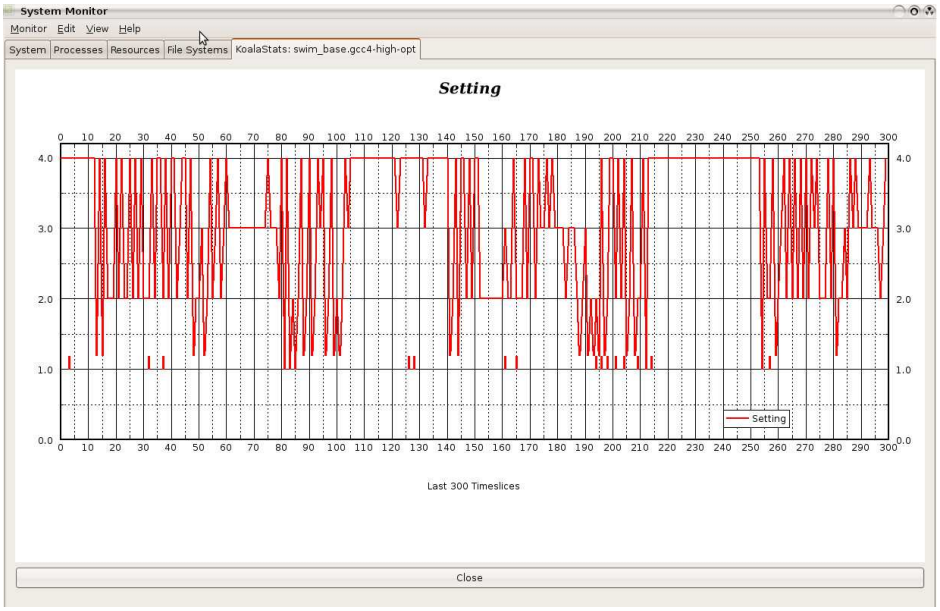


Figure 6.3: A modified version of `gnome system monitor` showing statistics recorded by Koala's trace module

6.7 Discussion

The implementation presented here demonstrates model-based power management. For this demonstration we have made a number design decisions that are not prescriptive. The implementation provides an experimental platform which can be ruggedised, rather than complete solution.

We roughly optimised the system, using fixed-point arithmetic in the kernel, and other optimisations which reduce the system's precision (such as the fast logarithm function used to implement the `generalised energy-delay` policy). While these optimisations certainly help with the system's speed, the performance is not yet optimal, and the optimisations require specific attention for each platform (for example, the fixed-point precision must be checked for each platform, since the magnitude of the power used is different).

Likewise, the system uses a brute-force approach to finding the optimal setting, so the best frequency will always be found, but requires energy and performance estimations for every setting. In systems like the I-Box where there are

thousands of potential settings, this was not practical. In that case, a search algorithm should be developed so that only a small subset of the potential settings needs to be examined.

We presently require the models to be statically compiled at build-time. The model constants are stored as an automatically generated python script which is used by the Linux build system to generate header files. As an alternative, the models could be built as kernel modules without loss in performance. As a long-term goal, the models should be adaptive, using measured feedback from the running system. Ideally, accurate models could be provided by the hardware component manufacturers and composed into a system model. System specific sources of model error could be identified via measured feedback. The models could be stored or even implemented by the hardware itself, allowing for a very small OS overhead. Since the manufacturers are likely to have the most detailed information available on the system's design, they are the ideal party to develop these models.

Another alternative would be a similar approach to that used in Intel's Core i7 processors [151]: the power management subsystem could be implemented on a power management processor (called the PCU in the Core i7) with access to the performance counters, current and temperature sensors, and control over the processor's power management settings. Operating-system interfaces would be used to facilitate workload prediction and the setting of policy parameters such as α. The approach would clearly lower the performance overhead of running Koala, allowing for more fine-grained frequency scaling. The sensors built into the Core i7 would make ideal parameters in Koala's models.

How Koala would be implemented in main-stream operating systems running on varying hardware remains an open problem. The issues are varied, but considered manageable, particularly if hardware vendors were to become involved.

One problem is the effort required in determining the models for a given system. This effort would be significantly reduced with hardware-vendor support in providing appropriate models and inputs. Another issue is the need for dedicated performance counters, which may be required for other purposes. Again, this is an issue which hardware manufacturers can easily address.

CHAPTER 7

EVALUATION

> *"You know more than you think you know, just as you know less than you want to know"* – Oscar Wilde, 1890

This chapter details our evaluation of the ideas presented in Chapters 3, 4 and 5, as well as the implementation detailed in Chapter 6, comparing the theory presented in the prior chapters with measured reality.

7.1 Methodology

7.1.1 Measurement Methodology

We constructed the infrastructure required to conduct experiments on Linux 2.6.24, including scripts for running benchmarks, drivers for collecting data from the platforms' inbuilt sensors and counters, and scripts for post-processing the data. Care was taken to minimise the any effects that would violate the assumptions outlined in Section 4.2. In particular, we ran benchmarks from a RAM filesystem, and discarded any output to minimise system I/O.

Energy

We used two methods to measure the energy drawn by the platforms. For low voltage (<20 V) measurements I designed a device, dubbed *Echidna*. For mains AC measurements, we used an Extech True RMS Power Analyzer 380801 [39]. In each case, we were able to measure the average current, average voltage, average power, and total energy.

Echidna (shown in Figure 7.1) is a simple circuit designed to measure the total energy drawn during the execution of a benchmark. It is accurate to within 5 mW

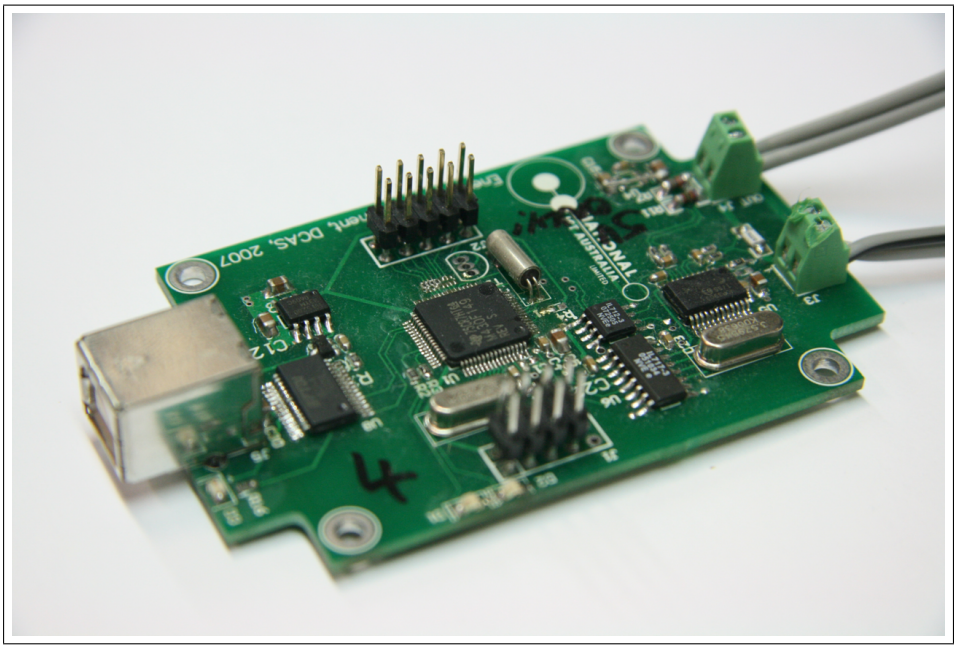


Figure 7.1: The custom-designed Echidna energy measurement device

. It is programmable and can be triggered via a hardware signal. Echidna also measures temperature. While triggered, the device integrates the power to obtain the energy. The platforms generate a trigger signal using the lowest-overhead means possible, giving good synchronisation. Further details of Echidna, including schematics and the board layout, can be found in Appendix B. The Extech 380801 True RMS Power Analyzer [39] is a device for analysing mains AC power loads. It has various displays, in addition to a serial interface. It samples at a much slower rate than the Echidna — 2.5 Hz — with an accuracy of 0.9%. Since there is no external triggering, some small overhead is required in this situation since the results must be recorded on the machine under test.

The power varies substantially with temperature, so all measurements were taken in a constant-temperature environment. In some experiments, we used a refrigerator to vary the ambient temperature. We used the system's internal sensors to measure the temperature.

Any platform specific methodology is discussed with the detailed platform descriptions in Appendix A.

Performance

Prior to each benchmark run, the system was put in a consistent, quiet, state. This involved minimising the system's background activity. In particular, the networking, disk and swapping mechanisms were disabled. In addition, we disable any background activity (for example, `init` executes some routines once per second). In the case of `init` this required source code modifications. For some benchmarks we reduced the system's timer tick frequency in order to minimise the overhead of running the scheduler, improving the consistency of the benchmarks. The `powertop` [80] utility was used to reduce the wakeups per second during an idle mode to less than 10Hz. Background activity was detected during a benchmark run via the interrupt count.

Idle Energy

Of the two models presented in Section 4.5, this thesis uses the dynamic energy to evaluate the effect of an idle power, since it does not depend on multiple benchmark executions so it can be calculated dynamically at run-time. Once the standard Linux mechanism for managing idle modes (`cpuidle`) was enabled, we measured the typical idle power for each platform after at least 10 s of being in the idle state. For the purposes of this evaluation, the idle power is assumed to be constant. The task of predicting the actual idle mode power based on the expected time spent in each idle mode is left to future work.

7.1.2 Benchmarks

While a variety of benchmark suites were used for evaluation, four suites are presented in this thesis. Three are those developed by the *Standard Performance Evaluation Corporation* (SPEC) and provide *"a comparative measure of compute-intensive performance across the widest practical range of hardware using workloads developed from real user applications"*. They are an industry standard, and commonly used for systems performance evaluation. In the context of this thesis, the SPEC CPU benchmarks provide real-world application workloads.

Three versions of SPEC were used: CINT95, CPU2000 [63] and

CPU2006 [64]. Wherever possible the benchmarks were used un-modified, but in some circumstances they needed to be updated to suit the newer compilers we used. The SPEC benchmarks were used to evaluate the energy and performance characteristics of desktop and PC-style systems, as well as some more powerful embedded systems.

The fourth benchmark suite was MiBench, which is a *free, commercially representative embedded benchmark suite* [61]. It is similar to the EEMBC suite. It consists of 35 real-world application benchmarks taken from typical embedded systems circa 2001.

Lastly, the memory boundedness of some systems was tested using specially written synthetic benchmarks.

All of the benchmarks were compiled using GCC [53]. The version of GCC and whether a cross-compilation was used varies with the platform.

Benchmarking difficulties

Consistent results are critical when comparing energy management schemes, as well as when determining the value of all possible performance events using a small number of counters. We observed some large variation in the execution time and energy for several of the experiments. The cause of that variation was severalfold. Firstly, background activity affects the performance and energy requirement of the benchmark under test and must be minimised. Secondly, there were problems with some of the benchmark suites themselves: one benchmark in MiBench (`pgp`) used the `gettimeofday` function to randomise its behaviour.

More interesting than these, the specific physical pages allocated to a process have a significant effect on the run time of the workloads on some systems, particularly the Latitude laptop. Rebooting the system between benchmarking runs ensures the system is in a consistent state, and the configuration of the disk cache could be made predictable by loading a large file and flushing the cache.

Despite these efforts, benchmarks like `mcf` and `twolf` do not run with consistent results on the Latitude. Koala handles the instantaneous workload execution, even if subsequent executions can not be compared. This can be seen in Figure 7.18, where the successive iterations of `mcf` have an inconsistent run-time.

7.1.3 Statistical Methods

A detailed summary of the implementation of the statistical techniques as used here is beyond the scope of this thesis, and are mostly implemented in the `R` statistics package [125]. All models were characterised using an un-weighted least-squares regression (`lm` in `R`).

We performed parameter selections using the `regsubsets` package. This compares all N-parameter models, for several values of N. It then selects the one with the highest coefficient of determination — R^2. The *bayesian inference criterion* (BIC) can be used to compare the merit of using more or less parameters in the model, helping to avoid co-linearities. The results of these analyses can be plotted in Figures like 7.6, 7.7, 7.8 and 7.9. In these plots, the plot area is split into a grid. Each row in the grid represents the best model which could be found for a given number of parameters. The bottom row has one parameter, the next row up has two, and so-on. Each column in the plot represents a parameter, and the X-axis labels these. If a block is coloured, then that column's parameter is used in that row's model. The Y-axis labels show the coefficient of determination (R^2) for each model/row. Each parameter in a row is the same colour, and a darker colour implies a better fit for that model.

In some cases it took several months to run a sufficient number of benchmarks to build a model. When multiple iterations were feasible, we calculated averages prior to any model fitting. Following a parameter selection, further experiments were conducted using the selected counters in order to generate the characterisation data.

According to sound statistical practice, validation and characterisation were performed using two different datasets. SPEC CPU2000 was generally used for characterisation and SPEC CPU2006 for validation. When evaluating the system or policy (as opposed to the model's accuracy), a selection from both the characterisation and validation set are used.

Platform	Type[a]	Processor	Architecture	Settings[b]	f^c	V^d	Counters[e]	Events[f]	Other notes
PLEB 2	Embedded	PXA255	ARMv5TE	22	3	1	3	14	
Gumstix	Embedded	PXA255	ARMv5TE	22	3	0	3	14	16-bit bus
I-Box	Embedded	PXA270	ARMv5TE	169	3	1	5	14	
Phycore	Embedded	iMX31	ARMv6	4	1	1	7	35	
Latitude	Laptop	Pentium-M 745	IA-32	12	1	1	3	>164	
Thinkpad	Laptop	Pentium-M 750	IA-32	8	1	1	3	>164	
EEEPC 901	Netbook	Atom N270	IA-32	7	1	1	3	>113	
Opteron	Server	Opteron 246	IA-32	5	2	1	5	>177	
Menlow	Embedded	Silverthorne	IA-32	4	1	1	3	>113	
Xeon	Server	Xeon	IA-32	3	1	1	3	164	

Table 7.1: A summary of all the platforms examined

[a]Embedded, laptop or server
[b]The number of settings which we tested, rather than those which are possible
[c]The number of dynamically variable frequencies, not including fixed frequencies
[d]The number of dynamically variable voltages, not including fixed voltages
[e]The total number of event counters, including those which measure a fixed event
[f]The events which we measured, rather than those which can be measured

7.2 Platforms

This thesis incorporates data gathered by examining ten platforms which represent advanced embedded systems, netbooks, notebooks, and servers. The details of these platforms is the subject of Appendix A. The benchmarks were run on each platform at all of the possible power management settings. It was through understanding these results that the problems described in Chapter 3 were identified. This thesis examines three of the more interesting platforms in detail, representing an embedded system, a laptop and a server. We leave the implementation of Koala on other platforms to future work. Unfortunately, because of the high frequency-switching overheads, the embedded system platform is omitted from the on-line evaluation.

The embedded system is a custom-designed PXA255 single board computer called PLEB 2. It is based on a 400 MHz PXA255, overclocked to 471 MHz. It has 64 MB of SDRAM with a bus clock rate between 100 MHz and 133 MHz. The CPU, bus and memory frequencies are all variable, along with the CPU core voltage which can be varied in 0.1 V steps. The bus and memory frequencies are generated using dividers from the CPU clock, so all three must be modified together. We identified 22 combinations of CPU, bus and memory frequency, along with CPU core voltage, that we call a *setting*.

The XScale core has two performance counters and a cycle counter. Each of the performance counters can measure one of 15 events which we used to create the models. The frequency switch latency is $500\mu s$, during which the CPU is unavailable. In addition, the voltage must be manually ramped using a slow I2C connection to the power supply IC.

We measured the power on the battery supply using the Echidna (see Section 7.1.1). This reflects the importance of the battery in a mobile embedded system.

The laptop is a Dell Latitude D600 [31] based on an 1.800 GHz Pentium-M processor paired with the Intel 855PM chipset. It has 1GB of DDR266 memory with a 133 MHz clock rate, and a frontside bus frequency of 100 MHz but quad

pumped to 400 MHz. The core frequency clock is varied from 0.8 to 1.8 GHz in 100 MHz steps and the core voltage is varied from 0.98 V to 1.34 V. We switch frequencies by accessing the respective registers directly rather than via ACPI, as ACPI did not export all possible frequencies voltages. The LCD backlight was switched off to reduce the system's static power, and improve savings gained from the CPU for the experiments.

The Pentium M has a cycle counter and two user-configurable counters which can each measure one of several hundred different events, of which we used 164 potentially-relevant events to create the models. During a frequency switch, the CPU is unavailable for $10\mu s$ while the frequency-synthesis circuitry (PLL) re-locks. The voltage is automatically ramped up and down by the hardware prior to or following the frequency switch, respectively. This means that for a short period of time following the switch request, the processor may operate at a frequency different than the requested one.

Power consumption was measured in the battery supply line using Echidna (see Section 7.1.1). Measuring at the battery reflects the importance of minimising the battery energy use (rather than wall-socket energy) while disconnected from an external supply.

In addition to a steady-state measurement of the Latitude laptop, the idle power for each sleep mode used by the operating system was determined via the method outlined in Section 4.5. The system has three sleep states, and uses 18.5 W in C2, 13.1 W in C3 and 11.1 W in C4.

The server is based on an AMD Opteron 246 processor clocked at 2 GHz. Using a custom driver, we were able to put the CPU in five different settings ranging from 0.8 GHz at 0.9V to 2 GHz at 1.5 V.

The system has 512 MB of DDR400 SDRAM, and while the memory frequency was fixed at 200 MHz, we noted that the memory controller changed the memory bus frequency down to 160 MHz at the lowest CPU frequency of 800 MHz.

The overhead for a switch on this platform varies depending on the current and target frequencies, ranging from as low as $15\mu s$ when dropping the frequency from 2 GHz to any value, to $140\mu s$ when increasing from the minimum to the

maximum frequency. This is due to delays while ramping the core voltage to the required level, and then the re-locking delay incurred by the PLL of the clock generator.

The processor has a cycle counter and four user-configurable counters which, like the Latitude, can each measure one of several hundred events, of which we examined 177.

Power measurement was performed using with the Extech power meter. This was inserted between the wall socket and the machine's power plug and thus measured the system's total AC power consumption (a reduction of the total AC power consumption being the main goal for server power management).

7.2.1 Comparison

We observed a substantial variation between the power management capabilities of the platforms. Embedded ARM-based platforms like PLEB 2, Gumstix, I-Box and Phycore are capable of a vast array of settings, including variable memory and bus frequencies. IA-32 platforms tended to have fewer possible settings, usually only the CPU frequency being variable (although the Opteron was a notable exception). The ARM-based chips also have very low-power processor idle modes. The Phycore was unique amongst the systems examined, with a variable-frequency interface to its level 2 cache (the PXA-based platforms do not have a level-2 cache).

The available memory bandwidth was also varied. For some workloads, platforms such as the Latitude, show near-constant-time behaviour across different settings for some benchmarks, indicating a highly memory-bound workload. On other platforms the memory-performance is rarely a bottleneck. Figures 7.2 and 7.3 compare the Latitude and the ARM-based Phycore platform. On the Latitude, the memory-bound `swim` increases its execution time by only $\sim$10% despite a clock speed divided by three. The commonly assumed model would predict a threefold increase in execution time, as is seen for the CPU-bound benchmark (`twolf`). On the Phycore, even the most memory-bound benchmark that was run, `mcf`, increases it execution time by nearly three times for a quarter of the clock speed.

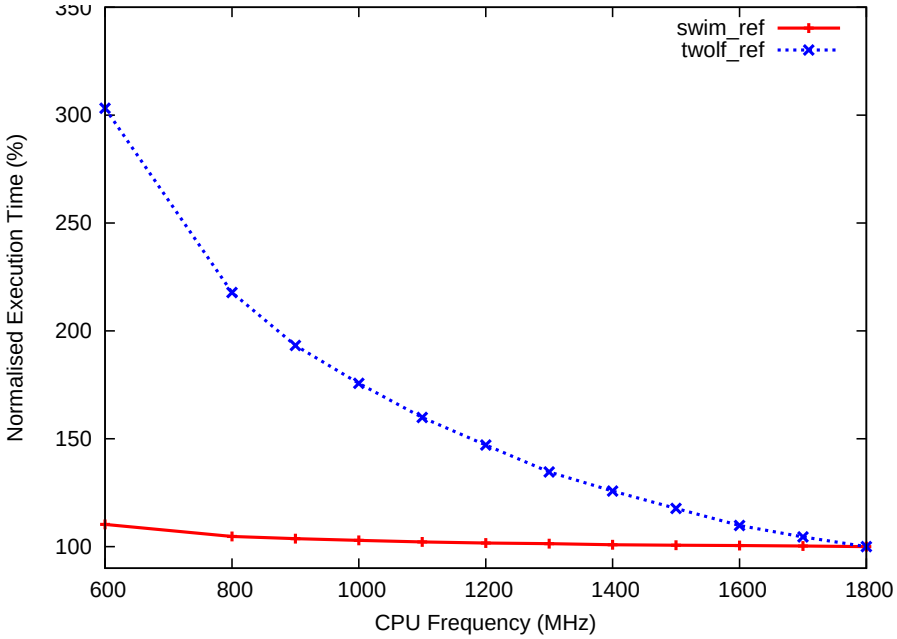


Figure 7.2: Normalised execution time for a memory-bound (`swim`) and a CPU-bound (`twolf`) on the Latitude.

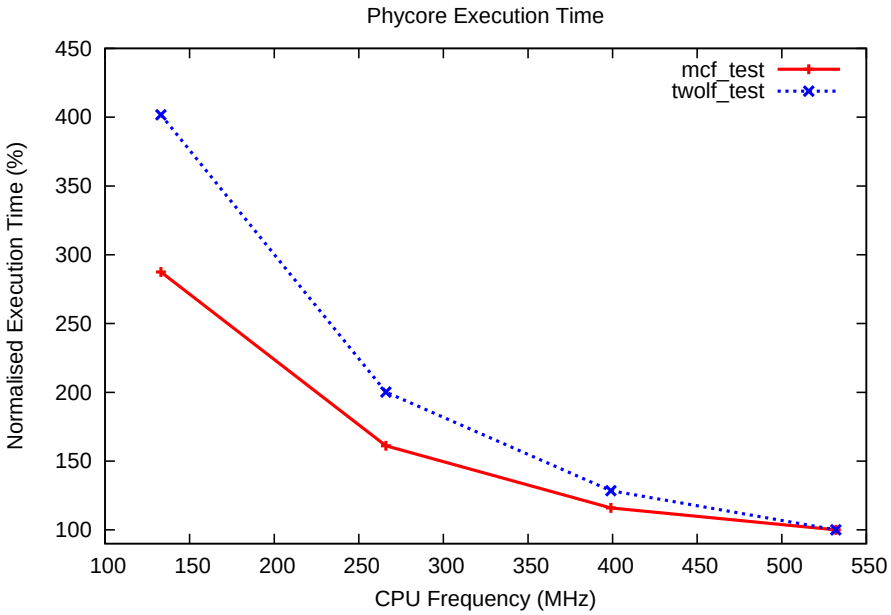


Figure 7.3: Normalised execution time for a memory-bound (`mcf`) and a CPU-bound (`twolf`) on the Phycore.

The frequency switch latency also varies significantly amongst platforms. The PXA-based platforms have a high latency of $\sim 500 \mu s$. The Pentium-M based Latitude and Atom-based EEEPC have the lowest overheads of $\sim 10 \mu s$.

The platform design, as well as the processor design, has a noticeable effect on the frequency-scaling behaviour of the system. This is most obvious when comparing the Gumstix and PLEB 2 platforms, which both use an Intel PXA255 system-on-chip processor. The memory-bus on the Gumstix is 16-bits wide, rather than the 32-bit bus on PLEB 2. Furthermore, PLEB 2 can scale its CPU voltage, whereas the Gumstix has a fixed voltage. These differences result in markedly dissimilar optimal frequency scaling decisions on these platforms — compare the minimum-energy setting in Figures 7.4 and 7.5.

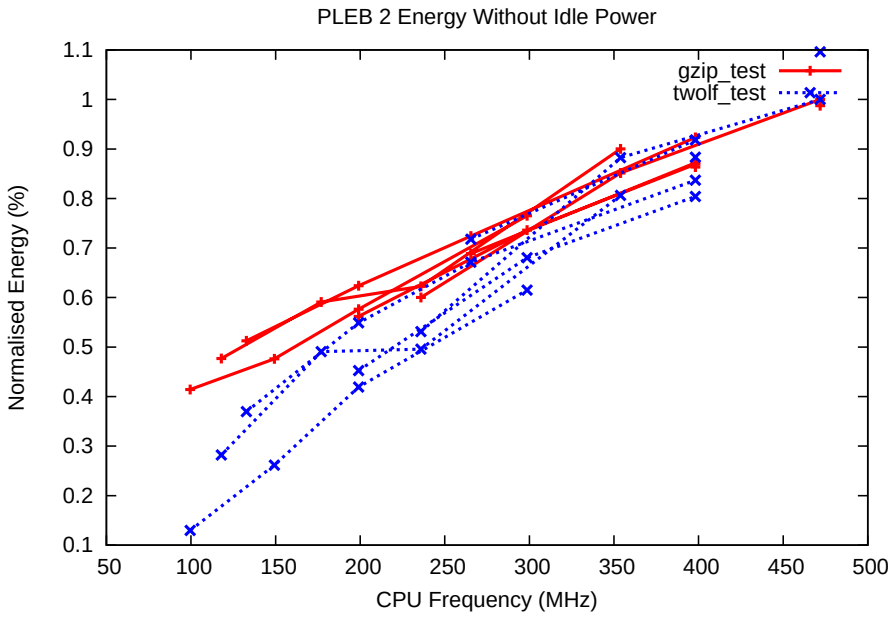


Figure 7.4: Dynamic energy for a memory-bound (`gzip`) and a CPU-bound (`twolf`) on PLEB 2. Lines connect points of equal memory frequency.

Platforms with lower power idle modes are usually energy-optimal at higher-performance settings. Several platforms' energy-optimal setting changed depending on the workload. For other platforms, the energy-optimal setting is constant, independent of the workload.

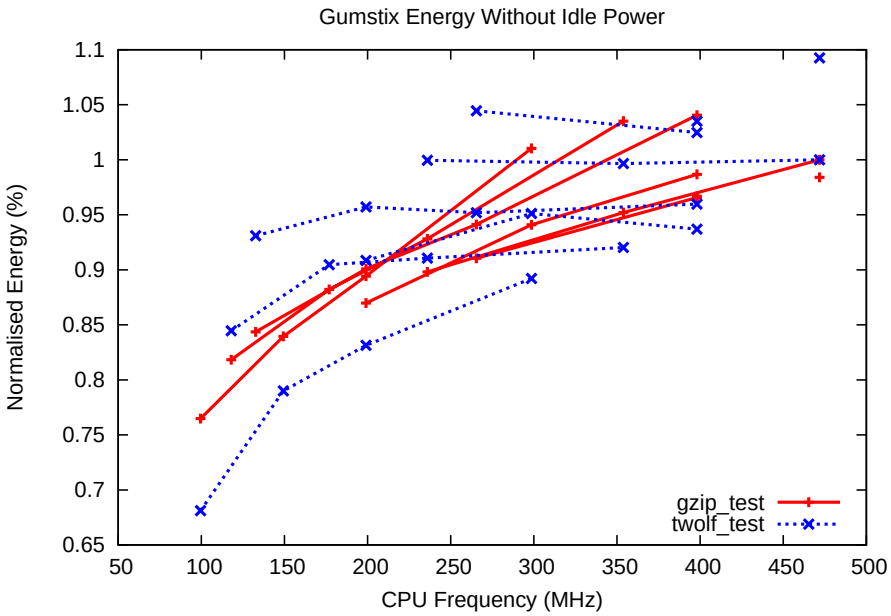


Figure 7.5: Dynamic energy for a memory-bound (`gzip`) and a CPU-bound (`twolf`) on Gumstix. Lines connect points of equal memory frequency.

No conceivable heuristic will produce optimal results for all of these platforms.

In the interests of tractability, we focussed on three platforms: PLEB 2 (the *Embedded System*), the Opteron (the *Server*) and the Latitude (the *Laptop*). Some of the others were eliminated for the following reasons: the PXA-based systems have high frequency switching overheads, eliminating the possibility of per-timeslice frequency changes; the iMX31 processor in the Phycore has many silicon bugs and could not be used reliably; the Intel Xeon-based systems have very few potential settings.

7.3 Characterisation

Here we demonstrate the model-building techniques presented in Chapter 4. The models will be used later for on-line predictions (Section 7.5).

First, data is gathered about the platform according to Section 4.10.1. For

PLEB 2 we use 37 benchmarks from the MiBench suite and SPEC CINT95 for characterisation, and selected benchmarks from SPEC CPU2000 were used for validation. The Latitude and Opteron used SPEC CPU2000 benchmarks for characterisation, and SPEC CPU2006 benchmarks for validation.

The models for each platform are formulated as in Equation 4.29. On the Latitude, where the memory frequency remains constant, the model can be simplified to Equation 4.30. The power models for the platforms are taken from Equation 4.12, although the platforms without variable bus or memory frequencies lack these parameters.

The parameters are selected using R's `regsubsets` command. In theory it would be possible select parameters based on a combined power/performance model, however this thesis found that we get good results if we use regression on the performance model alone to select the events.

For a complete example of a model which was generated, see Listing 1, which is the human-readable source file used to store the model for an Opteron-based server.

7.3.1 PLEB 2

The parameter selection for PLEB 2 is shown in Figures 7.6 and 7.7. The event associated with each number are given in Table 7.2. The figures show R^2 values (indicated by colour and y-axis labels) starting with a single-parameter model in the lowest row, adding one parameter in each higher row. The parameters are listed as the x-axis labels, "Bus PMC$_x$" indicating PMC_x used for predicting C_{bus}, etc. "Intercept" represents the x-axis intercept of the linear model.

These figures allow us to understand how power is used in the platform. In the performance model the best two-parameters are DTLB misses (PMC4) for both bus and memory. Similarly, the best four-parameter model uses TLB misses and data cache misses (PMC11). Each of these is a associated with the time spent waiting on memory. The best six-parameter model is less clear-cut, either using instruction cache misses (PMC0), data-dependency stalls (PMC2), ITLB misses (PMC3) or data-cache-buffer-full stalls (PMC9). In most cases, the same counters are selected for predicting C_{mem} and C_{bus}. This means that with n counters we

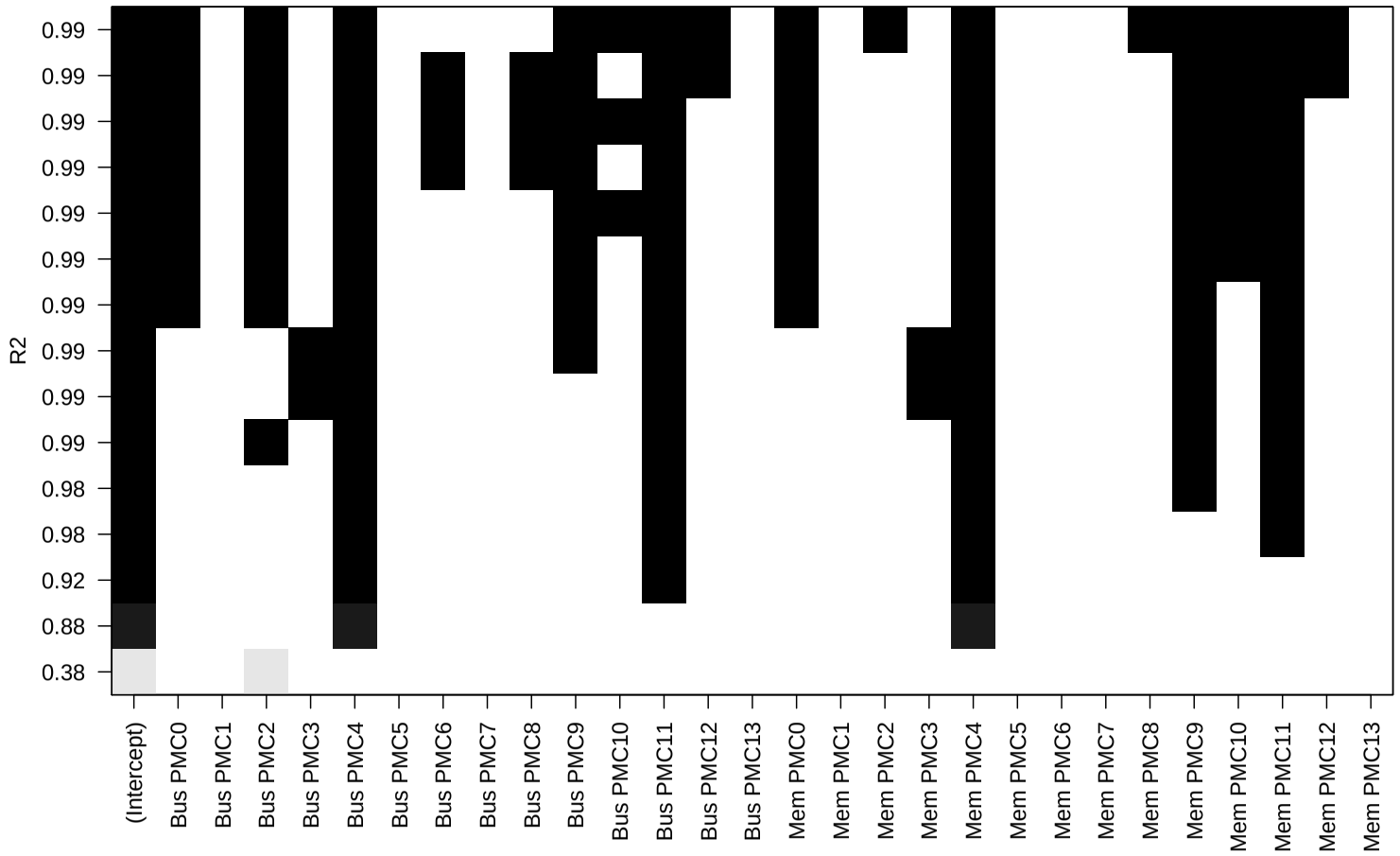


Figure 7.6: Parameter selection for the PLEB 2 time model. The X-axis labels correspond with the terms in Equation 4.29 for PLEB 2. See Section 7.1.3 for an explanation of this plot.

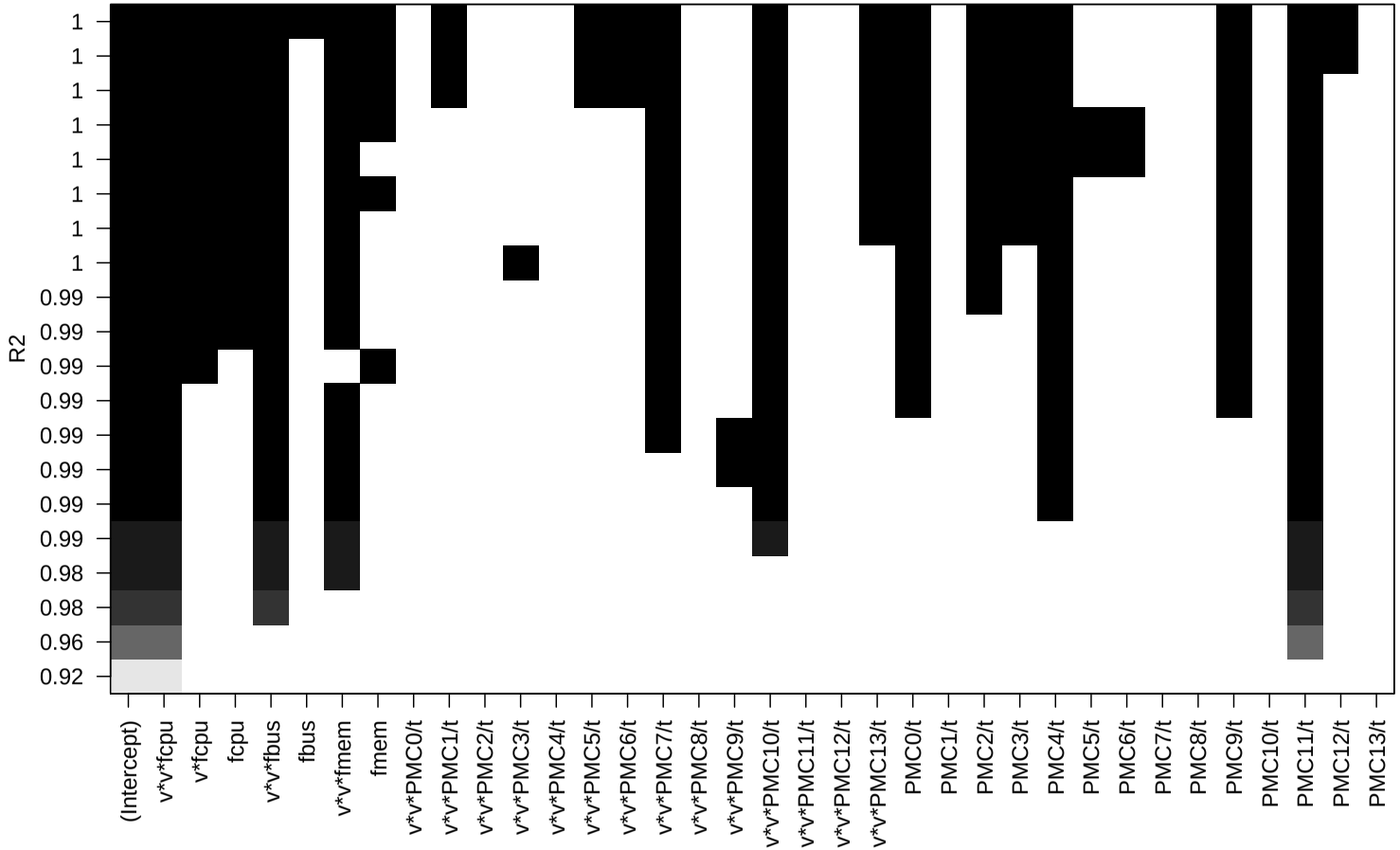


Figure 7.7: Parameter selection for the PLEB 2 energy model. The X-axis labels correspond with the terms in Equation 4.12 for PLEB 2. See Section 7.1.3 for an explanation of this plot.

Event#	Description
0	Instruction cache miss.
1	Cycles during which the instruction cache can not deliver.
2	Cycles during data dependency stalls.
3	Instruction TLB misses.
4	Data TLB misses.
5	Branch instructions executed.
6	Branches mispredicted.
7	Instructions executed.
8	Cycles during stalls due to full data cache buffers.
9	Number of stalls due to full data cache buffers.
10	Data cache accesses.
11	Data cache misses.
12	Data cache half-line write-backs.
13	Occasions on which software changed the program counter.

Table 7.2: Events available on PLEB 2's PXA255 processor [73]

obtain a good $2n$-parameter model.,

Examining the parameters selected in the power model, it is informative to observe which parameters are dependent on the system's core voltage (V^2), and which are independent. The intercept (P_{static}), and then fV^2 are obvious selections, and agree with the commonly assumed model. Data cache misses (PMC_{11}), without the V^2 dependence, are selected next. This, and the data TLB misses event PMC_4, are used to estimate the power used in the memory chips during transactions. The selection of both bus and memory frequency, *with* a V^2 dependence, are also easily explained — the bus is on-die, and therefore bus cycles consume energy according to the core voltage. The latter is because the PXA255 has an on-chip memory-controller, for which $f_{mem}V^2$ estimates the power. The number of instructions executed (PMC_7) is a good estimator for the switching activity in a processor.

Each parameter selected by this technique has a clear explanation. Parameters toward the top of the graph only show a small increase in both R^2 and BIC, and are therefore not likely to improve the predictive capability of the models.

Varying the number of counters used for power estimation has a clear effect on the predictive ability. The fitted model was used to predict the power at the sam-

pled setpoint (i.e. with a perfect performance model) for the validation dataset. The results are shown in Table 7.3, which clearly shows that the model can accurately predict the average power, and that increasing the number of counters used reduces the error in those predictions.

Counters	Param.	R^2	Max Err (%)	Avg Err (%)
1	4	0.9836	7.46	2.14
2	5	0.9871	6.94	2.31
3	6	0.9904	4.85	1.26
4	7	0.9922	3.78	1.16
5	8	0.993	3.68	0.92
6	9	0.9938	2.94	0.89
6	11	0.9947	2.75	0.77

Table 7.3: Regression and validation data for various models

For the purposes of illustration, we chose four performance events based on the parameters selected for each of the execution time and power[1], and the models are then characterised. Predicting the power for each benchmark in the validation set at the maximum frequency after running at the other settings gives an expected error. The maximum error when estimating the energy using both models combined is 4.9% and the average is 1.5%. If the measured time is used in the model (thereby isolating the error of the power model), the maximum error is 3.7% and the average error is 0.72%.

7.3.2 Opteron

On the server platform, the characterisation procedure for the performance model selected the following four events, all of which are intuitively related to memory-boundedness:

- quadword write transfers (`0x016D`);
- L2 cache misses (datacache fill) (`0x027E`);
- dispatch stalls due to reorder buffers being full (`0x00D5`);

[1]The PXA255 in PLEB 2 only provides two counters, but other variants like the PXA270 in the I-Box provide four

- DRAM accesses due to page conflicts (`0x04E0`).

Of those shown in Listing 1, three (`0x04E0`, `0x27E` and `16D`) have positive coefficients, and one (`0x00D5`) has a negative coefficient. The first three are related to the number of memory accesses, and so an increasing frequency results in an increased number of cycles, and a positive coefficient for each. The fourth is related to the number of stall cycles caused by the re-order buffers being full — an effect reduced by an increased frequency, resulting in a negative coefficient. A more informed discussion of these coefficients is difficult without a detailed knowledge of the platform's internals.

We selected the terms used in the power model after limiting the possible parameters to those events chosen for the performance model. This is shown in Figure 7.8. Listing 1 shows that we chose a model with 7 terms including the intercept. Like other platforms, $f_{cpu}V^2$ is selected first. The number of L2 cache misses (PMC 027E) is chosen in both the V^2_{cpu} domain and in the constant-voltage domain, with a negative coefficient in the former and positive in the latter. These memory-related events clearly have a negative effect on the CPU core power (since the core stalls during memory operations) but a positive effect on the constant-voltage domain (where the memory itself consumes power). DRAM accesses due to page conflicts (PMC 04E0) is clearly related to increased memory accesses and the system's power is therefore related to it. Curiously, the number of quadword write transfers (PMC 016D) is chosen in the CPU domain, but the small coefficient means that this term contributes only a minor amount. It is possible that quad-word transfers combines with the number of L2 cache misses, and that larger transfers result in longer spent stalled (and so a lower power in the CPU, hence the negative coefficient). Also curious is the negative coefficient associated with the CPU frequency, although this is, again, a small coefficient and may relate to the contribution of off-chip signalling to the CPU power.

We gathered data using for benchmarks using the these events. The performance model resulting from this event selection and the cycle counter fits the characterisation data with a coefficient of determination of $R^2 = 0.98$, indicating an excellent fit. The power model also resulted in $R^2 = 0.98$.

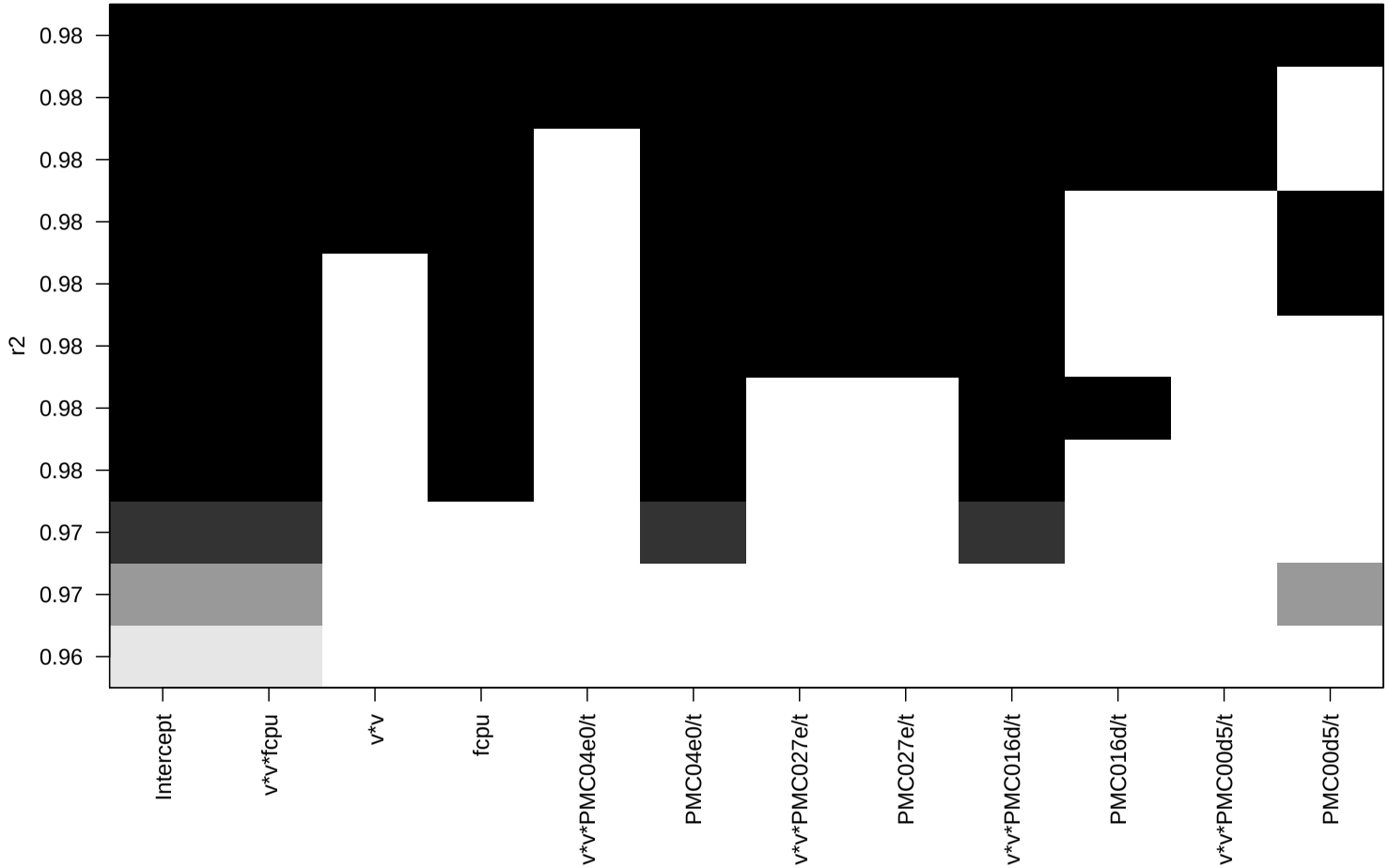


Figure 7.8: Parameter selection for the Opteron power model. The X-axis labels correspond with the terms in Equation 4.12 for Opteron. See Section 7.1.3 for an explanation of this plot.

7.3.3 Latitude

The Latitude is less straightforward when building a model. An initial parameter selection for the time model, based on the SPEC CPU2000 reference benchmarks that took three months to execute, is shown in Figure 7.9. Despite the high R^2 shown, the characterised models did not have a good predictive ability when validated using SPEC CPU2006. This is because of the complexity of this platform, and the limited number of performance counters available.

We had great difficulty obtaining consistent data on the Latitude, due to the benchmarking difficulties discussed in Section 7.1.2. The parameter selection procedure is particularly sensitive to benchmarks which do not run consistently, since it relies on multiple executions to measure each event at the same setting. On the latitude these inconsistencies are observable for `mcf`, `swim` and `twolf` among others.

Also, building a performance model with only two performance counters leads to a model whose accuracy is highly sensitive to the particular workload. To build an accurate performance model, we used a combination of benchmarks from both CPU2000 and CPU2006 for the selection process. This biases the parameter selection toward counters which perform an adequate performance estimation of both CPU2000 and CPU2006.

This highlights a weakness in this parameter selection methodology: the accuracy of the selected model depends on the coverage of the workloads used for selection. To have complete coverage, the parameter selection workloads must, exhibit all of the characteristics of the workloads to be executed. In some cases, the number of counters, and the events they can count, will not be sufficient to obtain a general model, and some workloads will be badly mispredicted.

We re-ran the benchmarks using our improved benchmarking technique, but to complete the necessary benchmarking in a reasonable timeframe we chose a subset of the performance counters with a mixed set of workloads from both CPU2000 and CPU2006. The following two events are selected, and are indicators of memory activity:

- Number of completed burst transactions (`0x006E`);
- Number of lines removed from the L2 cache (`0x1F26`).

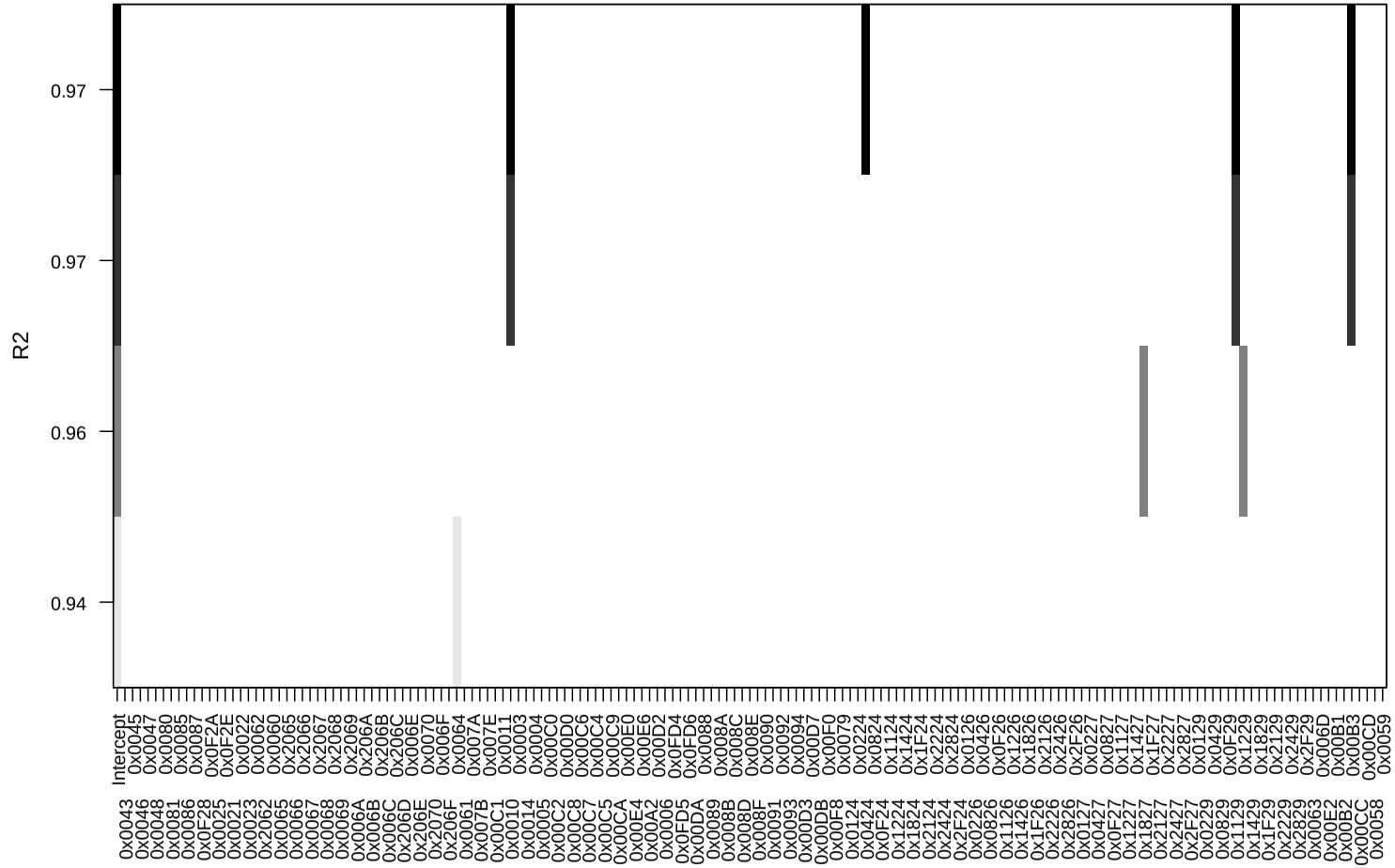


Figure 7.9: Original parameter selection for the Latitude time model. The X-axis labels correspond with the terms in Equation 4.29 for the Latitude. See Section 7.1.3 for an explanation of this plot.

Again, the power model was selected using only the events selected by the performance model. This gives a model based on $f_{cpu}V^2$ only.

When characterised using the CPU2000 data, this resulted in a performance model with $R^2 = 0.98$ and a power model with $R^2 = 0.96$, which, given the limited number of performance counters and complexity of the platform, can also be considered a good fit.

7.4 Adaptation to workload

At each time slice, the Koala implementation selects a frequency setting based on the system's energy-management policy as described in Chapter 6.

Figure 7.10 shows how Koala responds differently to four different workloads on the Latitude using a minimum-energy policy, from the CPU-bound `gzip` to the memory-bound `swim`. `swim` is periodic, with varying frequencies, whereas `gzip` remains at the highest frequency after a short initialisation period at the lowest setting on startup.

Figure 7.11 shows how Koala adapts to the memory-and CPU-bound phases of the `swim` benchmark using a minimum-energy policy on the Opteron server. The lower graph shows how the addition of the frequency switch latency terms greatly reduces the number of frequency switches, by introducing a bias against switching. This improved accounting saves an additional 1% of the system energy in this case.

Three time-slice lengths were trialled, the commonly-used 100 Hz and 250 Hz, as well as 1000 Hz (sometimes used in real-time applications). The system worked well at both 100 Hz and 250 Hz. At 1000 Hz the accuracy decreased markedly. We attribute this to two effects: firstly, the frequency switch overhead becomes more significant compared to the time slice length. Secondly, averaging over a shorter period makes the result more sensitive to short-term fluctuations in the workload. We did not investigate this issue further, as the (more standard) longer time slices worked well.

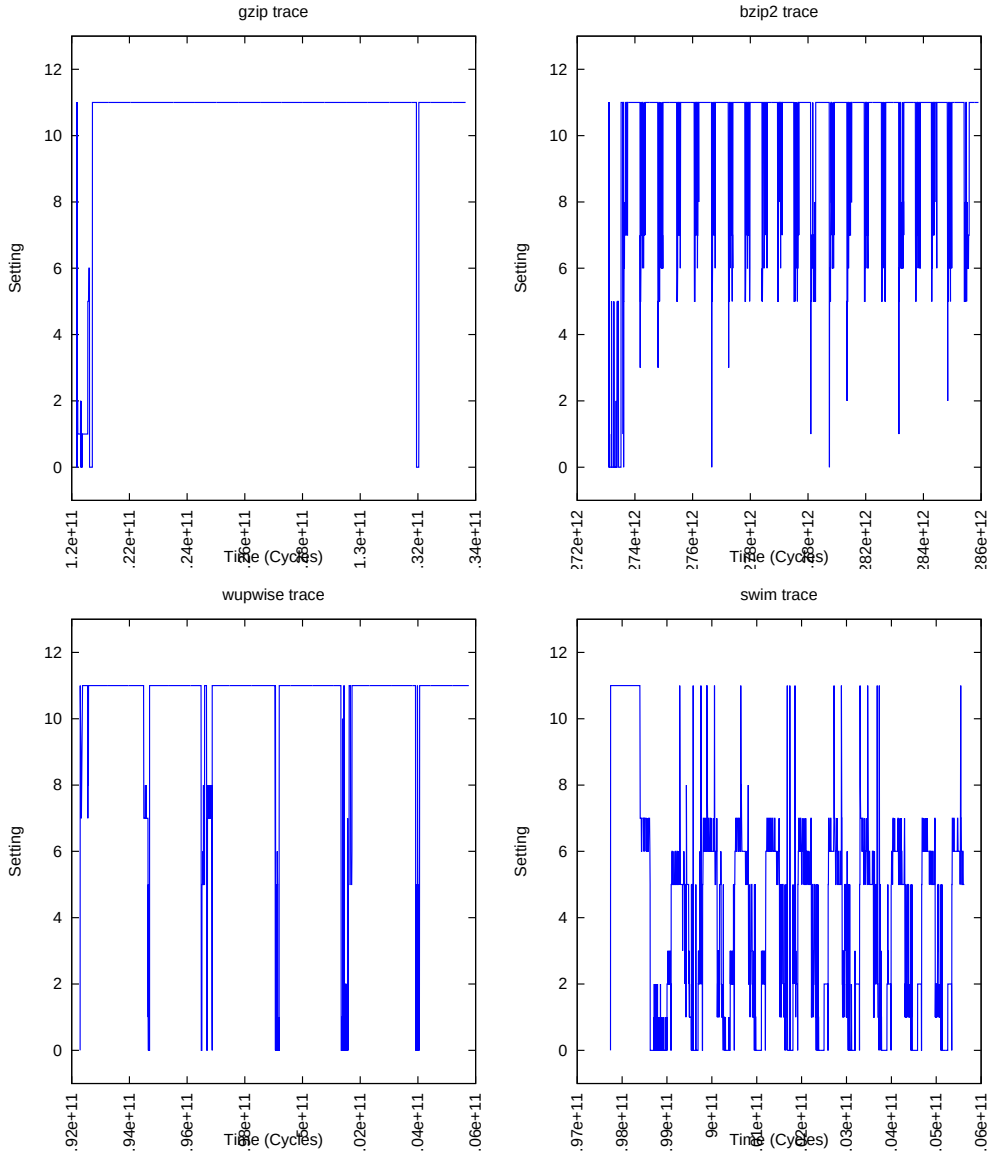


Figure 7.10: Koala behaviour for the first 2000 time slices of (clockwise from top left) `gzip`, `bzip2`, `wupwise` and `swim`.

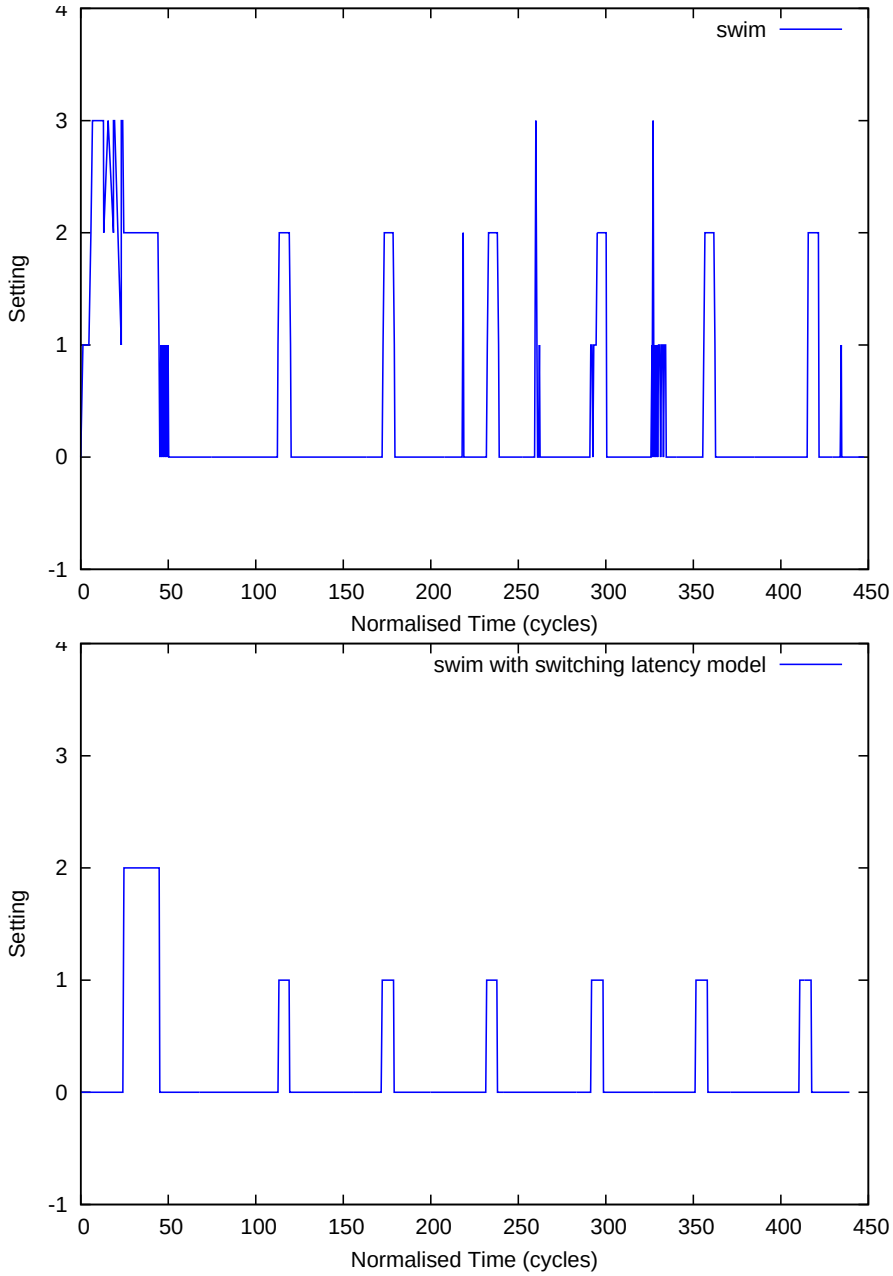


Figure 7.11: Koala behaviour for the first 1000 time slices of `swim` on the server with and without latency terms.

7.5 Model accuracy

Figure 7.12 shows the performance and energy use of 27 SPEC CPU2006 benchmarks under the minimum-energy policy ($\alpha = 0$ in Equation 5.1) on the Opteron-based server. The benchmarks omitted for clarity are all CPU-bound and thus uninteresting for this platform (since they are easily predicted). The energy saving is between 0 and 15% of the *total* system energy compared with the maximum performance setting. The latitude showed even more significant energy savings (see Figure 7.15). For some benchmarks (the memory-bound `swim`) Koala was able to save 26% of the energy for less than a 1% loss in performance. This amounts to 46% of the dynamic energy.

For most benchmarks there is good agreement, generally within a few percent, between the actual performance and energy use and the estimates produced by the model, which indicates that the approach generally works well. However, there is a single case where the model fails spectacularly on both platforms, mispredicting performance of the LBM benchmark by 25% (107 vs. 86) and energy by 20% (68 vs. 85) on the server. The system still saves energy in this case — while the models fail to predict accurately, they still provide a good heuristic for frequency selection. More accurate models would simply allow more reliable, predictable energy savings. LBM was the only such case we observed where the models failed in this way. The possible reasons for this inaccuracy are discussed in Section 7.3.

We enabled idle energy in the model (which adjusts the energy for any extra idle time created thanks to frequency increases), and ran benchmarks over a fixed time period. In this case, on both platforms, the energy-optimal frequency is almost always the minimum.

7.6 Policy

Figures 7.13 and 7.14 show how Koala implements the *maximum-degradation policy* (see Chapter 5). Curves in the top graph in each figure show the actual performance of representative benchmarks under varying performance goals. The thick diagonal line represents the ideal response, under perfect operation all curves should be on or just above this line.

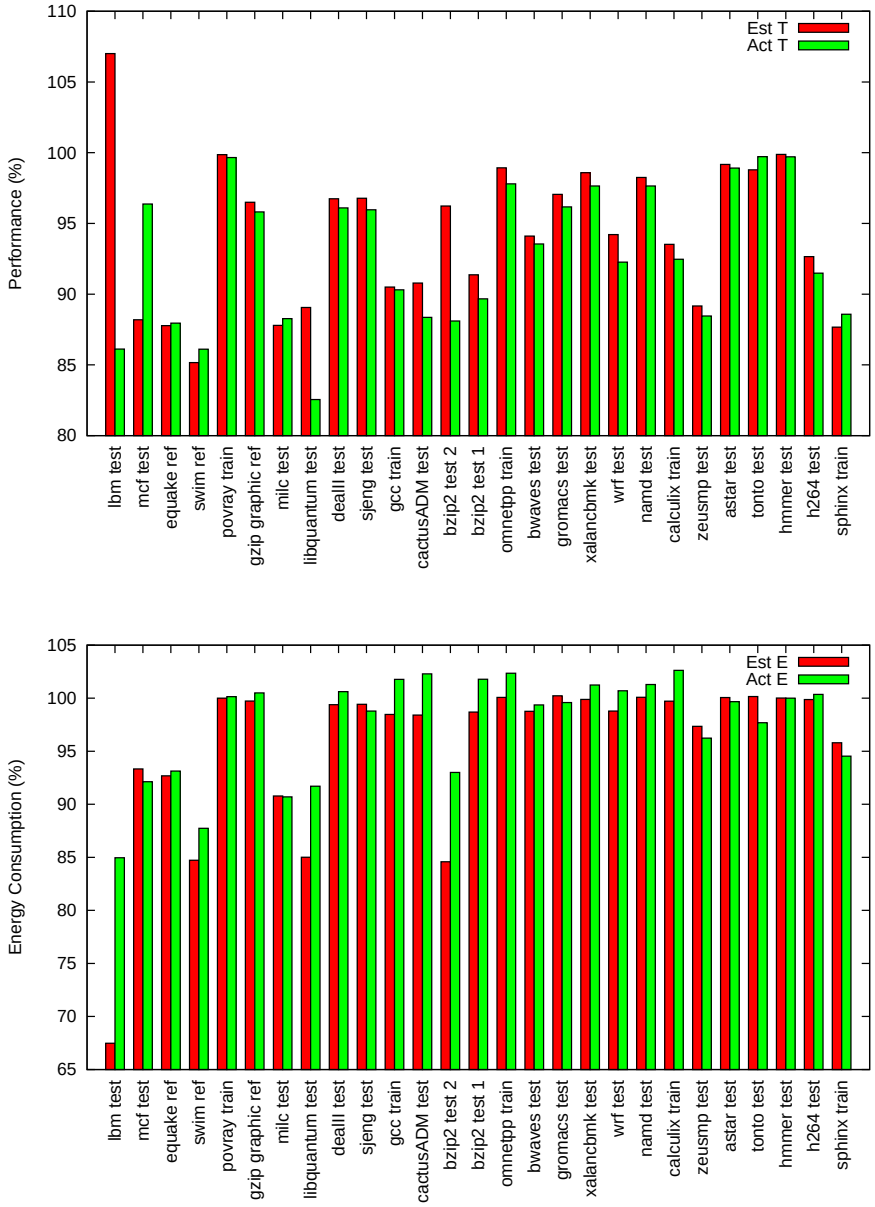


Figure 7.12: Comparison of estimated vs. actual performance (top) and energy (bottom) for the minimum-energy policy on the server platform.

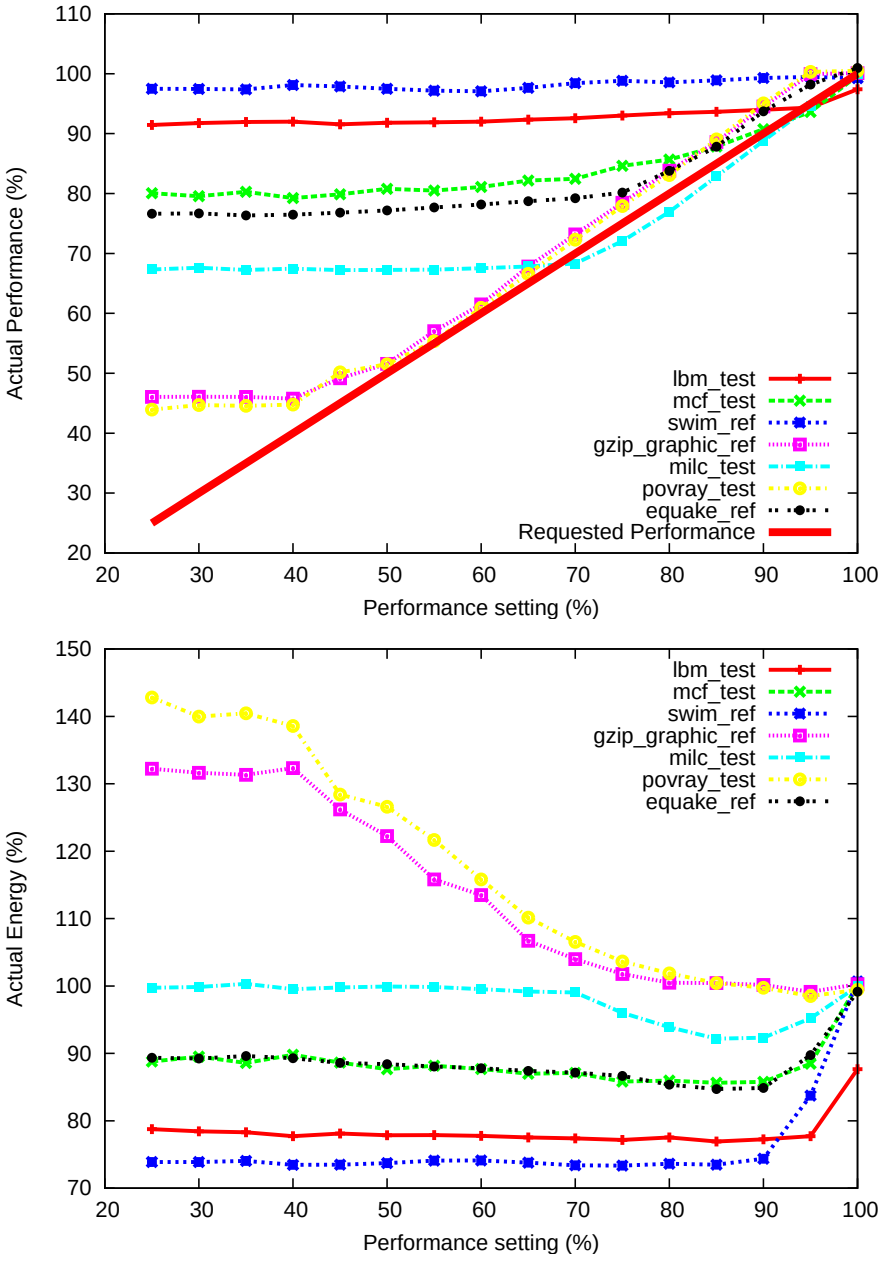


Figure 7.13: Maximum-degradation policy on the Latitude

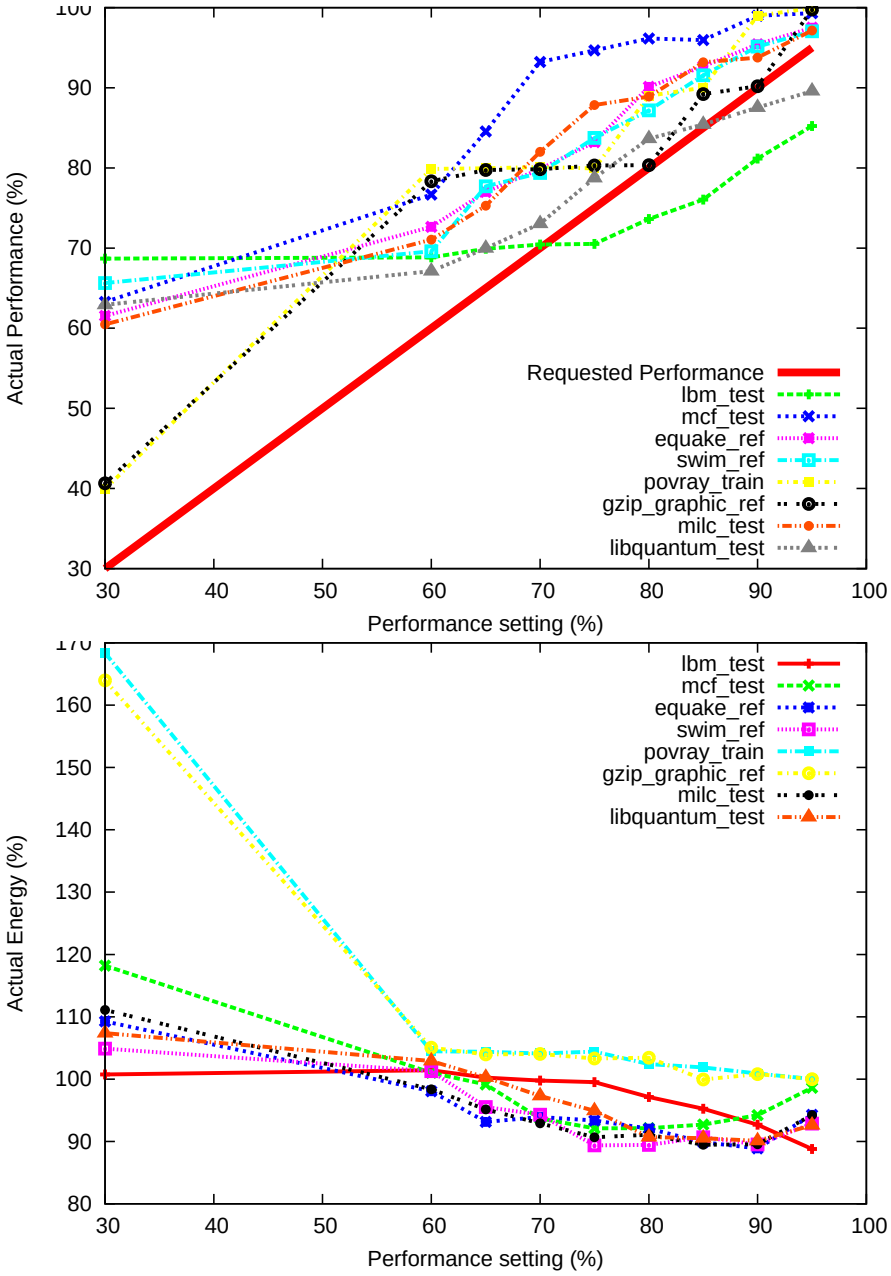


Figure 7.14: Maximum-degradation policy on the Opteron

On the Latitude, it can be seen that actual performance mostly gets close to the target. Some benchmarks run at slightly less than the target performance, this results from the discrete setpoints, inaccurate performance estimation, and Koala's adjustment lagging behind changes of workload behaviour.

The horizontal lines extending to the left of the graph are a result of the limited frequency range available — the processor cannot be throttled enough to reach the lower performance targets. This effect is particularly strong for the memory-bound benchmarks.

The bottom graph in the two figures shows the corresponding energy use. We can see that the maximum-degradation policy saves significant energy (up to about 25%) on memory-bound benchmarks, but actually wastes energy on CPU-bound benchmarks, clearly indicating that this policy is not suitable for a wide range of workloads.

The reason is that a CPU-bound benchmark executes in a constant number of cycles, irrespective of the core frequency. Lower frequency leads to a longer overall execution time, which increases the static energy used. This is the effect we have shown in Figure 3.1, which indicates that race-to-halt is the best policy for CPU-bound workloads on this platform.

The Opteron behaves in a similar fashion, although, because of the much smaller number of settings there is rarely one that satisfies the performance request accurately. The result is a sloppier adherence to the ideal line in Figure 7.14.

Figure 7.15 shows that the *generalised energy-delay policy* produces much better results. As expected, $\alpha = 1$ yields the highest performance while $\alpha = 0$ produces the lowest energy consumption (with a slight aberration of the pathological `lbm` benchmark), and intermediate values produce intermediate results. The graphs also show that the standard energy-delay policy ($\alpha = 0.33$) produces, for most benchmarks, an energy use close to that of the minimum-energy setting, for a moderate performance degradation. Negative values of α are not useful for energy management, but can be used to throttle power dissipation for thermal management.

Figure 7.15 also shows that some benchmarks, specifically the notorious `lbm`, fail to reach more than about 90% performance at $\alpha = 1$. This is obviously a result of incorrect performance estimates leading Koala to choosing an incorrect

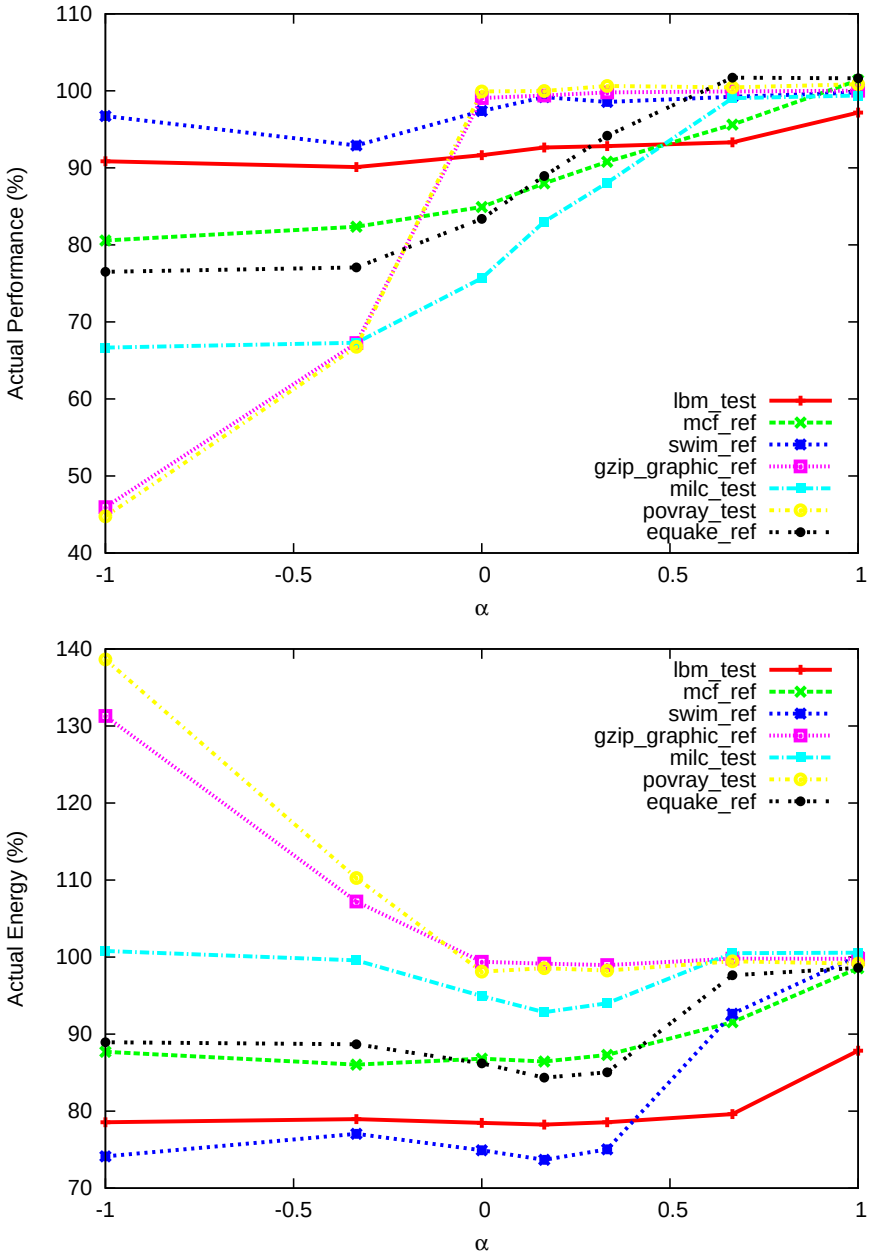


Figure 7.15: Generalised energy-delay policy on the Latitude.

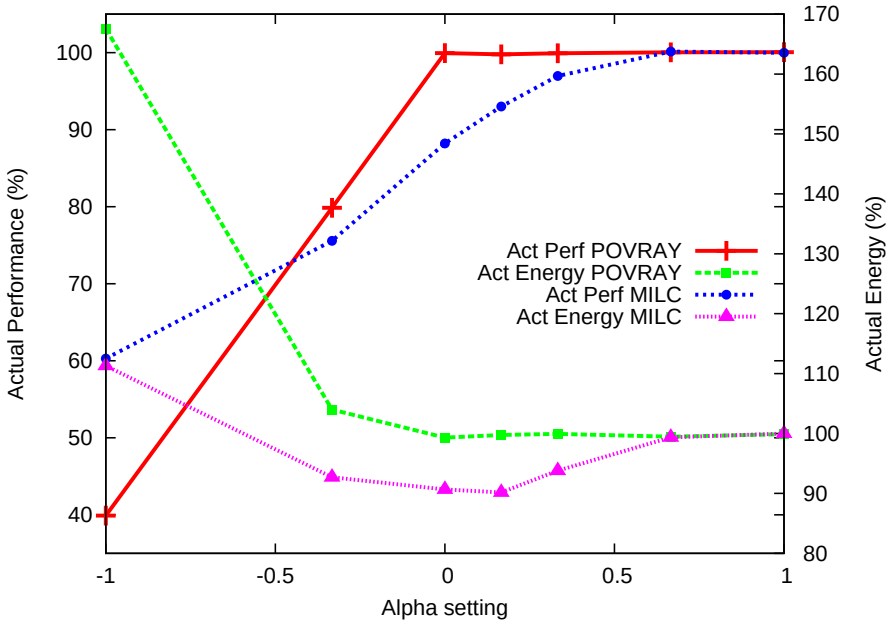


Figure 7.16: Generalised energy-delay policy on the server.

setting. (This is confirmed by `lbm` also failing to reach its maximum-frequency energy use at $\alpha = 1$).

The strength of the generalised energy-delay policy with its single global parameter is particularly evident when comparing the CPU-bound `povray` with the memory-bound `milc` (Figure 7.16). `povray` is not slowed down at all for positive α, since there is no energy to save. For the same α values, `milc` is scaled in order to save energy. The policy only sacrifices performance when there is a corresponding energy benefit. Below $\alpha = 0$, `povray` is scaled aggressively to reduce the system power consumption, but with a corresponding increase in energy used.

Enabling the switch overhead model, we see the number of frequency switches reduced for most policies and benchmarks (in the case of `swim` on the server, this is about 9%) because the model predicts a higher performance for the incumbent frequency, which it therefore favours slightly. We also see the energy savings and model accuracy increase when using these models. We use well-behaved benchmarks here to highlight the effect of the switch overhead model.

7.7 System Evaluation

7.7.1 Multi tasking

Figure 7.17 shows the effect of running a multi-tasking workload consisting of memory-bound `swim` and CPU-bound `gzip`. The top part of the figure shows that the energy and performance predictions of the combined workload under the minimum-energy policy is about as good as for separate executions, and the energy saved is about the average of the savings for the two loads when run independently, as can be expected. The trace in the bottom graph shows how Koala adapts the setting for the two processes independently.

7.7.2 Higher-level policies

One advantage of the generalised energy-delay policy is that the single parameter (α), allows the system to adapt to changing energy-management objectives.

As a demonstration we implemented a daemon which monitored the laptop's battery state of charge using ACPI. At capacities greater than 70%, the daemon sets α to 1, and the system runs at maximum performance. As the battery is depleted, the daemon lowers α until the battery gets below 30% and then α is set to 0, i.e. minimum-energy. Figure 7.18 shows how the performance-energy tradeoff changes as the battery depletes while running `mcf`

Another high-level policy on top of the generalised energy-delay policy emulates the `ondemand` governor in Linux: CPU scaling is controlled based on the available idle time. During periods of low utilisation, α is lowered towards 0 (the minimum-energy setting), and in times of high load, α is increased toward 1.0 (the maximum performance setting). The frequency at which this happens is a user variable.

More complex high-level policies could be constructed in order to meet real-time deadlines and other performance goals. It would, for example, be possible to combine a bounded-performance policy with the generalised energy-delay policy, choosing a candidate setpoint that minimises Equation 5.1 while maintaining a specified performance (as estimated using the same performance-bound model). How to use these tools is the topic of future DVFS research.

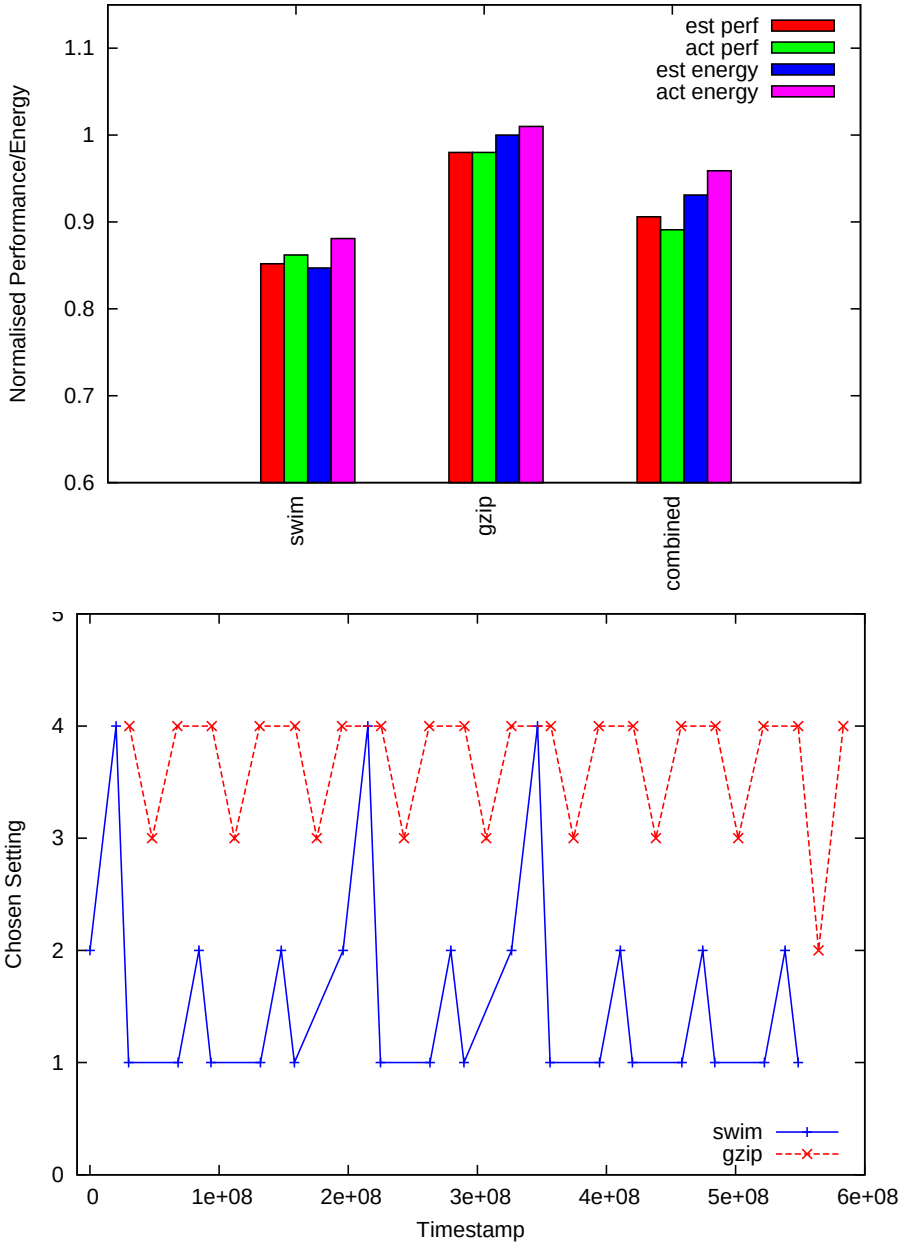


Figure 7.17: Koala multi-tasking on the server

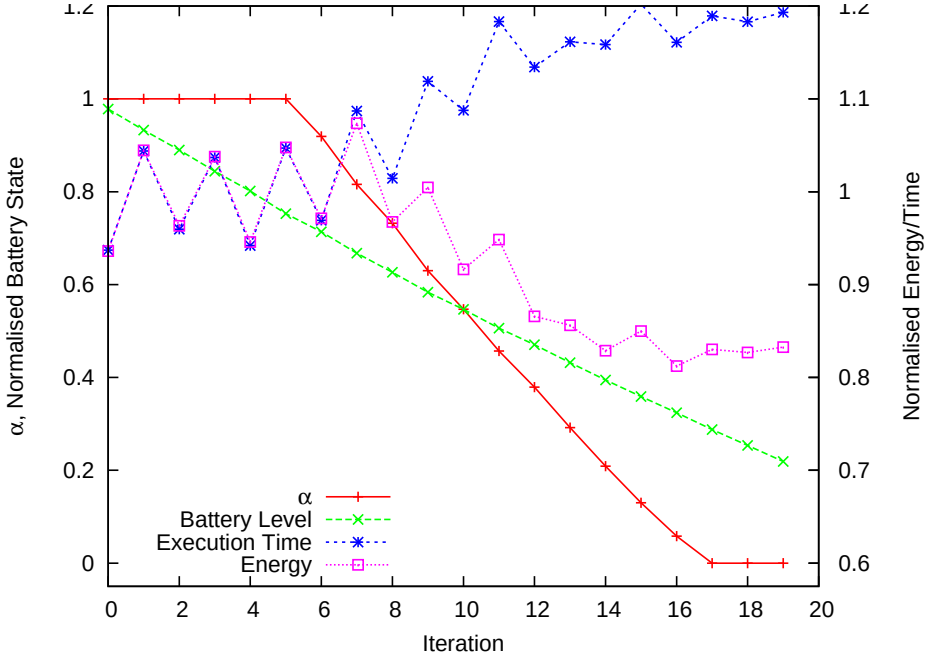


Figure 7.18: Using the Latitude's battery state of charge to drive the power management policy.

7.7.3 Calculation overheads

A major concern when implementing Koala was the overhead introduced, since this could reduce the energy savings and be detrimental to performance. In order to minimise the overhead, all calculations were performed in fixed point, and pre-calculated lookup-tables used where appropriate.

Reducing the number of setpoints considered in the calculations also reduces the overhead. Therefore, setpoints which are never chosen should be excluded.

On the server, the memory frequency drops to 160MHz (vs. 200MHz) at the lowest setting ($f_{cpu} = 800MHz$). Our initial model for this platform took this into account, but because of the lower memory frequency, this setting was never selected. We removed this setting from the model, and in doing so, simplified the performance model (further reducing overhead).

In addition, we experimented with different policies for selecting a setting. Instead of calculating the predicted performance and energy for all possible settings, the policies execute the energy and performance models as they search the

settings. In the common case, where the current setting is the most optimal, only four settings need be calculated (current, above, below, and maximum). On the Laptop, with 11 settings, this reduces the number of settings by nearly a factor of three. The CPU may not be at the energy optimal frequency for every timeslice, and the effect on energy savings will depend on the workload characteristics. Most benchmarks we observed were not disadvantaged by this optimisation.

Depending on the memory behaviour of the workload that ran in the previous slice, the performance of the calculations may be affected by a small number of cache misses to bring in model parameters from main memory. The performance of the calculations was particularly affected when running the memory-bound `swim`. The cost of a cache miss on the server is 200 cycles. Therefore cache misses are a significant portion of the total cost of the calculations.

We ran the set of benchmarks on kernels with and without Koala enabled. On both the laptop and the server, for both a single task executing, and for two concurrent tasks, the mean performance difference was well within the standard deviation of the benchmarks. To emphasise the overheads, the timer tick frequency was increased up to 1000Hz, but the mean difference in performance between the two kernels was still unmeasurable.

CHAPTER 8

CONCLUSIONS

> *"To punish me for my contempt of authority, Fate has made me an authority myself."* – Albert Einstein, 1930

Operating-system level power management is a critical component when reducing the energy used by computing systems, from small mobile devices to high-end servers. It is a complicated problem, and the heuristic approach often used by modern operating systems generally leads to poor results, since optimal decisions can only be made with intimate knowledge of both the platform and the workload.

This thesis presents the Koala approach, which solves this problem using a combination of two techniques. Firstly, it gives the OS insight into the relevant properties of a platform and an application, with a model that allows the OS to obtain accurate estimates of a running process's power consumption, as well as the performance and power response to frequency scaling. The models can be used for prediction when combined with a temporal locality argument, which gives a sufficient accuracy in the case of the workloads we tested.

These models encapsulate a knowledge of the underlying hardware. If the models reflect the behaviour of the system, Koala's policies will naturally adapt to that behaviour. Koala can therefore replicate the effect of many previously-presented policies which dealt with challenging hardware behaviour explicitly, such as those which dealt with the complexities of non-ideal batteries.

The second component is a policy that allows the OS to tune the system's operation towards an energy-management objective. The generalised energy-delay policy contains a single parameter which the OS can use to run the system at maximum performance, minimum energy, reduced thermal load, or intermediate values representing other trade-offs between performance and energy. The OS can tune this parameter to adapt to a changing energy-management objective.

Rather than treating every process the same, Koala adjusts individual processes differently in order to achieve the best overall result. Specifically, memory-bound processes, where small reductions in performance result in large energy savings, are throttled more than CPU-bound processes, where small energy savings come at the cost of a significant performance penalty. The approach led to impressive results, with one test showing more than 26% of the system's energy saved for less than a 1% performance loss. On one platform we studied, our system even *improved* some applications' performance!

The main shortcoming is in the accuracy of the models for specific worst-case benchmarks. In both systems we examined closely, a single benchmark's behaviour was poorly predicted. While in both cases energy was still saved for this benchmark, the significant errors in the prediction of its performance and energy use certainly lead to sub-optimal power management. The reason for these lapses in accuracy are likely a result of an insufficient characterisation set, co-linearities in the selected parameters, or simply because the hardware-provided performance events are not sufficient to predict the system's behaviour for that particular workload. In any of these cases, the accuracy could be improved by hardware manufacturers adding support for Koala's performance and energy models: with a knowledge of the system's internal operation, and the ability to add specific counters, it would be possible to build a highly-accurate model with a low calculation overhead.

While the techniques presented in this thesis allow for excellent energy management, the improvements outlined later in this chapter will improve the modelling accuracy on a wider range of platforms and workloads (such as multi-core systems and I/O-based workloads), reduce the calculation overheads and provide the presently-missing high-level policy components which automatically responds to a system's performance and thermal demands.

8.1 Contributions

The contributions of this thesis are outlined explicitly here.

First, through the experimental examination of a large range of platforms, this thesis showed that the traditional academic approach to frequency scaling was at

odds with the behaviour of real platforms. We identified the cause of many of these deviations, and showed that it was impossible for any single heuristic to accurately, or even effectively, manage the energy consumption in that environment. This is the first presentation of many of these effects that we are aware of.

The operating system must, therefore, be aware of the platform's and workload's behaviour. This thesis presented a method of encapsulating that knowledge in a platform model which can estimate the parameters which need to be controlled (performance and power) rather than attempting to control the parameters which merely have some effect on the these (frequency and voltage).

The thesis refined the previously existing single-setting energy modelling techniques into models which allow the prediction of the performance and energy of a system at alternate power-management settings. Moreover, it presents a methodology for building such models. The resulting equations took into account many of the power-management challenges identified, and present no obstacle to representing all of these. The model's output is real-world and testable: the system's performance and energy consumption. These models are useful in themselves, allowing user feedback about the system's power consumption in the absence of dedicated sensors. We also considered methods of integrating idle-mode management, switching-latency overhead and other dynamic effects. These form extensions to the basic model which improve the accuracy of the power management scheme.

We also developed a new low-level policy, the *generalised energy-delay* policy, which allows a continuous trade-off between power, energy and performance. The policy, like the majority of the Koala framework, is generic for any platform and workload, given the above power and performance models.

Lastly, we conducted a thorough evaluation of the new power-management scheme using a real operating system, real platforms, real workloads, real measurements, with stated assumptions. Our assumptions are not fundamentally problematic, but serve to temporarily limit the scope of this work for tractability. We could demonstrate that the system saves substantial amounts of energy in some cases, and optimally trades power and performance. The implementation includes user-land tools for use in modelling, and in run-time monitoring. We showed that the overheads involved were negligible.

8.2 Future Work

This work prompts a new focus in active power management, with a variety of future work.

It is imperative that we relax the assumptions made when developing Koala, since these limit the types of systems on which the framework can be effectively deployed. While these assumptions are substantial, we cannot conceive of any fundamental problems with the approach. For example, a solution to the single-core limitation will predict the effect of other cores' activity on the core being scaled. Similarly, memory contention must be taken into account in systems using DMA. I/O can be modelled using similar techniques to those we've used for the computing components. We are presently investigating the latter problem, building extensive I/O models for different classes of peripherals based on information gathered in the operating system.

Hardware support should be built in to systems concerned with power management. We have seen technologies such as IEM and the Core i7 processor with this kind of support, and this could be further tailored for Koala. Hardware manufacturers should provide:

- dedicated hardware for constructing workload-specific power and performance models. This would include dedicated counters and specific PMC events. Such hardware would dramatically improve the accuracy of the models by providing a sufficient number of counters. If the calculation overheads associated with the models became unreasonable, this too could be implemented in hardware.

- power, temperature and other sensors suitable for those models, or suitable for measurement-based estimation;

- dedicated power-management controllers such as those implemented in the Core i7 and Montecito processors. Implementing parts of Koala in firmware at the hardware level would allow for much finer grained frequency scaling, potentially improving the energy savings possible. Such a controller requires a very tight coupling with the operating system.

This thesis has provided and validated Koala's low-level policies, but has only provided toy policies at the high-level. These high-level policies should determine the importance of performance, power and energy, expressed as α. Much of the related work discussed in Chapter 2 is now relevant, since these unwittingly deal with this problem. Integrating Koala-derived information into the scheduler would also give significant benefits, allowing fairer time-sharing. Processes which are throttled could be given extra CPU time according to the performance loss calculated by Koala. Every process would then suffer an equal apparent performance loss. Other modifications to the scheduler could reduce frequency switching overheads by scheduling processes with identical setpoints in sequence.

Lastly, some attention needs to be paid to practicality. The experiments in this thesis took many thousands of man-hours to devise and conduct. Hardware manufacturers could provide a models for their hardware, alleviating these concerns, and putting the task of model-building the hands of those with the most detailed relevant knowledge. Even roughly-calibrated models would provide a basis for later characterisation during a burn-in phase.

8.3 Final words

The development of Koala presents an exciting reality: that academic-developed DVFS techniques can be applicable to real-world systems. By effectively handling the platform oddities which are not traditionally examined, this work provides a predictable environment for making power-management decisions. While the systems and workloads examined during this thesis are necessarily limited, the fundamental concepts and policies can be applied to any system for which a model can be built. In that context, this is the first development and implementation of a truly generic power-management infrastructure.

BIBLIOGRAPHY

[1] Nevine AbouGhazaleh, Bruce R. Childers, Daniel Mosse, and Rami G. Melhem. Integrated CPU cache power management in multiple clock domain processors. In *Proceedings of the 3rd International Conference on High Performance Embedded Architectures and Compilers*, pages 209–223, Göteborg, Sweden, January 27-29 2008.

[2] Andrea Acquaviva, Luca Benini, and Bruno Ricco. Software-controlled processor speed setting for low-power streaming multimedia. *IEEE Transactions on CAD ICAS*, 20(11):1283–1292, November 2001.

[3] Tarek A. AlEnawy and Hakan Aydin. On energy-constrained real-time scheduling. In *Proceedings of the 16th Euromicro Conference on Real-Time Systems (ECRTS04)*, pages 165–174, July 2004.

[4] Apple Computer, Inc. Darwin web page. URL http://developer.apple.com/Darwin, 2009.

[5] ARM Limited. Application Note 32: The ARMulator. URL http://infocenter.arm.com/help/topic/com.arm.doc.dai0032f/AppNote32_ARMulator.pdf, 2003.

[6] ARM Limited. *IEM Software Technical Overview*, r0p1 edition, 2005.

[7] ARM Limited. Intelligent energy manager (IEM) hardware control system in the ARM1176JZF-S development chip. Application Note 172, ARM Limited, November 2006.

[8] ARM Limited. *Intelligent Energy Manager documentation*, 2007.

[9] ARM Limited. IEM technology web page. URL http://www.arm.com/products/esd/iem_home.html, 2008.

[10] ARM Limited. *Intelligent Energy Controller Technical Reference Manual (ARM DDI 0304C)*. ARM Limited, r0p1 edition, 2008.

[11] ASUSTek COMPUTER INC. *Eee PC User's Manual, Windows XP Edition, Eee PC 901 Series*, June 2008.

[12] T. Austin, E. Larson, and D. Ernst. SimpleScalar: an infrastructure for computer system modeling. *IEEE Computer*, 35(2):59–67, February 2002.

[13] Hakan Aydin, Vinay Devadas, and Dakai Zhu. System-level energy management for periodic real-time tasks. In *Proceedings of the 27th IEEE Real-Time Systems Symposium*, pages 313–322, Rio de Janeiro, Brazil, December 2006. IEEE Computer Society Press.

[14] Jan-Derk Bakker, Erik Mouw, Marc Joosen, and Johan Pouwelse. The LART pages. http://www.lart.tudelft.nl, 2000.

[15] Nikhil Bansal, Kanishka Lahiri, and Anand Raghunathan. Automatic power modeling of infrastructure IP for system-on-chip power analysis. In *Proceedings of the 20th International Conference on VLSI Design*, pages 513–520, Jan. 2007.

[16] Nikhil Bansal, Kanishka Lahiri, Anand Raghunathan, and Srimat T. Chakradhar. Power monitors: A framework for system-level power estimation using heterogeneous power models. In *Proceedings of the 18th International Conference on VLSI Design*, 2005.

[17] Frank Bellosa. The benefits of event-driven energy accounting in power-sensitive systems. In *Proceedings of the 9th SIGOPS European Workshop*, pages 37–42, Kolding, Denmark, September 17–20 2000.

[18] Frank Bellosa, Simon Kellner, Martin Waitz, and Andreas Weißel. Event-driven energy accounting for dynamic thermal management. In *Proceedings of the Fourth Workshop on Compilers and Operating Systems for Low Power (COLP 03)*, New Orleans, LA, USA, September 27 2003.

[19] Lloyd Bircher, Madhavi Valluri, Jason Law, and Lizy John. Runtime identification of microprocessor energy saving opportunities. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 275–280, San Diego, CA, USA, August 8–10 2005.

[20] W. Lloyd Bircher and Lizy K. John. Complete system power estimation: A trickle-down approach based on performance events. In *Proceedings of the IEEE In-*

ternational Symposium on Performance Analysis of Systems and Software, pages 158–168, San Jose, CA, USA, April 25–27 2007. IEEE Computer Society.

[21] Scott A. Brandt, Scott Banachowski, Caixue Lin, and Timothy Bisson. Dynamic integrated scheduling of hard real-time, soft real-time and non-real-time processes. In *Proceedings of the 24th IEEE Real-Time Systems Symposium*, Cancun, Mexico, December 2003.

[22] D. Brooks, P. Bose, S. Schuster, H. Jacobson, P. Kudva, A. Buyuktosungolu, J-D Wellman, V. Zyuban, M. Gupta, and P. Cook. Power-aware microarchitecture: design and modeling challenges for the next generation microprocessors. *IEEE Micro*, 20:26–44, 2000.

[23] David Brooks, Vivek Tiwari, and Margaret Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th Annual International Symposium on Computer Architecture (ISCA)*, pages 83–94, 2000.

[24] F. Chang, K. Farkas, and P. Ranganathan. Energy-driven statistical profiling: Detecting software hotspots. In *Proceedings of the Workshop on Power Aware Computing Systems*, 2002.

[25] Kihwan Choi, R. Soma, and M. Pedram. Fine-grained dynamic voltage and frequency scaling for precise energy and performance tradeoff based on the ratio of off-chip access to on-chip computation times. *IEEE Transactions on CAD ICAS*, 24(1):18–28, January 2005.

[26] Kihwan Choi, Ramakrishna Soma, and Massoud Pedram. Dynamic voltage and frequency scaling based on workload decomposition. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 174–179, August 2004.

[27] C.K.Y. Chun. eXtreme energy conservation for mobile communications. In *Proceedings of the 2006 IEEE International SOC Conference*, pages 185–188, September 2006.

[28] Sung Woo Chung and Kevin Skadron. Using on-chip event counters for high-resolution, real-time temperature measurement. In *Proceedings of the 10th Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronics Systems (ITHERM06)*, pages 114–120, June 2006.

[29] Todd Cignetti, Kirill Komarov, and Carla Ellis. Energy estimation tools for the Palm. In *Proceedings of ACM MSWiM 2000: Modeling, Analysis and Simulation of Wireless and Mobile Systems*, 2000.

[30] Gilberto Contreras and Margaret Martonosi. Power prediction for Intel XScale processors using performance monitoring unit events. In *Proceedings of the International Symposium on Low Power Electronics and Design*, San Diego, CA, USA, August 2005.

[31] Dell Corporation. DELL Latitude D600 product specification sheet: SS_LAT_D600_081304. http://www.hackerconsulting.ca/Downloads/DELL%20laptops/dell%20D600%20specs.pdf, August 2004.

[32] Intel Corporation. Intel XScale technology web page. http://developer.intel.com/design/intelxscale/, 2003.

[33] Intel Corporation. Mobile Intel Pentium III processors web page. http://www.intel.com/support/processors/mobile/pentiumiii/ss.htm, 2003.

[34] V. Delaluz, M. Kandemir, N. Vijaykrishnan, A. Sivasubramaniam, and M. J. Irwin. Memory energy management using software and hardware directed power mode control. Technical Report CSE-00-004, Department of Computer Science and Engineering, Pennsylvania State University, 2000.

[35] Advanced Micro Devices. AMD PowerNow! technology web page. URL http://www.amd.com/us-en/Processors/ProductInformation/0,,30_118_9486_964,00.html, 2000.

[36] Jack Doweck. Inside intel core microarchitecture and smart memory access. Whitepaper, Intel Corporation, 2006.

[37] Evelyn Duesterwald, Calin Cascaval, and Sandhya Dwarkadas. Characterizing and predicting program behavior and its variability. In *Proceedings of the 12th International Conferenece on Parallel Architectures and Compilation Techniques*, September 2003.

[38] Carla Ellis. The case for higher-level power management. In *Proceedings of HotOS*, 1999.

[39] Extech Instruments Corporation. *Extech 380801 Appliance Tester / Power Analyzer Product Datasheet*, August 2008. URL http://www.extech.com/instrument/products/310_399/380801.html.

[40] Xiaobo Fan, Carla S. Ellis, and Alvin R. Lebeck. Synergy between power-aware memory systems and processor voltage scaling. Technical Report CS-2002-12, Duke University, 2002.

[41] Krisztian Flautner, Steven K. Reinhardt, and Trevor N. Mudge. Automatic performance setting for dynamic voltage scaling. In *Mobile Computing and Networking*, pages 260–271, 2001.

[42] Marc Fleischmann. Longrun power management: Dynamic power management for Crusoe processors. Whitepaper, Transmeta Corporation, January 2001.

[43] Jason Flinn. *Extending Mobile Computer Battery Life through Energy-Aware Adaptation*. PhD thesis, Carnegie Mellon University, 2001.

[44] Jason Flinn, Eya de Lara, M. Satyanarayanan, Dan S. Wallach, and Willy Zwaenepoel. Reducing the energy usage of office applications. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms, Heidelberg, Germany*, 2001.

[45] Jason Flinn, Keith Farkas, and Jennifer Anderson. Power and energy characterization of the ItSY pocket computer (version 1.5). Technical Report TN-56, Western Research Laboratory, Compaq, 2000.

[46] Jason Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP)*, 1999.

[47] Jason Flinn and M. Satyanarayanan. Powerscope: A tool for profiling the energy usage of mobile applications. In *Proceedings of the Second IEEE, Workshop on Mobile Computing Systems and Applications*, 1999.

[48] R. Fonseca, P. Dutta, P. Levis, I. Stoica, CA Berkeley, and CA Stanford. Quanto: Tracking energy in networked embedded systems. In *Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI08)*, pages 323–338, 2008.

[49] Freescale Semiconductor, Inc. eXtreme energy conservation: Advanced power-saving software for wireless devices. Whitepaper XTMENRGYCNSVWP, Freescale Semiconductor, Inc., February 2006.

[50] Freescale Semiconductor Literature Distribution Centre, P.O. Box 5405, Denver, Colorado, 80217. *MCIMX31 and MCIMX31L Applications Processors Reference Manual*, December 2008.

[51] Florian Fruth. Run-time energy characterization of the Intel PXA. Study thesis, Operating System Group, University of Erlangen, Germany, April 2005.

[52] Future Technology Devices Itnernational Ltd. *FT232R USB UART IC*, 2009.

[53] *The GNU Compiler Collection*. http://gcc.gnu.org/.

[54] Mohammad Ali Ghodrat, Kanishka Lahiri, and Anand Raghunathan. Accelerating system-on-chip power analysis using hybrid power estimation. In *Proceedings of the 44th annual Design Automation Conference (DAC07)*, pages 883–886, San Diego, California, 2007.

[55] Richard A. Golding, Peter Bosch II, Carl Staelin, Tim Sullivan, and John Wilkes. Idleness is not sloth. In *Proceedings of USENIX Winter*, pages 201–212, 1995.

[56] Kinshuk Govil, Edwin Chan, and Hal Wasserman. Comparing algorithms for dynamic speed-setting of a low-power CPU. In *Mobile Computing and Networking*, pages 13–25, 1995.

[57] Paul M. Greenawalt. Modeling power management for hard disks. In *Proceedings of the Conference on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS94)*, pages 62–66, 1994.

[58] Dirk Grunwald, Philip Levis, Keith I. Farkas, Charles B. Morrey III, and Michael Neufeld. Policies for dynamic clock scheduling. In *Proceedings of the 4th USENIX Symposium on Operating Systems Design and Implementation*, pages 73–86, San Diego, CA, USA, October 2000.

[59] Dirk Grunwald, Philip Levis, Keith I. Farkas, Charles B. Morrey III, and Michael Neufeld. Policies for dynamic clock scheduling. In *Proceedings of the 4th Symposium on Operating Systems Design and Implementation (OSDI)*, pages 73–86, 2000.

[60] Sudhanva Gurumurthi, Anand Sivasubramaniam, Mary Jane Irwin, Narayanan Vijaykrishnan, Mahmut T. Kandemir, Tao Li, and Lizy Kurian John. Using complete machine simulation for software power estimation: The softwatt approach. In *Proceedings for the 8th International Symposium on High Performance Computer Architecture*, pages 141–150, 2002.

[61] Mathhew R Guthaus, Jeffrey S. Reingenberg, Dan Ernst, Todd M. Austing, Trevor Mudge, and Richard B Brown. Mibench: A free, commercially representative embedded benchmark suite. In *Proceedings of the 4th IEEE Annual Workshop on Workload Characterization*, December 2001.

[62] Graham Hellestrand. Using virtual system prototyping technology to optimize real-time systems for power. Whitepaper, VaST Systems Technology, October 2005.

[63] J. L. Henning. SPEC CPU2000: measuring CPU performance in the new millennium. *IEEE Transactions on Computers*, 33(7):28–35, July 2000.

[64] J.L. Henning. SPEC CPU2006 benchmark descriptions. *Computer Architecture News*, 34(4), September 2006.

[65] S. Herbert and D. Marculescu. Variation-aware dynamic voltage/frequency scaling. In *Proceedings of the 15th International Symposium on High Performance Computer Architecture (HPCA09)*, pages 301–312, February 2009.

[66] Hewlett-Packard Corporation, Intel Corporation, Microsoft Corporation, Phoenix Technologies Ltd., and Toshiba Corporation. Advanced configuration and power interface specification. URL http://www.acpi.info/spec.htm, June 2009.

[67] Paul G. Howard. Next generation intel® microarchitecture nehalem. Whitepaper, Microway, Inc, 2009.

[68] Chung-Hsing Hsu and Wu chun Feng. Effective dynamic voltage scaling through CPU-boundedness detection. In *Proceedings of the 2004 Workshop on Power-Aware Computer Systems*, pages 135–149, Portland, OR, USA, December 2004.

[69] Chung-Hsing Hsu and Ulrich Kremer. The design, implementation, and evaluation of a compiler algorithm for CPU energy reduction. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 38–48, San Diego, California, USA, June 2003.

[70] Gumstix Inc. Gumstix basix and connex website. http://docwiki.gumstix.org/Basix_and_connex, July 2009.

[71] Intel Corp. *IA-32 Architecture Software Developer's Manual Volume 3: System Programming Guide*, 2001. URL ftp://download.intel.com/design/Pentium4/manuals/245472.htm.

[72] Intel Corp. *IA-32 Architecture Software Developer's Manual*, 2002. URL http://developer.intel.com/design/pentium4/manuals.

[73] Intel Corp., http://developer.intel.com. *Intel Xscale Microarchitecture for the PXA255 Processor*, March 2003.

[74] Intel Corp. *Intel PXA 255 Processor Developer's Manual*, 2004. URL http://www.xscale-freak.com/XSDoc/PXA255/27869302.pdf.

[75] Intel Corp. *Intel PXA 27x Processor Family Developer's Manual*, 2004. URL http://mmpod.googlecode.com/files/Intel_PXA270_Developers_Manual.pdf.

[76] Intel Corp. *Intel XScale Core Developer's Manual*, 2004. URL http://download.intel.com/design/intelxscale/27347302.pdf.

[77] Intel Corp. *Mobile Intel Atom Processor N270 Single Core*, May 2008. http://www.intel.com/products/processor/atom/techdocs.htm.

[78] Intel Corporation. *Intel Pentium M Processor with 2-MB L2 Cache and 533-MHz Front Side Bus*, July 2005.

[79] Intel Corporation. *Intel Pentium M Processor on 90nm Process with 2-MB L2 Cache*, January 2006.

[80] Intel Corporation. Powertop web site. URL http://www.lesswatts.org/projects/powertop/, July 2009.

[81] Canturk Isci, A. Buyuktosungolu, and Margaret Martonosi. Long-term workload phases: duration predictions and applications to DVFS. *IEEE Micro*, 25(5):39–51, Sept-Oct 2005.

[82] Canturk Isci, Gilberto Contreras, and Margaret Martonosi. Live, runtime phase monitoring and prediction on real systems with application to dynamic power

management. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 359–370, Washington, DC, USA, 2006. IEEE Computer Society.

[83] Canturk Isci and Margaret Martonosi. Runtime power monitoring in high-end processors: Methodology and empirical data. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-36)*, page 93, December 2003.

[84] Canturk Isci and Margaret Martonosi. Phase characterization for power: Evaluating control-flow-based and event-counter-based techniques. In *Proceedings on the 12th International Symposium on High-Performance Computer Architecture*, pages 121–132, February 2006.

[85] K. Itoh, K. Sasaki, and Y. Nakagome. Trends in low-power RAM circuit technologies. In *Proceedings of the IEEE*, pages 84–87, 1995.

[86] Ravindra Jejurikar. Optimized slowdown in real-time task systems. In *Proceedings of the 16th Euromicro Conference on Real-Time Systems (ECRTS04)*, pages 1588–1598, July 2004.

[87] W. Kim, D. Shin, H. Yun, J. Kim, and S. Min. Performance comparison of dynamic voltage scaling algorithms for hard real-time systems. In *Proceedings of the Symposium on Real-Time and Embedded Technology and Applications*, 2002.

[88] J.G. Koomey. Estimating total power consumption by servers in the U.S. and the world. Technical report, Lawrence Berkeley National Laboratory, February 2007.

[89] R. Kotla, A. Devgan, S. Ghiasi, T. Keller, and F. Rawson. Characterizing the impact of different memory-intensity levels. In *Proceedings of the 7th IEEE International Workshop on Workload Characterisation (WWC04)*, pages 3–10, 2004.

[90] Amit Kumar, Li Shang, Li-Shiuan Peh, and Niraj K. Jha. HybDTM: a coordinated hardware-software approach for dynamic thermal management. In *Proceedings of the 43rd ACM/IEEE Conference on Design, Automation and Test in Europe*, pages 548–553, San Francisco, CA, July 2006.

[91] Martin P. Lawitzky, David C. Snowdon, and Stefan M. Petters. Integrating real time and power management in a real system. In *Proceedings of the 4th Workshop on*

Operating System Platforms for Embedded Real-Time Applications, Prague, Czech Republic, July 2008.

[92] Alvin Lebeck, Xiaobo Fan, Heng Zeng, and Carla Ellis. Power aware page allocation. In *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS IX)*, 2000.

[93] Kester Li, Roger Kumpf, Paul Horton, and Thomas E. Anderson. A quantitative analysis of disk drive power management in portable computers. In *Proceedings of USENIX Winter*, 1994.

[94] Jacob Lorch. A complete picture of the energy consumption of a portable computer. Technical report, University of California at Berkeley, 1995.

[95] Jacob R. Lorch and Alan Jay Smith. Improving dynamic voltage scaling algorithms with PACE. In *Proceedings of the ACM SIGMETRICS Conference*, pages 50–61, 2001.

[96] Yung-Hsiang Lu, Luca Benini, and Giovanni De Micheli. Operating-system directed power reduction. In *Proceedings of the 2000 International Symposium on Low Power Electronics and Design*, pages 37–42. Stanford University, 2000.

[97] A. Mallik, B. Lin, G. Memik, P. Dinda, and R.P. Dick. User-driven frequency scaling. *IEEE COMPUTER ARCHITECTURE LETTERS*, 5(2):61, 2006.

[98] Thomas L. Martin. *Balancing Batteries, Power, and Performance: System Issues in CPU Speed-Setting for Mobile Computing*. PhD thesis, Carnegie Mellon University, Pittsburgh, Pennsylvania, 2001.

[99] Thomas L. Martin. *Balancing Batteries, Power, and Performance: System Issues in CPU Speed-Setting for Mobile Computing*. PhD thesis, Carnegie Melon University, 2001.

[100] Thomas L. Martin and Daniel P. Siewiorek. Nonideal battery and main memory effects on cpu speed-setting for low power. *IEEE Transactions on Very Large Scale Integration Systems*, 9(1):29–34, February 2001.

[101] R. McGowen, C.A. Poirier, C. Bostak, J. Ignowski, M. Millican, W.H. Parks, and S. Naffziger. Power and temperature control on a 90-nm Itanium family processor. *IEEE Journal of Solid-State Circuits*, 41(1):229–237, Jan. 2006.

[102] Andreas Merkel and Frank Bellosa. Balancing power consumption in multiprocessor systems. In *Proceedings of the First ACM SIGOPS EuroSys Conference*, Leuven, Belgium, April 18–21 2006.

[103] Microchip Technology Inc. *MCP3909: Energy Metering IC with SPI Interface and Active Power Pulse Output*, March 2009.

[104] Micron Technology, Inc. Micron web site. URL http://www.micron.com, 2009.

[105] R. Min, M. Bhardwaj, S. Cho, A. Sinha, E. Shih, A. Wang, and A. Chandrakasan. Low-power wireless sensor networks, 2001.

[106] R. Min, T. Furrer, and A. Chandrakasan. Dynamic voltage scaling techniques for distributed microsensor networks. In *Proceedings of the IEEE Workshop on Very Large Scale Integration*, 2000.

[107] Akihiko Miyoshi, Charles Lefurgy, Eric Van Hensbergen, Ram Rajamony, and Raj Rajkumar. Critical power slope: understanding the runtime effects of frequency scaling. In *Proceedings of the 16th International Conference on Supercomputing*, pages 35–44, New York, NY, USA, June 2002. ACM Press.

[108] P. Nagpurkar. *Analysis, Detection, and Exploitation of Phase Behavior in Java Programs*. PhD thesis, UNIVERSITY OF CALIFORNIA, 2007.

[109] Dushyanth Narayanan, Jason Flinn, and M. Satyanarayanan. Using history to improve mobile application adaptation. In *Proceedings of the 3rd IEEE Workshop on Mobile Computing Systems and Applications*, 2000.

[110] Rolf Neugebauer and Derek McAuley. Energy is just another resource: Energy accounting and energy pricing in the Nemesis OS. In *Proceedings of the 8th IEEE Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, 2001.

[111] Venkatesh Pallipadi, Shaohua Li, , and Adam Belay. cpuidle — do nothing, efficiently... In *Proceedings of the 2007 Ottawa Linux Symposium*, 2007.

[112] Venkatesh Pallipadi and Alexey Starikovskiy. The ondemand governor. In *Proceedings of the Ottawa Linux Symposium*, volume 2, pages 223–238, 2006.

[113] Jorgen Peddersen and Sri Parameswaran. CLIPPER: Counter-based low impact processor power estimation at run-time. In *Proceedings of the 2007 Asia and South*

Pacific Design Automation Conference (ASPDAC07), pages 890–895, Washington, DC, USA, 2007. IEEE Computer Society.

[114] Jorgen Peddersen and Sri Parameswaran. Energy driven application self adaptation at run-time. In *Proceedings of the 20th International Conference on VLSI Design*, pages 385–390, January 2007.

[115] Paul I. Pénzes and Alain J. Martin. Energy-delay efficiency of VLSI computations. In *Proceedings of the 12th ACM Great Lakes symposium on VLSI (GLSVLSI02)*, pages 104–111, New York, NY, USA, 2002. ACM.

[116] Colin Percival. Enhanced speedstep driver for freebsd. URL http://www.daemonology.net/freebsd-est/, 2009.

[117] Trevor Pering, Thomas Burd, and Robert Brodersen. Voltage scheduling in the lpARM microprocessor system. In *Proceedings of the International Symposium on Low Power Electronics and Design*, 2000.

[118] Trevor Pering, Tom Burd, and Robert Brodersen. The simulation and evaluation of dynamic voltage scaling algorithms. In *Proceedings of the International Symposium on Low Power electronics and Design*, pages 76–81, 1998.

[119] Phytec America LLC. Phytec America website. URL http://www.phytec.com, 2009.

[120] Padmanabhan Pillai and Kang G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP01)*, pages 89–102, Lake Louise, Alta, Canada, October 2001.

[121] Christian Poellabauer, Leo Singleton, and Karsten Schwan. Feedback-based dynamic voltage and frequency scaling for memory-bound real-time applications. In *Proceedings of the 11th IEEE Real-Time and Embedded Technology and Applications Symposium*, volume 00, pages 234–243, 2005.

[122] J. Pouwelse, K. Langendoen, and H. Sips. Dynamic voltage scaling on a low-power microprocessor. In *Proceedings of the 7th ACM International Conference on Mobile Computing and Networking (MOBICOM01)*, pages 251–259, Rome, Italy, 2001.

[123] M.D. Powell, A. Biswas, J.S. Emer, S.S. Mukherjee, B.R. Sheikh, and S. Yardi. CAMP: A technique to estimate per-structure power at run-time using a few simple parameters. In *Proceedings of the 15th IEEE International Symposium on High Performance Computer Architecture (HCPA09)*, pages 289–300, February 2009.

[124] Voodoo Power. voodoo-power: OSX alternative power management. URL http://code.google.com/p/voodoo-power/, 2009.

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

[126] Dinesh Rajan, Russell Zuck, and Christian Poellabauer. Workload-aware dual-speed dynamic voltage scaling. In *Proceedings of the 12th IEEE Conference on Embedded and Real-Time Computing and Applications*, pages 251–256, 2006.

[127] Y. Sazeides, R. Kumar, D.M. Tullsen, and T. Constantinou. The danger of interval-based power efficiency metrics: When worst is best. *IEEE Computer Architecture Letters*, 4(1), 2005.

[128] Jed Scaramella. Worldwide server power and cooling expense: 2006-2010 forecast. White paper 203598, IDC, September 2006. http://www.sun.com/service/eco/IDCWorldwideServerPowerConsumption.pdf.

[129] Seiko Epson Corporation. *S1F81100 Series: System power supply IC*, March 2005.

[130] Semtech Corporation. *SC1476: Portable IMVP-IV Dual Phase Power Supply Controller Datasheet*, December 2002.

[131] D. Sengupta and R. Saleh. Generalized power-delay metrics in deep submicron CMOS designs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26(1):183–189, Jan. 2007.

[132] Kiran Seth, Aravindh Anantaraman, Frank Mueller, and Eric Rotenberg. FAST: Frequency-aware static timing analysis. *ACM Transactions on Embedded Computing Systems*, 5(1):200–224, 2006.

[133] Timothy Sherwood and Brad Calder. Time varying behaviour of programs. Technical Report UCSD-CS99-630, University of California at San Diego, August 1999.

[134] Timothy Sherwood, Erez Perelman, and Brad Calder. Basic block distribution analysis to find periodic behavior and simulation points in applications. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, September 2001.

[135] Dongkun Shin, Jihong Kim, and Seongsoo Lee. Low-energy intra-task voltage scheduling using static timing analysis. In *Proceedings of Design Automation Conference*, pages 438–443, 2001.

[136] T. Simunic, H. Vikalo, P. Glynn, and G. De Micheli. Energy efficient design of portable wireless systems. In *Proceedings of the 2000 International Symposium on Low Power Electronics and Design (ISLPED'00)*, pages 49–54, 2000.

[137] Tajana Simunic, Luca Benini, Andrea Acquaviva, Peter W. Glynn, and Giovanni De Micheli. Dynamic voltage scaling and power management for portable systems. In *Proceedings of the Design Automation Conference*, pages 524–529, 2001.

[138] Tajana Simunic, Luca Benini, Peter Glynn, and Giovanni De Micheli. Dynamic power management of a laptop hard disk. In *Proceedings of IEEE conference on Design, Automation and Test in Europe (DATE)*, pages 11–19, 2000.

[139] Tajana Simunic, Luca Benini, and Giovanni De Micheli. Cycle-accurate simulation of energy consumption in embedded systems. In *Proceedings of the Design Automation Conference*, pages 867–872, 1999.

[140] Tajana Simunic, Luca Benini, and Giovanni De Micheli. Energy-efficient design of battery-powered embedded systems. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 212–217, 1999.

[141] Tajana Simunic, Giovanni De Micheli, Luca Benini, and Mat Hans. Source code optimization and profiling of energy consumption in embedded systems. In *Proceedings of the IEEE International Symposium on System Synthesis*, pages 193–199, 2000.

[142] Karan Singh, Major Bhadauria, and Sally A. McKee. Real time power estimation and thread scheduling via performance counters. *SIGARCH Computer Architecture News*, 37(2):46–55, 2009.

[143] Amit Sinha and Anantha Chandrakasan. Jouletrack — a web based tool for software energy profiling. In *Proceedings of the Design Automation Conference*, pages 220–225, 2001.

[144] David C. Snowdon, Etienne Le Sueur, Stefan M. Petters, and Gernot Heiser. Koala: A platform for OS-level power management. In *Proceedings of the 4th EuroSys Conference*, Nuremberg, Germany, April 2009.

[145] David C. Snowdon, Stefan M. Petters, and Gernot Heiser. Power measurement as the basis for power management. In *Proceedings of the 1st Workshop on Operating System Platforms for Embedded Real-Time Applications*, Palma, Mallorca, Spain, July 2005.

[146] David C. Snowdon, Stefan M. Petters, and Gernot Heiser. Accurate on-line prediction of processor and memory energy usage under voltage scaling. In *Proceedings of the 7th International Conference on Embedded Software*, pages 84–93, Salzburg, Austria, October 2007.

[147] David C. Snowdon, Sergio Ruocco, and Gernot Heiser. Power management and dynamic voltage scaling: Myths and facts. In *Proceedings of the 2005 Workshop on Power Aware Real-time Computing*, New Jersey, USA, September 2005.

[148] David C. Snowdon, Godfrey van der Linden, Stefan M. Petters, and Gernot Heiser. Accurate run-time prediction of performance degradation under frequency scaling. In *Proceedings of the 3rd Workshop on Operating System Platforms for Embedded Real-Time Applications*, Pisa, Italy, July 2007.

[149] Elmar Stahleder. Optimization of energy consumption. Whitepaper, Lauterbach GmbH, June 2007.

[150] Jan Stoess, Christian Lang, and Frank Bellosa. Energy management for hypervisor-based virtual machines. In *Proceedings of the 2007 USENIX Technical Conference*, Santa Clara, CA, June 2007.

[151] Jon Stokes. Power gating and turbo mode: Intel talks nehalem at IDF. URL http://arstechnica.com/hardware/news/2008/08/power-gating-and-turbo-mode-intel-talks-nehalem-at-idf.ars, August 2008.

[152] Tai Kee Tan, Anand Rahunathan, Ganesh Lakshminarayana, and Niraj K. Jha. High-level energy macromodeling of embedded software. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(9):1037–1050, September 2002.

[153] Texas Instruments Incorporated. *MSP430x13x, MSP430x14x, MSP430x14x1 Mixed Signal Microcontroller*, July 2004.

[154] Texas Instruments Incorporated. *TPS65021: Power management IC for Li-Ion powered systems*, July 2007.

[155] O.S. Unsal, J.W. Tschanz, K. Bowman, V. De, X. Vera, A. Gonzalez, and O. Ergin. Impact of parameter variations on circuits and microarchitecture. *ACM/IEE International Symposium on Microarchitecture*, 26(6):30–39, Nov.-Dec. 2006.

[156] Energy Star Program U.S. Environmental Protection Agency. Report to congress on server and data center energy efficiency public law 109-431. Technical Report Public Law 109-431, U.S. Environmental Protection Agency, August 2007.

[157] Amin Vahdat, Alvin Lebeck, and Carla Ellis. Every joule is previous: The case for revisiting operating system design for energy efficiency. In *Proceedings of the 9th ACM SIGOPS European Workshop*, 2000.

[158] Prashant Vaibhav. xnu-speedstep: A kernel extension for OS X to enable Intel speedstep and undervolting on any kernel. URL http://code.google.com/p/xnu-speedstep/, 2009.

[159] A. Varma, B. Ganesh, M. Sen, S. R. Choudhary, L. Srinivasan, and B. Jacob. A control-theoretic approach to dynamic voltage scaling. In *Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES)*. Dept. of Electrical & Computer Engineering, University of Maryland at College Park, October 2003.

[160] Suji Velupillai and Ken Tough. Intelligent energy manager (IEM) benchmarking on a Freescale's iMX31 multimedia processor. Technical report, Intrinsyc Software, September 2007.

[161] Vasanth Venkkatachalam, Chrisian Probst, and Michael Franz. A new way of estimating compute boundedness and its application to dynamic voltage scaling. *International Journal on Embedded Systems*, 1(1):64–74, 2006.

[162] VMware, Inc. VMware helps enterprises and governments of all sizes go green. URL http://www.vmware.com/company/news/releases/greenit_09.html, April 2009.

[163] Martin Waitz. Accounting amd control of power consumption in energy-aware operating systems. Diploma thesis, Operating System Group, University of Erlangen, Germany, January 2003.

[164] Mark Weiser, Brent Welch, Alan J. Demers, and Scott Shenker. Scheduling for reduced CPU energy. In *Proceedings of the 1st USENIX Symposium on Operating Systems Design and Implementation (OSDI94)*, pages 13–23, Monterey, CA, USA, 1994.

[165] Andreas Weissel and Frank Bellosa. Process cruise control—event-driven clock scaling for dynamic power management. In *Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, Grenoble, France, October 8–11 2002.

[166] Andreas Weissel and Frank Bellosa. Process cruise control: event-driven clock scaling for dynamic power management. In *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES02)*, Grenoble, France, October 2002.

[167] Andreas Weissel, Bjoern Beutel, and Frank Bellosa. Cooperative I/O: A novel I/O semantics for energy-aware applications. In *Proceedings of the 5th Symposium on Operating System Design and Implementation (OSDI02)*, 2002.

[168] FreeBSD Wiki. FreeBSD powerd website. URL http://wiki.freebsd.org/powerd.

[169] John Wilkes. Predictive power conservation. Technical Report HPL-CSP-92-5, Hewlett Packard Laboratories, 1992.

[170] Fen Xie, Margaret Martonosi, and Sharad Malik. Compile-time dynamic voltage scaling settings: Opportunities and limits. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 49–62, New York, NY, USA, 2003. ACM Press.

[171] Fen Xie, Margaret Martonosi, and Sharad Malik. Efficient behavior-driven runtime dynamic voltage scaling policies. In *Proceedings of the 3rd International Conference on Hardware/Software Codesign and System Synthesis*, pages 105–110, 2005.

[172] Yisehac Yohannes and John Hoddinott. Classification and regression trees: An introduction. Whitepaper, International Food Policy Research Institute, March 1999.

[173] Wanghong Yuan and Klara Nahrstedt. Energy-efficient soft real-time CPU scheduling for mobile multimedia systems. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP03)*, pages 149–163, Urbana, IL, 2003. University of Illinois at Urbana-Champaign, ACM Press.

[174] Hang Zeng and Carla S. Ellis Alivn R. Lebeck. Experiences in Managing Energy with ECOSystem. *IEEE Pervasive Computing*, 4(1):62–68, 2005.

[175] Heng Zeng, Carla S. Ellis, Alvin R. Lebeck, and Amin Vahdat. Currentcy: Unifying policies for resource management. In *Proceedings of the USENIX 2003 Annual Technical Conference*, San Antonio, Texas, June 2003.

[176] Heng Zeng, Xiaobo Fan, Carla Ellis, Alvin Lebeck, and Amin Vahdat. ECOSystem: Managing energy as a first class operating system resource. In *Tenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS X)*, 2002.

[177] Y. Zhang, D. Parikh, K. Sankaranarayanan, K. Skadron, and M. Stan. Hotleakage: A temperature-aware model of subthreshold and gate leakage for architects. Technical report, University of Virginia Department of Computer Science, March 2003.

APPENDIX A

PLATFORMS

A variety of platforms were evaluated during the course of this thesis. Some of the platforms were used more than others because fine-grained frequency scaling, such as that implemented by Koala, is by nature more effective, and therefore more interesting in the context of this thesis, on some platforms than others. All of the platforms which were examined are described here, including those that were given cursory inspection.

A.1 PLEB 2 (PXA255) — *PLEB 2*

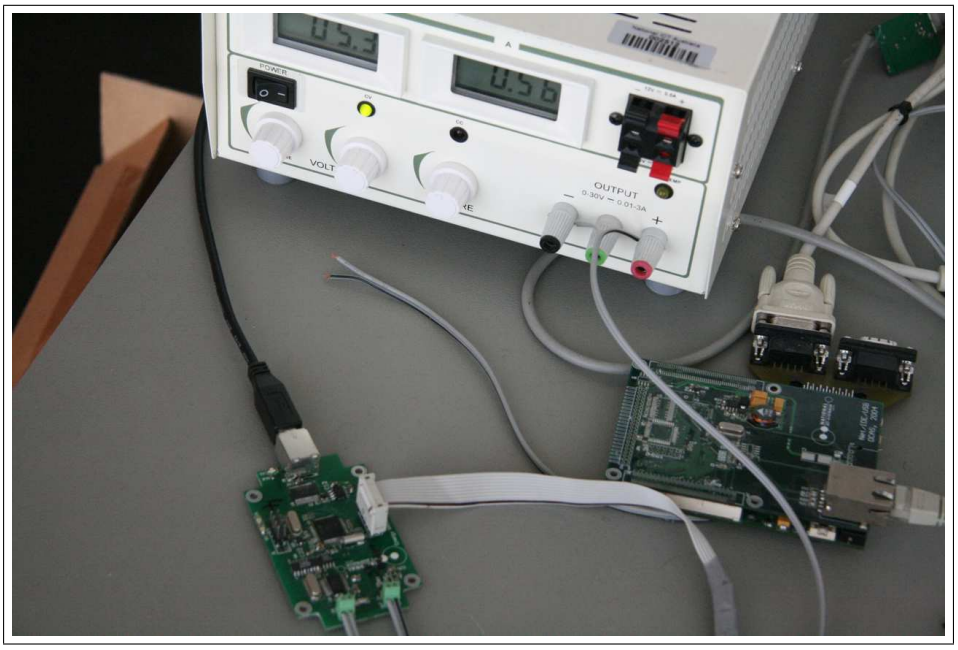


Figure A.1: PLEB 2 with Echidna

Processor	PXA255
Architecture	ARMv5TE
Tested Settings	22
Variable Frequencies	3
Variable Voltages	1
Energy Measurement	Echidna
Performance Counters	2 + cycle counter
Performance Events	14

PLEB 2 was designed for the purposes of experimentation during this thesis. It is a single-board computer, consisting of a PXA255 processor [74] which, as specified, can be clocked at up to 398MHz (but was over-clocked to 471MHz for the purposes of collecting a larger range of data). It has 64MB SDRAM and 8MB flash. The PXA255 is based on an ARMv5T-compatible XScale core with a 7-stage integer and 8-stage memory pipeline. It has split L1 caches and TLBs, write, fill and pend buffers. The data cache supports both write-through and write-back policies.

The system includes a LAN91C111 network interface IC, which was used to transfer benchmark software and results. The interface was disabled during all measurements to avoid interrupts and active power. There are several other components on-board, such as LEDs and resistors. Combined with the CPU, memory and flash, these contribute to a sizeable static power.

Various voltage supplies are generated by an S1F81100 power-management chip from a constant 4.1V source. It consists of three buck converters which switch to the CPU core, memory and IO interface voltages. The maximum efficiency for the core voltage is quoted at 80%, with 90% for the memory and IO interface voltage supplies. The S1F81100 technical manual [129] presents graphs showing an approximately constant efficiency for the loads in PLEB 2. This chip can be controlled by the PXA255 via an I2C bus, allowing for adjustment of the processor's core voltage. All other circuits run at a fixed 3.3V.

A number of frequencies are generated by the PXA255's clock management unit. These include the core clock (f_{cpu}), system bus clock (f_{bus}), SDRAM clock (f_{mem}), peripheral bus clock (f_{io}), real-time clock (f_{rtc}), etc. Because of the way these clocks are synthesised, only certain combinations can be generated. We call each of these clock combinations a *setpoint*. Typical of a real system, only f_{cpu}, f_{bus} and f_{mem} are varied in these experiments.

All possible combinations of f_{cpu}, f_{bus} and f_{mem} were considered, including those outside the chip's specifications. The frequencies were limited to reasonable values: f_{cpu} varies between 99 and 471MHz, f_{bus} varies between 50 and 236MHz and f_{mem} varies between 99 and 133MHz. A total of 22 unique setpoints were used.

The electrical specifications for the PXA255 give the appropriate CPU core voltage for a number of frequency settings. Since this information does not specify the core voltage required for every possible frequency, a roughly linear relationship between frequency and voltage was found and used to calculate the voltage for the unspecified frequencies. Furthermore, the S1F81100 power supply chip only allows discrete steps of 0.1V. For each frequency the available voltage closest to a linear interpolation between frequency-voltage pairs specified in the processor's documentation was chosen and confirmed to operate correctly.

Switching frequencies on PLEB 2 is software sequenced, with a slow interface to the S1F81100 power supply IC. The I2C bus used can run at a maximum of 400kHz, the S1F81100 requires a 6-byte transfer to switch voltages, and the PXA255's I2C hardware provides no buffering so each byte's transmission causes an interrupt in order to load the next. In addition the frequency change overhead can be large — the CPU is unavailable

for $\sim 500\mu$s when the system's PLL re-locks. A faster frequency-change mechanism is available via the *Turbo mode* bit. This controls a divider which is configured during the PLL lock. The latency for this switch was measured at $\sim$20 cycles. Therefore to change between two pre-determined frequencies has a low overhead,but changing between arbitrary frequencies, or changing voltages, has a high overhead (in the context of a 4ms timer-tick period).

The PXA255 provides two performance counters and one cycle counter. The configurable counters can each be set to count one of 14 events, described in the PXA255 developer's manual [74].

All PLEB 2 experiments presented is this thesis were conducted in Linux 2.6.24 patched to support the platform. Kernel modules were written for benchmarking purposes: to support the performance-monitoring unit, voltage scaling via the power supply chip, triggering of energy measurements, and frequency scaling based on `cpufreq`.

Energy measurements were made via the Echidna board, with triggering via a GPIO pin on the PXA255. The Echidna measures the power used to supply the S1F81100 power supply IC at 4.1V. This was supplied by a voltage controlled laboratory power supply.

In addition to the above energy measurement mechanism, a small microcontroller is embedded on the motherboard, along with sense resistors and amplifiers for measuring the current drawn from the memory, IO and CPU core power supplies. This microcontroller can communicate with the PXA255 via an SPI bus.

Figures A.2, A.3 and A.4 show the normalised execution time, measured power and normalised energy for a memory-bound and a CPU-bound benchmark. In each of these plots connected points run at the same memory and bus frequency.

It is clear from Figure A.2 that memory and CPU intensive applications behave differently on this platform. The CPU-bound `twolf_test` scales as expected with the CPU frequency (reducing the frequency from 400 MHz to 100 MHz results in about four times the execution time), whereas `gzip_test` is much more consistent. The difference in the memory performance resulting from the changing memory and bus frequencies can clearly be seen in the plot of `gzip_test`.

The average power for each of those benchmarks can be seen in Figure A.3, with the power required to access memory resulting in a higher power draw for `gzip_test`.

More interesting in this case is the energy used as depicted in Figure A.4 (normalised to the energy at one of the 471 MHz settings. Counter-intuitively, the energy required for the CPU-bound benchmark is reduced as the frequency increases. This is because the run-time (see Figure A.2) is significantly reduced, and, for this graph, we assume that the

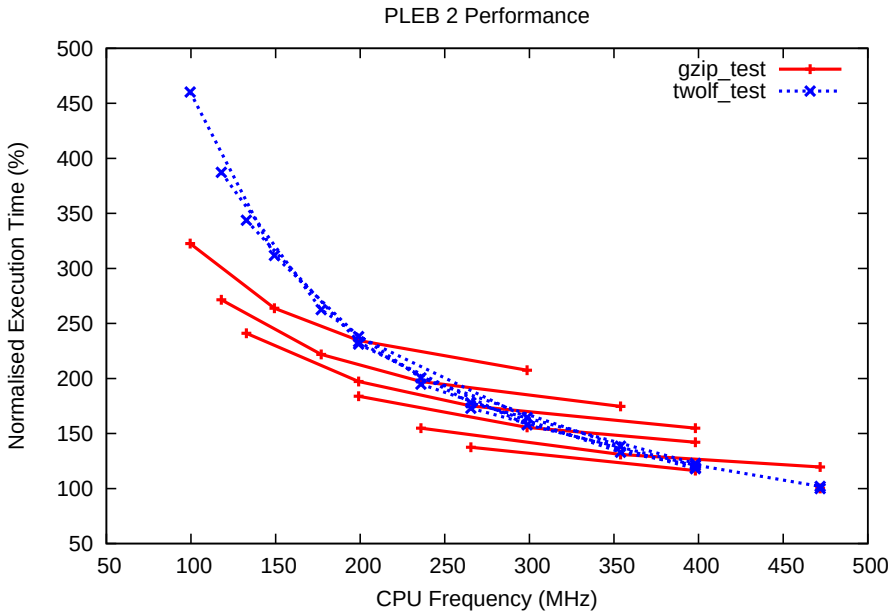


Figure A.2: Normalised execution time for memory-bound (`gzip_test`) and CPU-bound (`twolf_test`) benchmarks on PLEB 2.

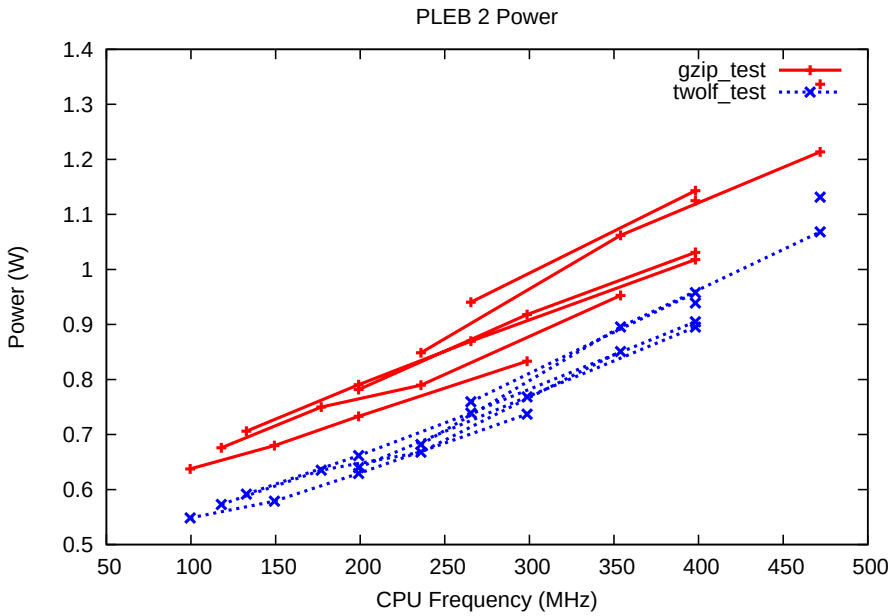


Figure A.3: Measured power for memory-bound (`gzip_test`) and CPU-bound (`twolf_test`) benchmarks on PLEB 2.

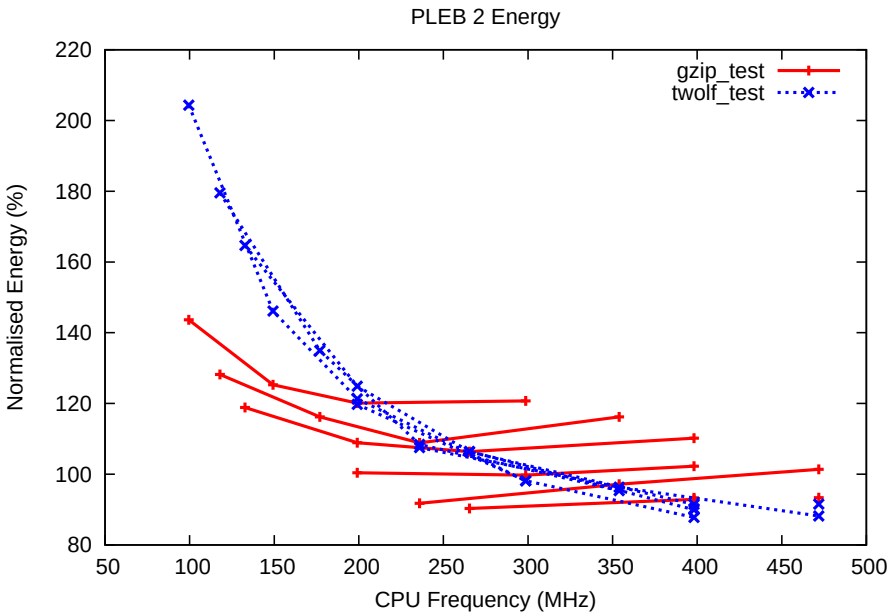


Figure A.4: Normalised energy for memory-bound (`gzip_test`) and CPU-bound (`twolf_test`) benchmarks on PLEB 2.

system is shut down following the benchmark run.

The bus and memory frequency make a difference to the energy used. At f_{cpu} =471 MHz the CPU-bound benchmark uses more energy with a higher bus and memory frequency (266 MHz and 133 MHz respectively). But, for the memory-bound benchmark, we see the opposite effect. Since the higher bus/memory frequency does not improve the system's performance for CPU-bound applications, it makes sense to use the lower bus/memory frequency settings for those workloads.

PLEB 2's idle power, running the benchmarking kernel with the network interface disabled, was measured at 0.533W. Figure A.5 shows the energy required to run the benchmark, once the power required to idle the system has been subtracted (see Section 4.5). The plot seems erratic, but this can be explained by the large minimum voltage step on PLEB 2 — 100mV. Figure A.6 shows the voltage which is used to run each frequency, which, unsurprisingly, looks very similar to the dynamic energy plot — according to the commonly-assumed model, energy is proportional to voltage.

While performance and power models were developed and tested for PLEB 2 (discussed in Section 7.5), Koala was not ported due to the prohibitively high overhead of switching frequencies on this platform.

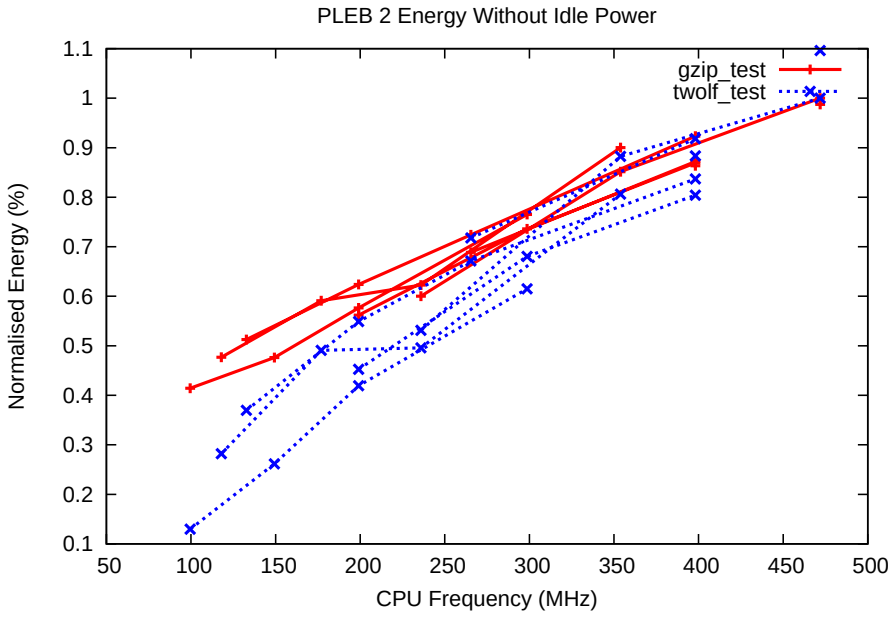


Figure A.5: Normalised energy without idle power for memory-bound (`gzip_test`) and CPU-bound (`twolf_test`) benchmarks on PLEB 2.

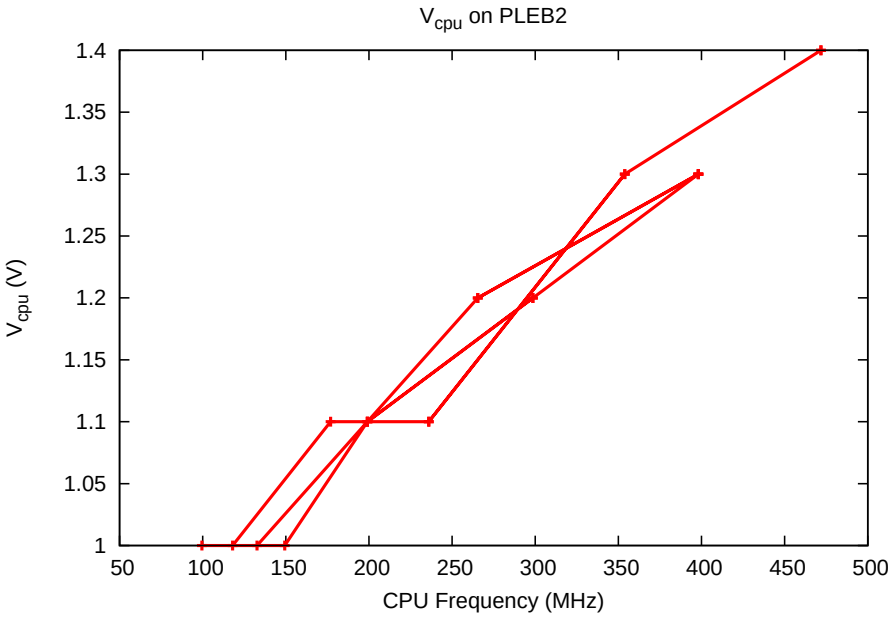


Figure A.6: Voltage setting vs. frequency for PLEB 2.

A.2 Gumstix Connex (PXA255) — *Gumstix*

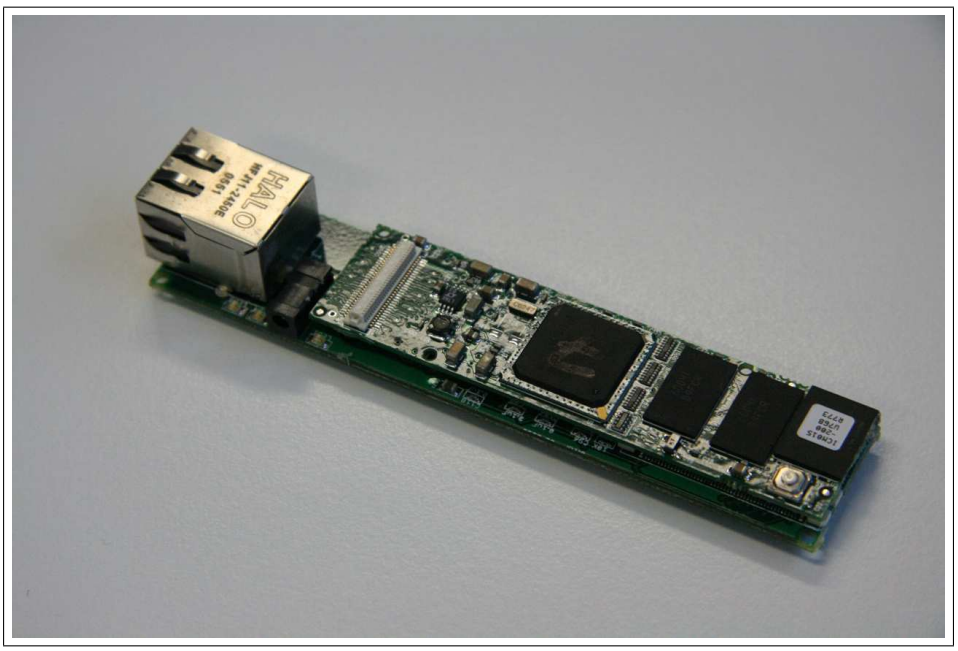


Figure A.7: Gumstix Connex with EtherStix network card

Processor	PXA255
Architecture	ARMv5TE
Tested Settings	22
Variable Frequencies	3
Variable Voltages	0
Energy Measurement	Echidna
Performance Counters	2 + cycle counter
Performance Events	14
Other notes	16-bit data bus

The Gumstix Connex platform [70] is very similar to PLEB 2 (Section A.1). It consists of the same PXA255 system-on-a-chip, with a similar arrangement of 64MB SDRAM and 4MB flash. In contrast to PLEB 2, which has a 32-bit data bus, the Gumstix has a 16-bit main memory data bus. This reduces the system's memory bandwidth.

The same set of 22 settings were tested, however the Gumstix uses a different power supply IC to PLEB 2. This power supply limits the CPU core voltage to a fixed 1.3V

and only frequency scaling, rather than voltage scaling, information is available for the Gumstix. The benchmarking, energy measurement methodology, kernel modifications, and other infrastructure, are shared with PLEB 2.

An Etherstix network interface board was used to provide network connectivity. It uses the same LAN91C111 network interface IC as the PLEB 2 NIC.

The Gumstix was supplied at 5V through an Echidna. 5V was supplied by a regulated supply.

This platform provides an example of a second system based on an identical processor to PLEB 2, and gives an idea of how platform design (as opposed to processor design) can affect the system power, and frequency scaling decisions. In the case of the Gumstix, the fixed CPU voltage and 16-bit data bus substantially modify the behaviour of a platform which is otherwise very similar to PLEB 2. Figures A.8, A.9 and A.10 show the behaviour of the system.

As for PLEB 2, an increased memory frequency decreases the total energy used for a memory-bound benchmark, and increases it for a CPU-bound benchmark. This is of particular note at the highest tested frequency, where an 8% energy saving can be made by using the highest memory frequency for `gzip_test`, but doing so for `twolf_test` would result in a 4.7% increase.

The effect of scaling the frequency alone (rather than in tandem with the voltage) is shown more clearly when a typical idle power (0.31W) is subtracted from the measured energy (see Section 4.5 for details). For a given memory frequency, the energy savings are nearly constant for the CPU-bound benchmark. Changing the memory frequency has an effect on the system's power, but not on the performance in this case.

When this typical idle power is subtracted from the measured energy (see Section 4.5 for details), we can see the effect of scaling frequency without voltage (since this platform doesn't support voltage scaling). This is shown in Figure A.11. The energy used to run the CPU-bound benchmark is clearly effected by the memory frequency, with increasing memory frequency leading to increased dynamic energy. This is not true for the memory-bound workload, since it runs more efficiently with a high memory frequency. For the CPU-bound benchmark, the energy is hardly effected by changes in CPU frequency (for the same memory frequency), whereas clear energy savings are possible by reducing the CPU frequency when running the memory-bound benchmark. This is consistent with the commonly assumed model, which suggests that the dynamic energy is independent of frequency when running at a constant voltage.

The idle power subtracted in this case is only the `idle` sleep state, rather than the

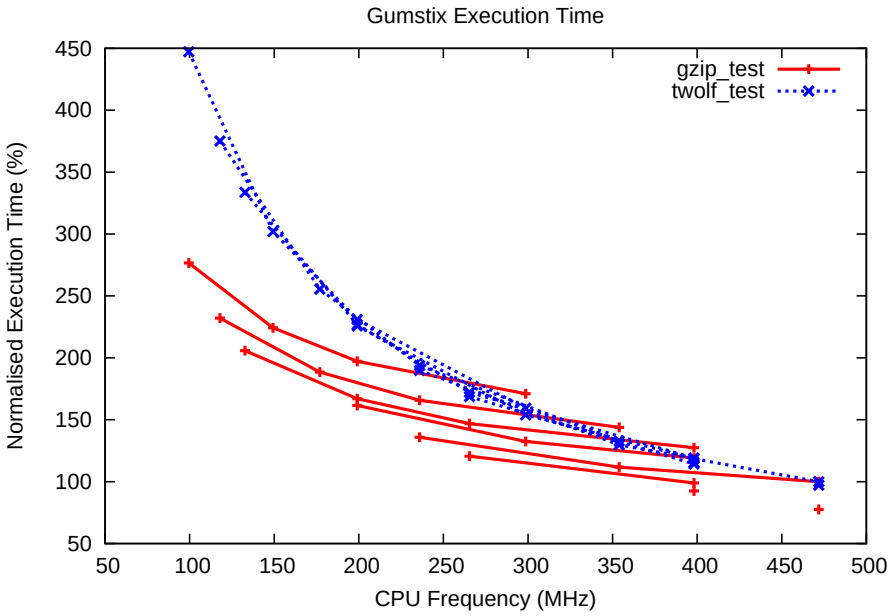


Figure A.8: Normalised execution time for memory-bound (`gzip_test`) and CPU-bound (`twolf_test`) benchmarks on Gumstix.

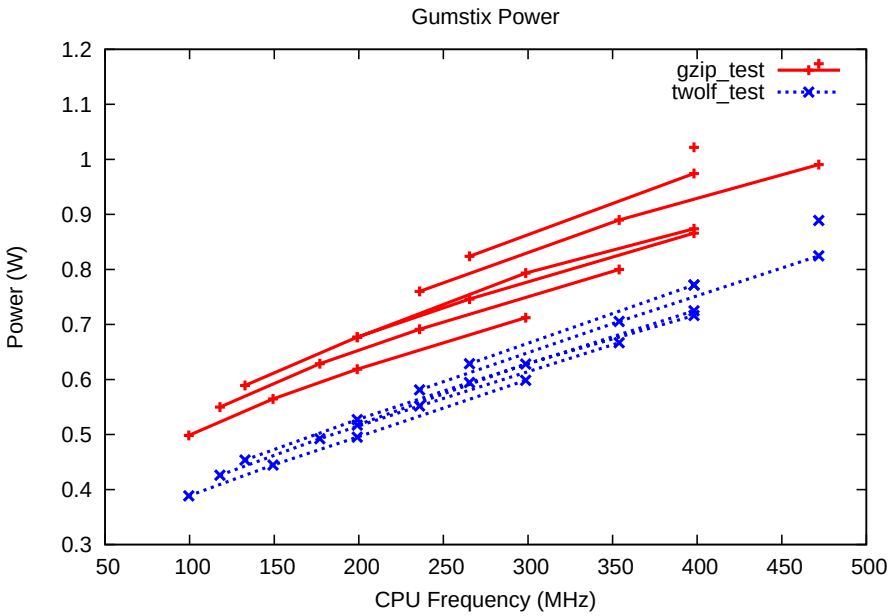


Figure A.9: Measured power for memory-bound (`gzip_test`) and CPU-bound (`twolf_test`) benchmarks on Gumstix.

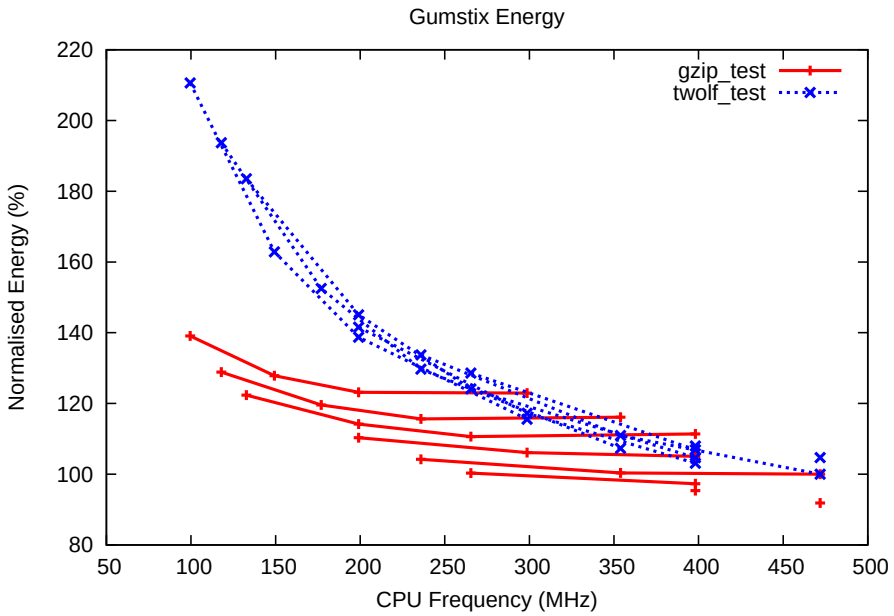


Figure A.10: Normalised Energy for memory-bound (`gzip_test`) and CPU-bound (`twolf_test`) benchmarks on Gumstix.

deeper `33MHz idle`, `sleep` or `deep sleep` states. Lower power sleep states would bias the results toward the higher frequencies (i.e. somewhere between Figure A.10 and Figure A.11).

Similarly to PLEB 2, performance and power models were developed and are discussed in Section 7.5. Koala was ported to the Gumstix, but the prohibitively high frequency and voltage switching overheads led to no further testing in the context of this thesis (although some work in combining Koala with real-time scheduling, taking into account switching overheads was conducted by a Lawitzky et al. [91]).

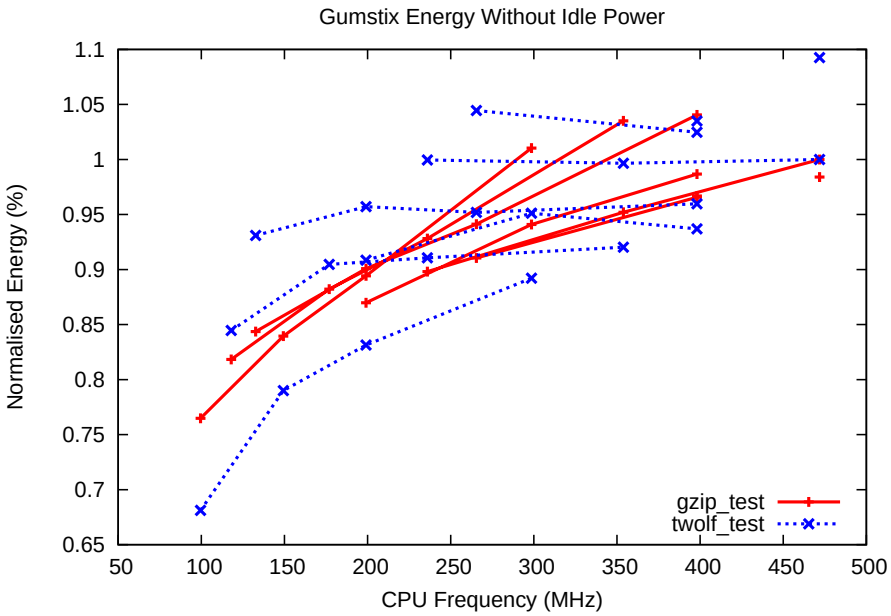


Figure A.11: Normalised Energy for memory-bound (`gzip_test`) and CPU-bound (`twolf_test`) benchmarks on Gumstix, after subtracting a typical idle power.

A.3 I-Box (PXA270) — *I-Box*

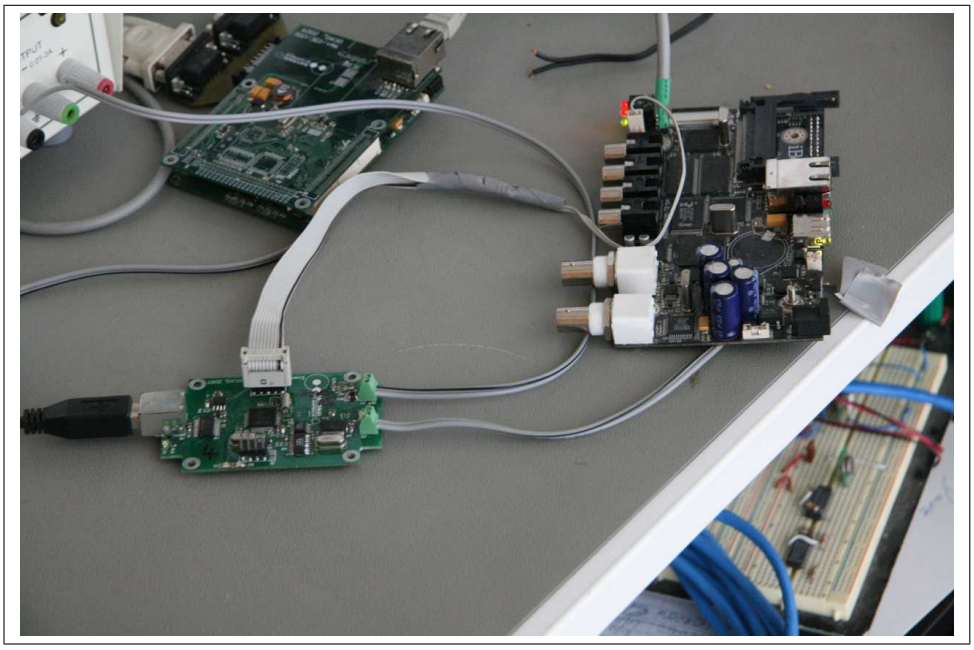


Figure A.12: I-Box with Echidna

Processor	PXA270
Architecture	ARMv5TE
Tested Settings	169
Variable Frequencies	3
Variable Voltages	1
Energy Measurement	Echidna
Performance Counters	4 + cycle counter
Performance Events	14

The I-Box was designed by the author as a side-project during the course of this thesis. It is a platform for the development of digital video surveillance applications. It is commercially licensed and therefore a good example of a typical real-world embedded system. Power was a key factor in the I-Box design, and the hardware involved is typical of many battery-powered embedded systems (although it is not specifically designed to run from a battery).

The I-Box hardware consists of a PXA270 processor [75] which can be clocked at up to 624MHz. It has 64MB SDRAM and 4MB flash. Various peripherals are integrated on-board, including a video decoder, MPEG encoder, video output IC, network interface, USB host, compact flash, LCD and audio I/O. Like the PXA255, the PXA270 is based on an XScale core.

Like PLEB 2 and the Gumstix, a LAN91C111 network IC is used for network connectivity, but was disabled during all measurements.

Various voltages are generated by a TPS65021 [154] power management chip. This chip interfaces with the PXA270 via an I2C bus, allowing for adjustment of the processor's core voltage. The memory bus and IO interfaces all run at a fixed 3.3V (although the PXA270 can handle voltage scaling of the memory bus). The TPS65021 is supplied with 5V by a second DC-DC converter. During energy measurements for this thesis, the TPS65021 was supplied with 5V directly via an Echidna. The Echidna was triggered using the I-Box's alarm interface (which is in turn controlled by a GPIO).

A number of frequencies are generated by the clock management unit within the PXA270, which is a more flexible version of the one in the PXA255. These include, among others, the core clock (f_{cpu}), system bus clock(f_{bus}), SDRAM clock(f_{mem}), peripheral bus clock(f_{io}), real-time clock(f_{rtc}). The first three were varied during experiments for this thesis.

Since the PXA270 documentation only specifies a limited number of frequency-voltage pairings, a quadratic model was derived from these (see Figure A.13) to calculate the voltage for a given frequency. The voltage setting is used as part of the system energy model.

All possible combinations of f_{cpu}, f_{bus} and f_{mem} were considered. The frequencies were limited to allowable values. Duplicate setpoints were removed for the purposes of modelling. f_{cpu} varies between 52 and 624MHz, f_{bus} varies between 52 and 246MHz and f_{mem} varies between 52 and 136.5MHz. A total of 169 unique setpoints were tested.

Like PLEB 2, the PXA processor in the I-Box has a high PLL re-lock overhead of $\sim$500μs, as well as a multi-cycle overhead for a *turbo mode* switch. The PXA270 adds an *half-turbo mode*, which changes the CPU frequency to a second pre-defined divider from the PLL-generated clock. Depending on the PLL and turbo settings, it is possible to obtain different combinations of three frequencies (run, half-turbo and turbo), and for the same frequency to be available in different subsets. The switching overhead therefore varies depending not only on the frequency, but on the PLL settings. The voltage scaling hardware was also improved in the I-Box with the addition of a dedicated I2C unit for

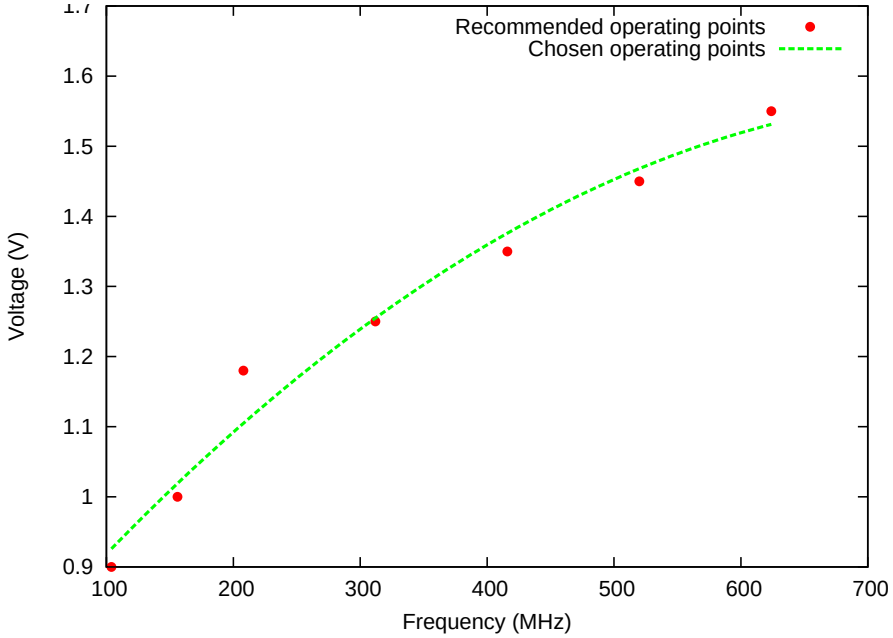


Figure A.13: Voltage vs. Frequency model for the PXA270

power supply communications, adding frequency and voltage change sequencing in hardware. Unfortunately, this unit runs at a slow 40kHz, and so the voltage change overhead is still very large.

The PXA270 has two more event counters than the PXA255, along with the cycle counter. The configurable counters can each be configured to count one of the same 14 events as the PXA255 [75].

All experiments were conducted under Linux 2.6.24. Much of the same benchmarking infrastructure as was used for PLEB 2 and the Gumstix was used in I-Box experimentation. Drivers were developed for frequency scaling and the TPS65021 power supply IC.

The characteristics of the I-Box are, unsurprisingly, similar to PLEB 2 (albeit with more available frequency setpoints) and are shown in Figures A.14, A.15 and A.16. CPU-bound benchmarks scale with the CPU frequency, and are unaffected by changes to the memory or bus frequency.

Similarly to PLEB 2, performance and power models were developed and are discussed in Section 7.5. Koala was ported to the I-Box, but the prohibitively high frequency and voltage switching overheads led to no further testing.

The I-Box idle power is dominated by peripherals rather than the processor power. In addition, the idle modes have not been fine-tuned. As a result, the idle power is nearly

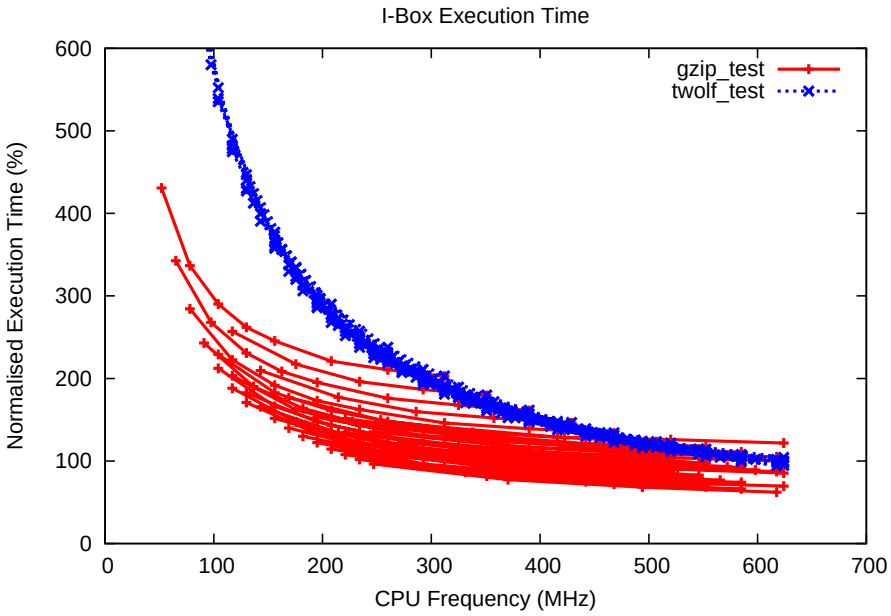


Figure A.14: Normalised execution time for memory-bound (`gzip_test`) and CPU-bound (`twolf_test`) benchmarks on I-Box.

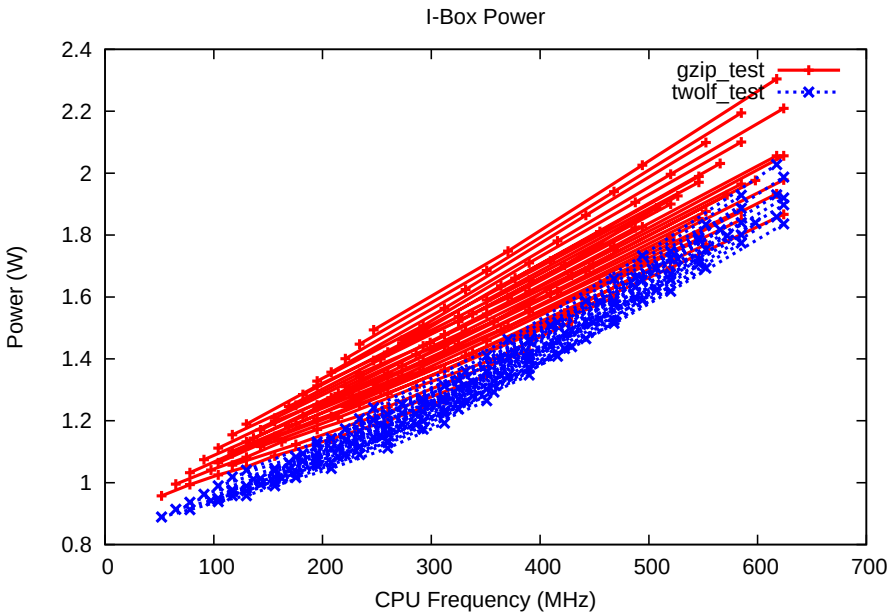


Figure A.15: Measured power for memory-bound (`gzip_test`) and CPU-bound (`twolf_test`) benchmarks on I-Box.

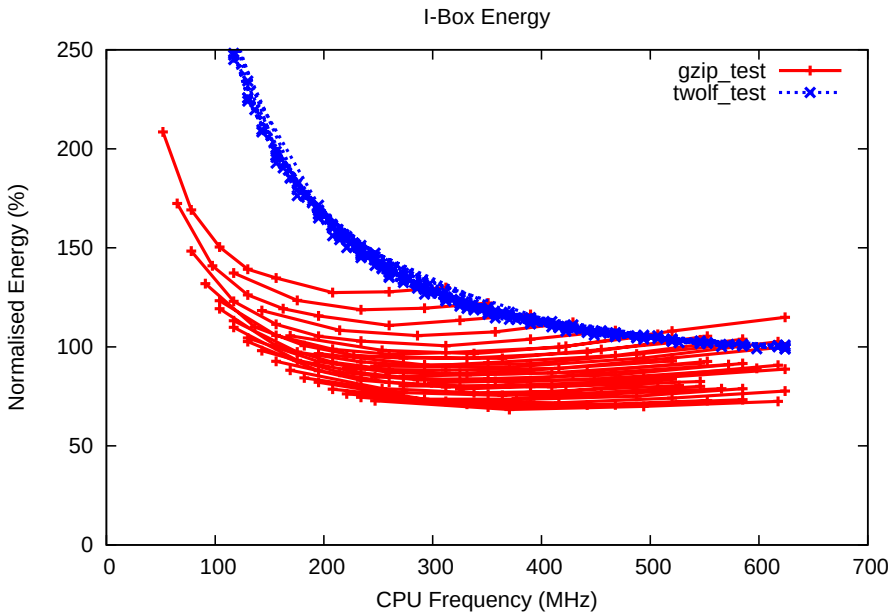


Figure A.16: Normalised Energy for memory-bound (`gzip_test`) and CPU-bound (`twolf_test`) benchmarks on I-Box.

indistinguishable from the active power running a CPU-bound benchmark at the lowest frequency setting. This is shown in Figure A.17, where the lowest-frequency setting uses 0% of the dynamic energy for the highest performance setting. The idle power in this situation is 0.89W.

If the system were able to disable some of the peripherals while idle, disable memory, or improve the idle power by only $\sim$0.2W, then the DVFS optimisation becomes much more interesting. Figure A.18 shows this scenario, where clear and dissimilar optima are present for both the memory-bound and CPU-bound workload.

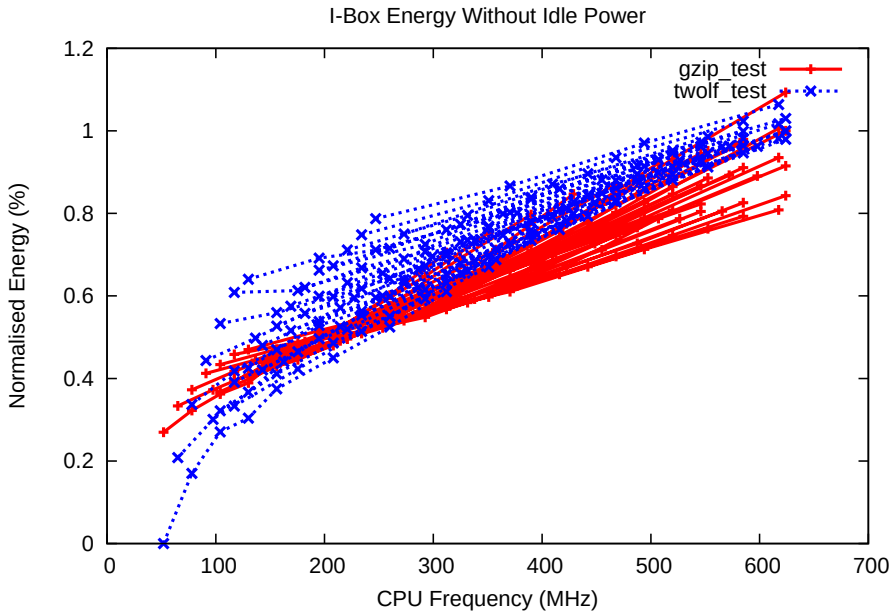


Figure A.17: Normalised Energy without idle power for memory-bound (`gzip_test`) and CPU-bound (`twolf_test`) benchmarks on I-Box.

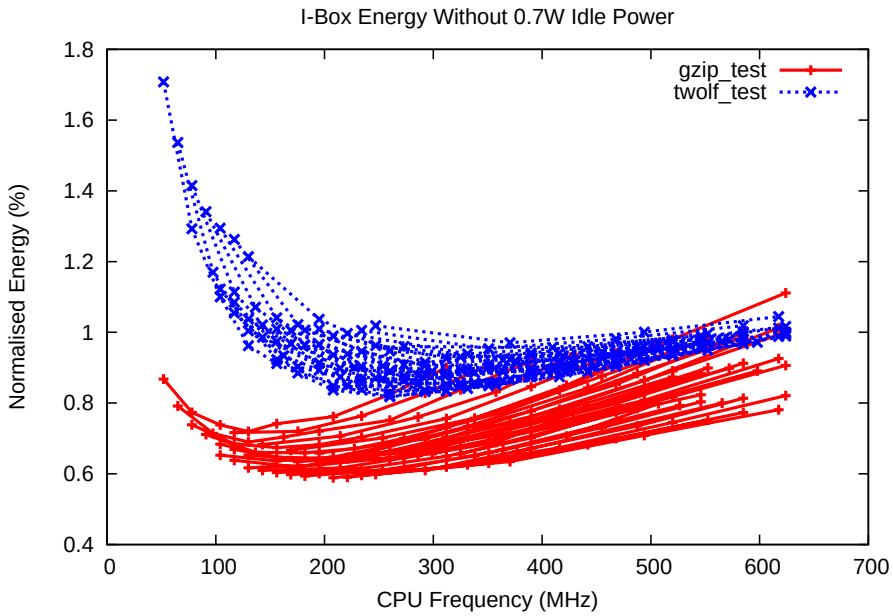


Figure A.18: Normalised Energy, assuming an 0.7W idle power, for memory-bound (`gzip_test`) and CPU-bound (`twolf_test`) benchmarks on I-Box.

A.4 phyCORE-iMX31 Rapid Development Kit (iMX31) — *Phycore*

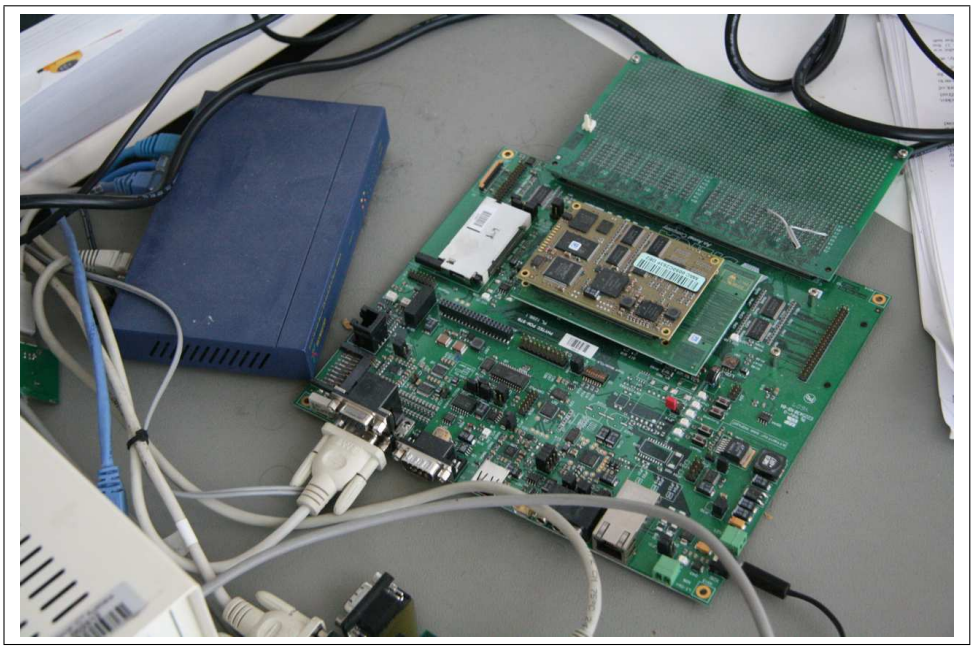


Figure A.19: phyCORE-iMX31 Rapid Development Kit

Processor	iMX31 (ARM1136JF-S)
Architecture	ARMv6
Tested Settings	4 [1]
Variable Frequencies	1 [2]
Variable Voltages	1
Energy Measurement	Echidna
Performance Counters	6 + cycle counter [3]
Performance Events	35 [3]

The phyCORE-i.MX31 Rapid Development Kit is a computer-on-module system provided by Phytec America [119]. It is based on a Freescale iMX31 system-on-chip processor [50]. The iMX31 is based on the ARM1136JF-S processor core, which implements the ARMv6 architecture. It has an 8-stage integer pipeline with separate load-store and arithmetic pipelines. The iMX31 in the Phycore board has a core frequency of up to 532MHz.

The iMX31 itself provides a number of integrated peripherals including a vector floating point unit, image processing unit, LCD controller, etc. The Phytec rapid development kit provides a rich set of peripherals including 10/100 MBit Ethernet, USB2.0 host, CAN, Audio, etc.

The Phycore is supplied at 5.0V. Power and energy were measured using an Echidna board triggered via a GPIO from the iMX31.

Two different performance monitoring units are available in the iMX31. The first is the in-core *performance metrics unit* (PMU). This resides in the ARM1136 core itself, acting as an ARM co-processor. It has two counters and a cycle counter. Each of the counters can monitor one of 20 events occurring within the core, including the number of instructions executed, data cache accesses, data cache misses, etc. The second performance monitoring unit, EVTMON, is external to the core. It is a 32-bit IP-bus peripheral used for monitoring *Level 2 Cache Controller* (L2CC) events [50], of which there are 15 which relate to the level 2 cache and include the number of read requests, cache hits, evictions, etc. There are four counters.

The iMX31 differed from all other processors examined in that it has a variable frequency bus between the level 2 cache and the core itself. While processors such as the PXA255 and PXA270 had variable bus and memory frequencies, the bus operated between the core and the memory controller with no level 2 cache, meaning the latency on each memory access is affected consistently by scaled memory and bus frequencies. The arrangement in the iMX31 means that the average latency of a memory instruction is dependent on the probability of residency in the L1 cache, the L2 cache bus frequency, probability of residency in the L2 cache, and then the main memory frequency. All of these are variable and must be taken into account in a reasonable performance model.

DVFS is supported very well by the iMX31, with significant hardware support. *Dynamic Process and Temperature Compensation* (DPTC) provides hardware adaptation of the CPU voltage to compensate for both process and thermal variation. The system times the delay through reference circuits in order to determine whether the core voltage is too high or low, increasing or decreasing the voltage when a delay less than the lower lower or greater than the upper limit (respectively) is observed. Another interesting feature is a hardware DVFS mechanism based on event counting. Events from all over the system are monitored, weighted and the weighted values aggregated. The events are largely associated with the system's multi-master memory interface, but include interrupt events. DVFS decisions can be automatically made by the hardware with a mechanism for calling an interrupt when a programmable threshold is exceeded.

Further DVFS support is seen in the form of external pins which can control the system voltage (which is how DPTC can interface with the power management IC). A virtually limitless range of settings is available by using the three on-chip PLLs to generate frequencies. The platform uses an MC13783 integrated power management IC which has voltage scaling support via a fast SPI interface. In addition the Freescale parts support a direct connection for increasing, decreasing or maximising the voltage quickly.

The way in which these features could be utilised in the context of this thesis is left to future work. The voltage chosen by DPTC could be used as a parameter to Koala's models. The load tracking count could be used as an input to Koala models, and the threshold-based interrupt generation might be used for workload prediction. In this way, the in-built DVFS hardware compliments the Koala approach.

This platform was intended to be representative of a cutting-edge ARM-based device with a rich set of DVFS hardware. The iMX31 has been used extensively in recent mobile smartphones and is an excellent example of a modern high-performance embedded processor. Unfortunately some difficulty was had implementing frequency and voltage switches reliably, and so the experimentation which was possible was limited.

The characteristics of the Phycore development platform seem to indicate that it is a relatively un-interesting platform for DVFS work. This is because the platform's memory runs at a high speed in comparison to the CPU frequency, and so the memory latency is smaller compared to other platforms. The platform's behaviour is shown in Figures A.20, A.21 and A.22.

Figure A.21 shows the power being used — the memory-bound benchmark saves less power than the CPU-bound benchmark from DVFS because less power is expended in the CPU. The reverse is true at higher frequencies, where the higher CPU power for the CPU-bound benchmark is exacerbated by the higher frequency and clock rate. The platform also has a relatively high static power in comparison to the power which can be saved via clock scaling. This is because the system is a development platform with many on-board peripherals, LEDs, and other power-consuming devices that would likely be omitted in a real mobile system.

A typical idle power for the entire platform was measured at 2.45W. The majority of this can be assumed to be associated with the platform rather than the processor. The effect of subtracting this (as in Section 4.5) is shown in Figure A.23. It is clear that real savings can be made when idle time is generated via frequency scaling. Also note that for all benchmarks there is an energy-optimal point which is neither the minimum nor maximum. It is also not the same for each benchmark – as expected, the memory-bound

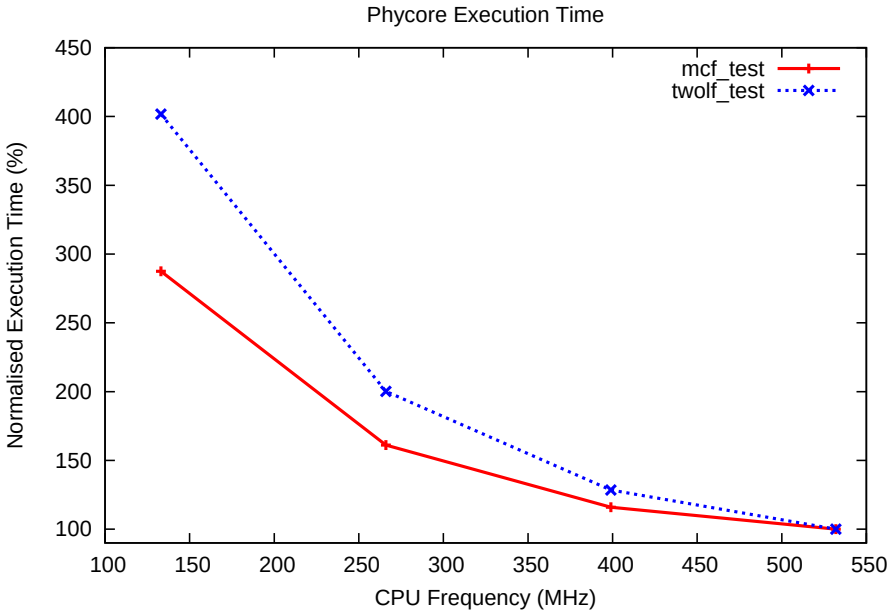


Figure A.20: Normalised execution time for memory-bound (`mcf_test`) and CPU-bound (`twolf_test`) benchmarks on Phycore.

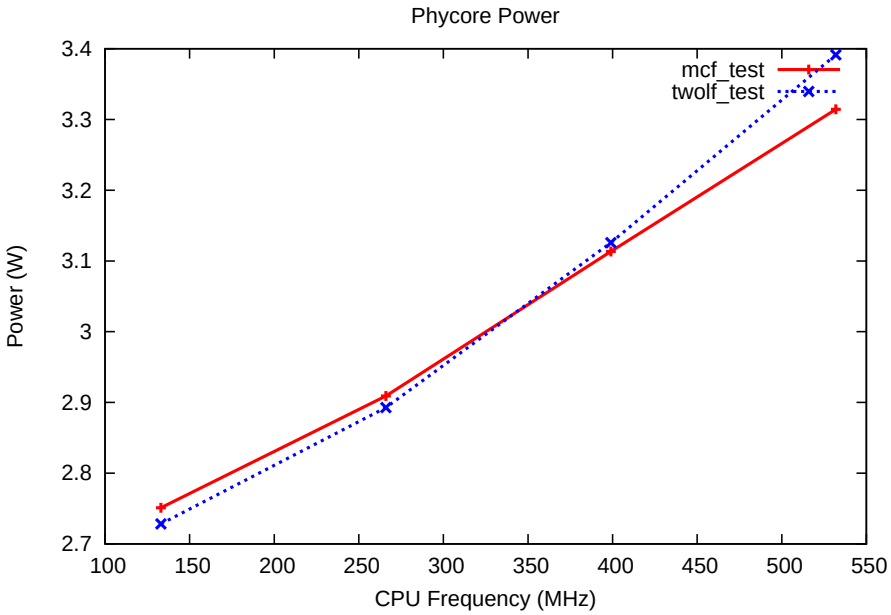


Figure A.21: Measured power for memory-bound (`mcf_test`) and CPU-bound (`twolf_test`) benchmarks on Phycore.

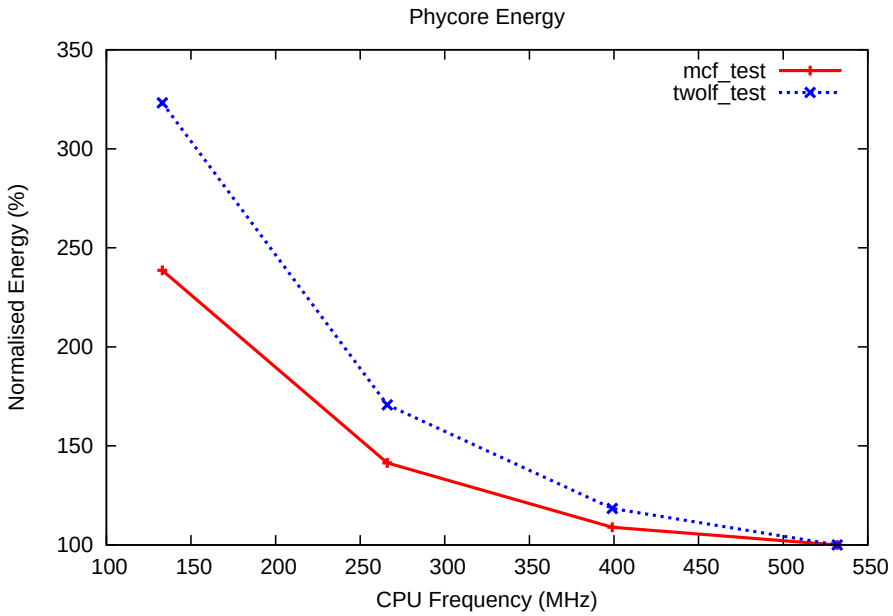


Figure A.22: Normalised Energy for memory-bound (`mcf_test`) and CPU-bound (`twolf_test`) benchmarks on Phycore.

benchmark has a lower energy-optimal frequency than the CPU-bound benchmark.

These results suggest that there is merit in investigating DVFS further on iMX31-based platforms (and comparing with ARM's IEM solution to DVFS). As mentioned, the difficulty when implementing the drivers required for DVFS on this platform resulted in very little further development.

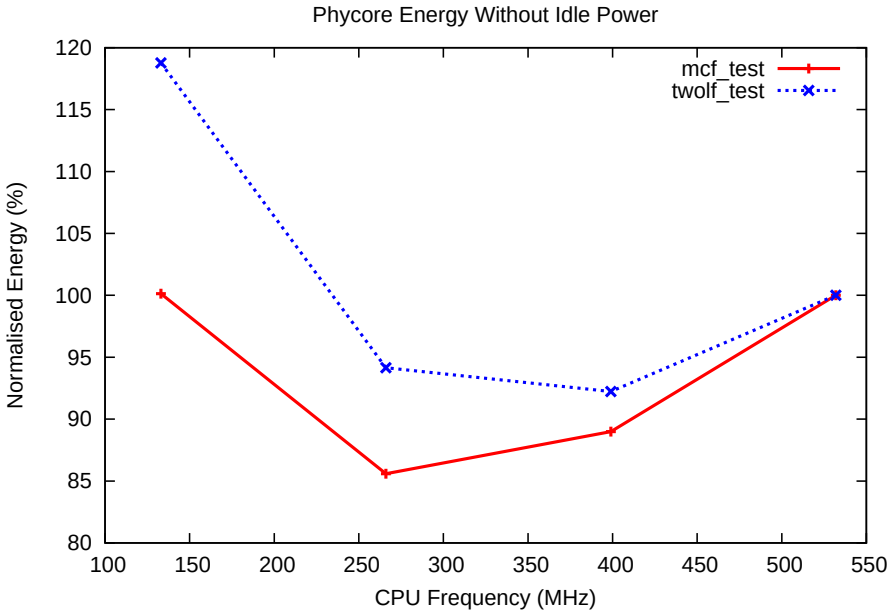


Figure A.23: Normalised Energy without idle power for memory-bound (`mcf_test`) and CPU-bound (`twolf_test`) benchmarks on Phycore.

A.5 Dell Latitude D600 (Pentium-M) — *Latitude*

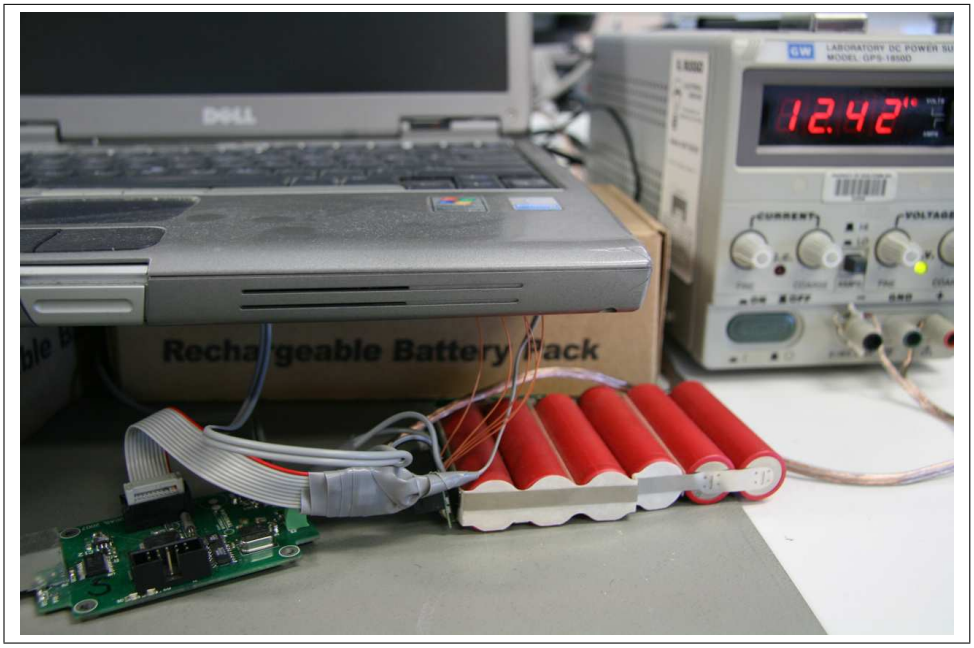


Figure A.24: Providing and measuring the power to the Latitude laptop via the battery.

Processor	Pentium-M 745 (Dothan)
Architecture	IA-32
Tested Settings	12
Variable Frequencies	1 1
Variable Voltages	1
Energy Measurement	Echidna
Performance Counters	2 + cycle counter
Performance Events	>164

The Dell Latitude D600 [31] laptop was one of two Intel Pentium-M based laptops which were tested, and exhibited a number of features confirming the theories in this thesis. It is based on an Intel Pentium-M 745 [79] which is a *Dothan* processor running at 1800MHz maximum frequency. The CPU is an IA-32 [72] based processor on a 90nm process with a front-side-bus running at 400MHz. It has a 32kB instruction cache and 32-kB write-back data cache. It has a 2MB level 2 cache, and uses a variant of the P6

microarchitecture. Its features include speculative and out-of-order execution, a 12-14 stage pipeline and register renaming. It was found to be of particular importance that the processor uses pre-fetching to increase the likelihood of level 2 cache hits.

The system uses an Intel 855PM chipset which supports DDR SDRAM, of which the laptop under test had 1 GB in two 512 MB modules. Major peripherals included in the system are a display with dimmable backlight connected to an ATI Radeon 9000 video controller on the AGP bus exported by the 855PM north bridge. 1000Base-T Ethernet, wireless ethernet, audio, a keyboard and touch pad. See the cited specifications for more details, since the peripherals are largely ignored for this study.

The system can be scaled to one of 12 settings. Not all of these were advertised via ACPI's P-states. The MSR settings which were configured by ACPI were examined, and then set directly using the `/dev/msr` interface in Linux for benchmarking, and, in the case of Koala, directly using MSR writes within the kernel. The MSR allows for the base frequency of 100MHz to multiplied by 6, 8, 9, ..., 17, or 18. Using a multiplier of 7 consistently caused the system to hang. Voltage scaling can be set via a VID, which communicates with the core power supply DC-DC converter. That converter generates a voltage which can be varied in 16mV steps. Communications between the processor and converter is via a direct parallel connection. The ACPI settings used by the machine were used as the basis for the chosen settings. Since not all frequencies were available via ACPI, the voltage for the extra frequencies was chosen via linear interpolation. Since the voltage can only be scaled in 16mV steps, this led to some non-linearities in the relationship — see Section 3.2.9 and Figure 3.11.

The system can be powered by either a battery, or a 19V DC supply provided by an mains-power adapter. Initially, power measurements were conducted by removing the battery and measuring the current provided by the power adapter using an echidna. This led to some very strange results — in some cases, the power used could *increase* when the frequency and voltage were *decreased* (see Figure A.32). This did not fit any previously discussed models. This strange behaviour was found to be caused by a change in the efficiency of the core voltage converter, which was dependent on both the current drawn by the CPU core, and the voltage with which it was supplied. When supplied at the battery's lower voltage ($\sim$12V), the core voltage converter's efficiency was constant. The effect is further described later in this Section — see Figures A.27 and A.32.

The problem was worked around by supplying the system from the battery rather than the power adapter. The battery was instrumented using an echidna. This was complicated by the battery management IC within the battery itself which must correctly communi-

cate with the laptop in order for the system to start up. It was further complicated via the battery management IC's protection features which disable the battery following any tampering (particularly disconnecting the battery cells from the battery management IC). The problem was solved by using patch wires to connect a second connector to the battery, with the echidna connected in series with the power pins. The battery was kept charged by permanently by connecting it to a laboratory power supply. See Figure A.24.

Triggering of the echidna was achieved via the system's parallel port.

Frequency switching overheads were the lowest of any system tested, with the CPU being stalled for only $10\mu s$. Frequency and voltage change are sequenced internally by hardware in the CPU, so changes to the frequency and voltage are effected by a simple write to an MSR.

The Pentium-M provides two performance counters and one cycle counter (referred to as the *time-stamp counter* (TSC) on x86). Both are quickly accessed by dedicated instructions. The two performance counters can each count one of a large variety of events as described in Intel's System Programming Guide [71]. A set of 164 events were seen to have some potential relevance to power and performance (this subset was chosen to minimise the benchmarking time). Events which were ignored included MMX-related events (since no MMX-enabled benchmark programs were used) and other clearly-irrelevant metrics.

The Dell Latitude D600 has an integrated temperature sensor which is primarily used to vary the speed of the system's cooling fan. This was accessible via the BIOS. Similarly, the fan speed can be set and read, also via the BIOS. Information available via the smart battery IC within the battery is available via the smart battery interfaces in ACPI, for which there are Linux drivers. Modifications to the `benchmarker` program were made to support the gathering of all of these statistics.

Figures A.25, A.26 and A.27 show how the Latitude laptop behaves when frequency scaling. Figure A.25 shows the execution time of a memory-bound and a CPU-bound benchmark. The CPU-bound benchmark's execution time scales with the inverse of the frequency, whereas the memory-bound benchmark's execution time stays nearly constant. This indicates that the memory-bound benchmark gets nearly no improvement in performance from a frequency increase. The result of this can be clearly seen in Figure A.27, where the two benchmarks have opposite behaviour – the CPU-bound benchmark increases its energy use as the frequency is increased (since the run-time is reduced). The memory-bound benchmark decreases its energy use as the frequency is increased, since the run-time remains the same and the power drawn is reduced.

The discontinuity in the Figure A.26 between 1100MHz and 1200MHz, and between

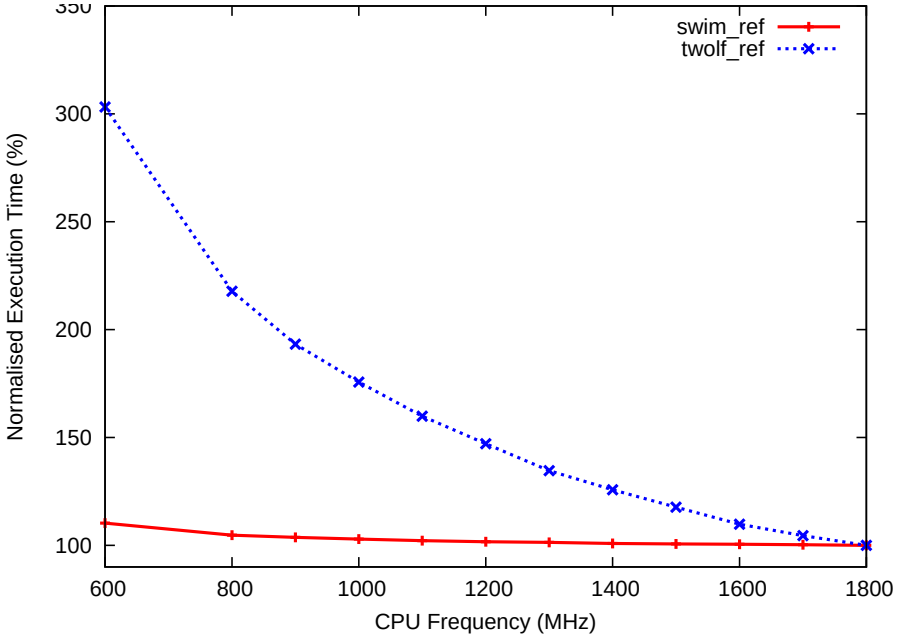


Figure A.25: Normalised execution time for memory-bound (`swim_test`) and CPU-bound (`twolf_test`) benchmarks on the Latitude laptop.

1700MHz and 1800MHz is caused by a discontinuity in the voltage setting. The voltage is adjusted by an equal step for all other intervals. See Figure 3.11 for illustration.

While `swim` and `gzip` show the opposite extremes of behaviour, other benchmarks show intermediate behaviours. Figure A.28 shows the `equake` benchmark shows distinct energy-optimal point that is at neither the maximum nor minimum frequency.

Figure A.29 shows the effect of frequency scaling without a voltage change. The total energy required for gzip is increased dramatically since the lower power due to a frequency reduction is more than compensated for by the increased run-time. Interestingly this shows that it is possible to save a small amount of energy for a memory-bound benchmark via frequency scaling alone — there is so little change in the run-time between the higher frequencies that the reduction in power due to a reduced frequency is enough to save some energy.

Figures A.30 and A.31 provide detail on the effect of using the power adapter. The power drawn is significantly higher due to extra losses in the DC-DC converters. More interestingly, compared with the case where the current at the battery is measured, there are new discontinuities. In the case where the voltage is kept constant at its maximum, and only the frequency is adjusted, it can be seen that by *reducing* the frequency for

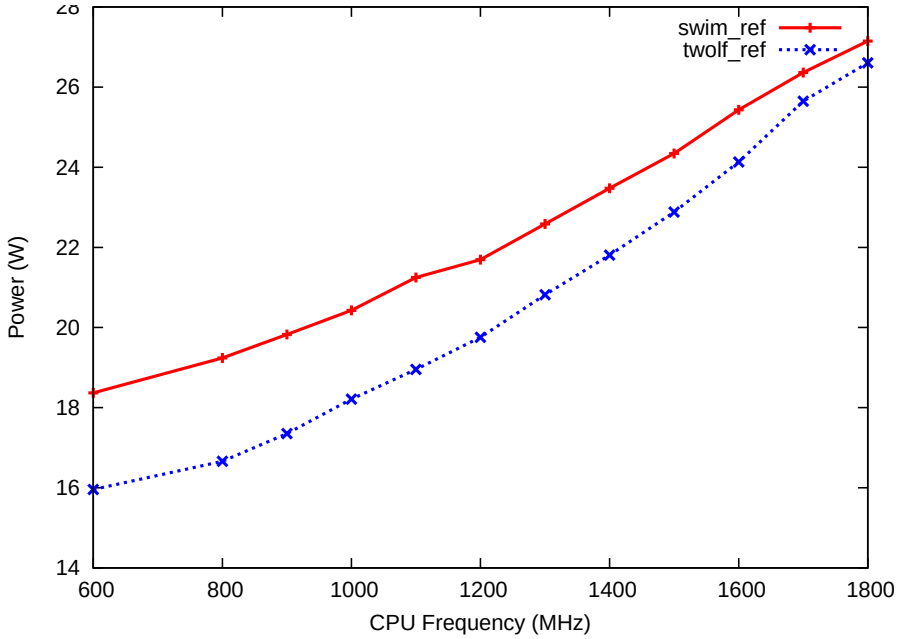


Figure A.26: Measured power for memory-bound (`swim_ref`) and CPU-bound (`twolf_ref`) benchmarks on the Latitude laptop.

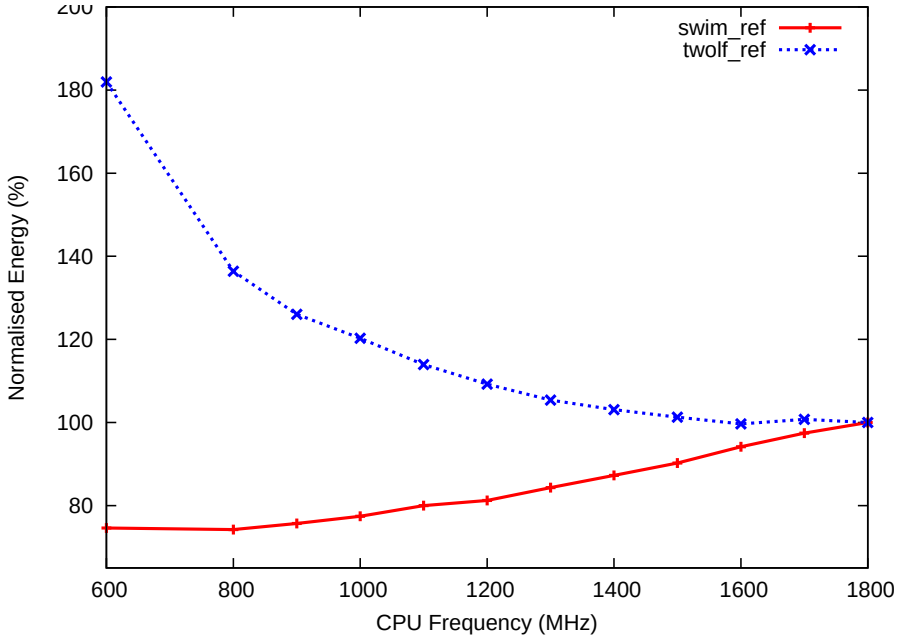


Figure A.27: Normalised Energy for memory-bound (`swim_ref`) and CPU-bound (`twolf_ref`) benchmarks on the Latitude laptop.

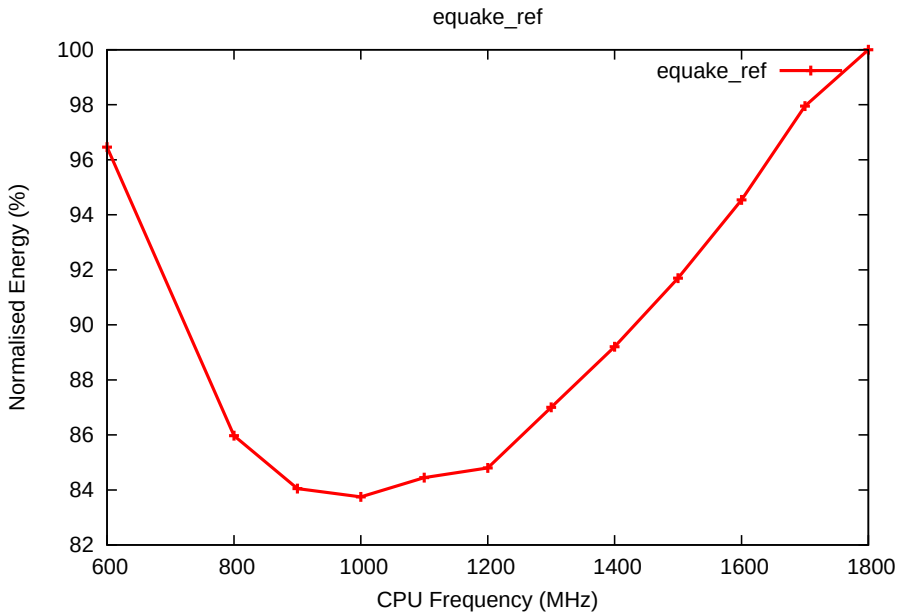


Figure A.28: Normalised Energy for the partially memory-bound (`equake_ref`) benchmark on the Latitude.

`swim` from 1500MHz to 1400MHz, the power is actually *increased*! The same can be said for the CPU-bound (on this platform) `gzip` between 900MHz and 1000MHz. This is completely counter-intuitive to all previously presented models, which would assume that a reduction in frequency should *always* reduce the power consumed. Note that this effect is not present when the power is measured directly from the battery. The effect can be traced to the efficiency of the DC-DC converter, which changes with load and input-voltage. At light loads with a high input voltage, the converter runs in discontinuous mode, and the particular converter used changes its modulation scheme, resulting in a dramatic change in the efficiency as it does so. Since the memory-bound `swim` has a lighter CPU load than the CPU-intensive `gzip`,

Other effects, such as the temperature and fan effects observed in Figure 3.7, also contribute to the irregularities in this graph (for example, the curve in the constant-voltage plot is caused by a reduced temperature at lower frequencies).

If the idle power is considered and subtracted in accordance with Section 4.5, the behaviour of the system changes dramatically. Comparing Figure A.27 with Figure A.32 shows the effect of removing a typical idle power component from the workload's energy measurement. The effect of increasing the idle power from zero to a typical value

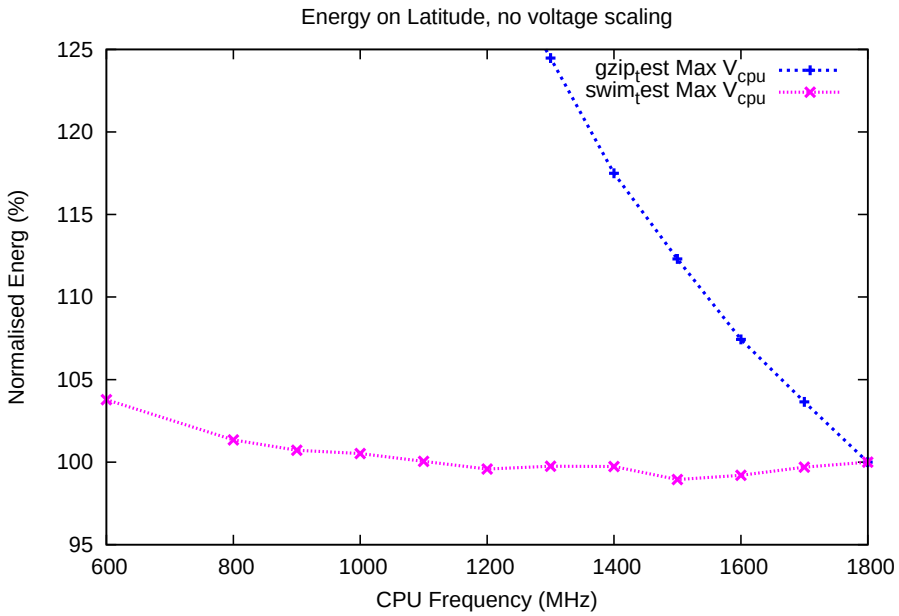


Figure A.29: Normalised Energy for memory-bound (`swim_test`) and CPU-bound (`gzip_test`) benchmarks on the Latitude laptop while running at the maximum voltage (1.34V) from the power adapter.

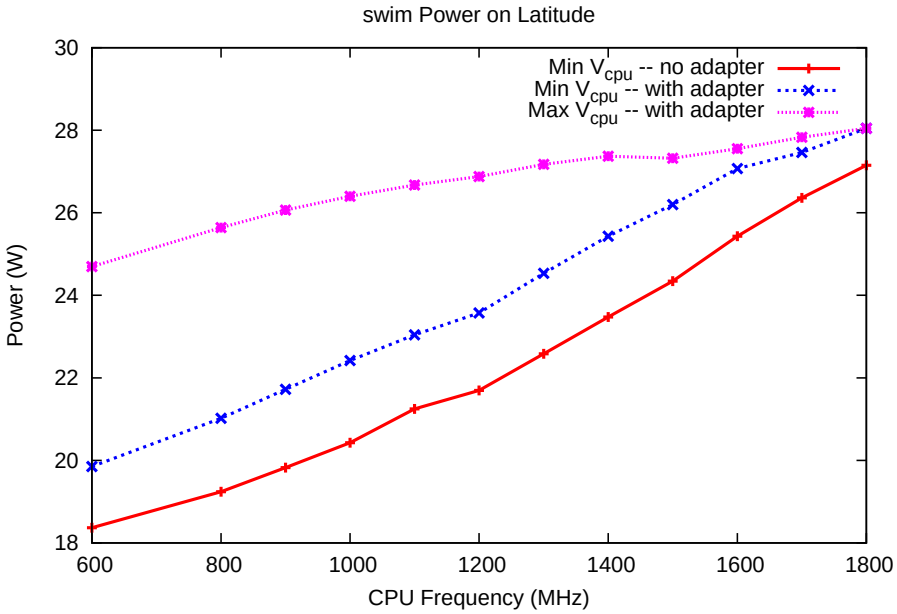


Figure A.30: Measured power for memory-bound (`swim_test`) benchmarks on the Latitude laptop, measured at both battery and power adapter.

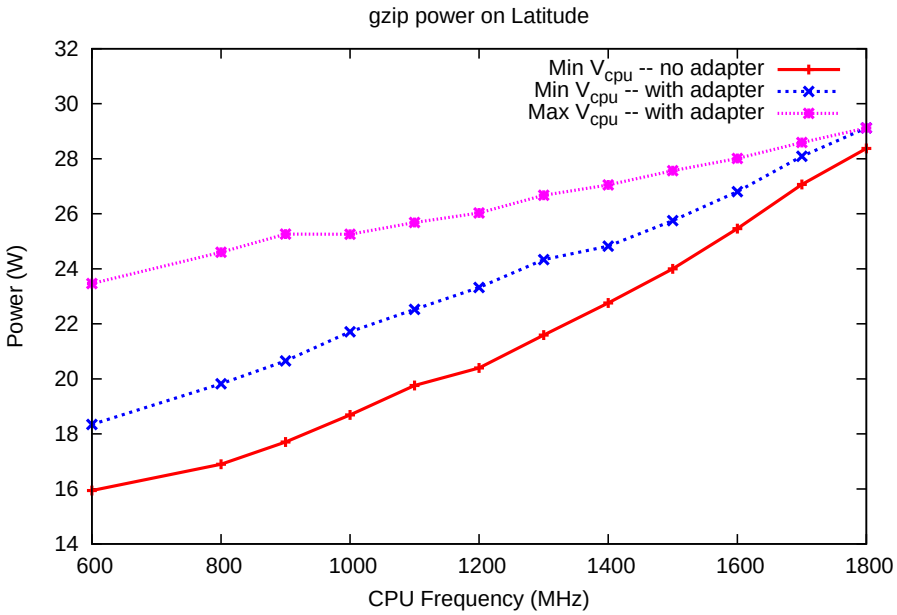


Figure A.31: Measured power for CPU-bound (`gzip_test`) benchmarks on the Latitude laptop, measured at both battery and power adapter.

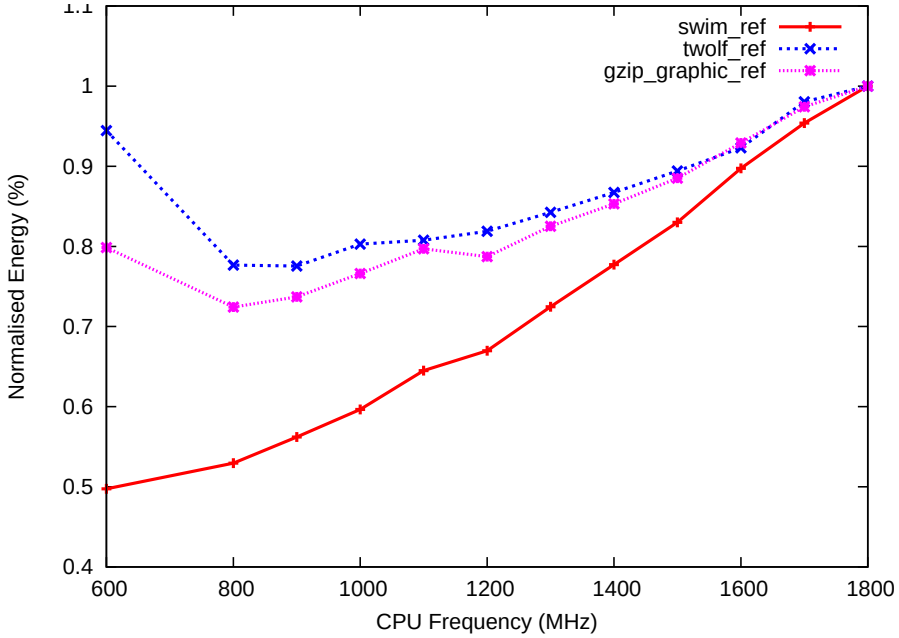


Figure A.32: Normalised energy for CPU-bound (`twolf_ref`) and memory-bound (`swim_ref`) benchmarks on Latitude, with the idle power component removed. `gzip_graphic_ref` is included here to demonstrate a clear non-linearity.

is to bias the minimum-energy frequency to a lower-performance setting, since the time spent idle (of which there is more at higher-performance settings) uses more energy. The `gzip_graphic_ref` was included in the latter figure to illustrate the case where the interaction between an increased idle power and a non-linear frequency-voltage relationship could lead to significant irregularities in the energy-frequency graph. Between 1100MHz and 1200MHz (where there is a non-linearity in the relationship between voltage and frequency) we see the energy increase as we reduce the frequency. This effect has been either undiscovered or ignored in academic DVFS research.

Note that while all benchmarks generally run more efficiently at lower frequencies, the energy savings compared with the relative performance is very different for each benchmark. This means that in a resource-constrained system memory-bound benchmarks should be scaled more aggressively than CPU-bound benchmarks.

The Latitude laptop was the only one of the systems which was tested at different temperatures. The results are shown in Chapter 3 — Figure 3.7. Changing the temperature alters the balance between dynamic and static power and the idle power. This changes the

optimal frequency setpoint. Testing was done using a small refrigerator to modify the ambient temperature around the laptop.

The Latitude laptop was the primary system used for experimentation in the context of this thesis.

A.6 IBM Thinkpad T43 (Pentium-M) — *T43*

Figure A.33: IBM T43 laptop with Echidna

Processor	Pentium-M 750 (Dothan)
Architecture	IA-32
Tested Settings	8
Variable Frequencies	1 1
Variable Voltages	1
Energy Measurement	Echidna
Performance Counters	2 + cycle counter
Performance Events	>164

The IBM Thinkpad T43 uses a slightly different version of the Pentium-M processor to the Dell Latitude D600 — the Pentium-M 750 [78]. It has a higher peak clock frequency (1866MHz) and a higher front-side bus (533MHz). The peripherals available are very similar to the Latitude.

Power was measured by removing the battery, and placing an echidna in-line with the power adapter. None of the non-linearities present in the Latitude laptop were observed in the T43. Triggering was via the parallel port.

The T43 was used only for preliminary experiments due to subsequent access issues.

A.7 Asus EEEPC 901 (Atom) — *EEEPC*

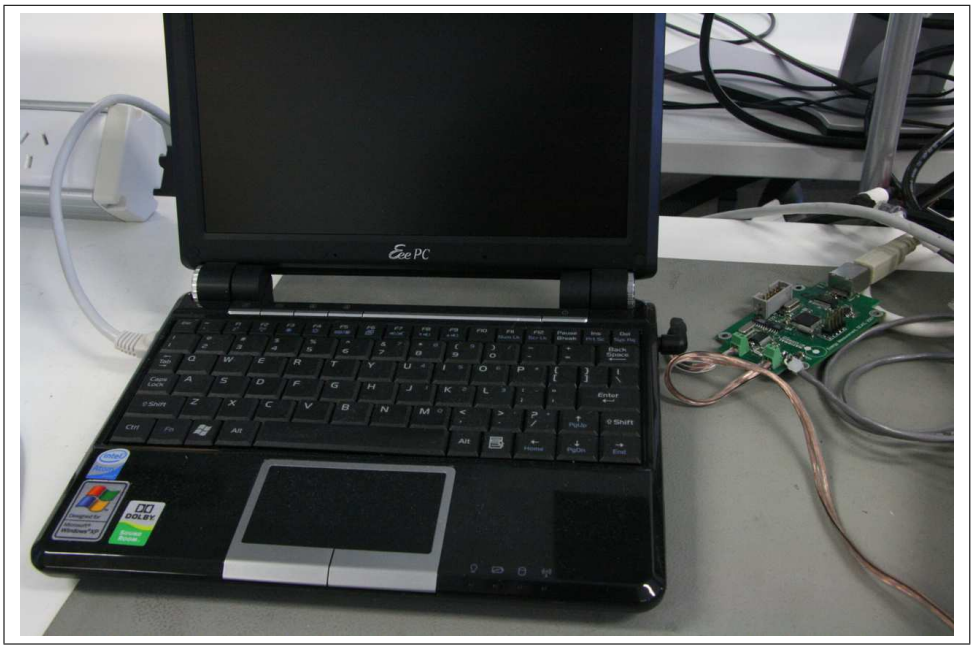


Figure A.34: EEEPC Atom-based computer

Processor	Atom N270
Architecture	IA-32
Tested Settings	7
Variable Frequencies	1
Variable Voltages	1
Energy Measurement	Echidna
Performance Counters	2 + cycle counter
Performance Events	>113

The EEEPC 901 [11] is based on an Intel Atom processor [77], which is intended for low-power applications such as embedded systems and netbooks. This processor is based around a different microarchitecture to the Pentium-M, but implements the same IA-32 ISA. It uses a two-issue, in-order pipeline. It supports *symmetric multi-threading* (SMT) in order to avoid pipeline stalls due to long-latency instructions (such as during cache misses), however SMT was disabled for all experiments. It has a 32-kB instruction cache and 24-kB write-back data cache, with a 512-kB, 8-way L2 cache.

The EEEPC 901 has 1GB of DDR2 SDRAM, an 8.9" display, 20GB flash drive, wired and wireless ethernet, and a 6-cell Lithium ION battery among other peripherals.

Speed setting is done in the same way as the Pentium-M — an MSR is set with a frequency and voltage ID, and the frequency change sequence is handled in hardware. The P-states provided by ACPI were used to generate the seven voltage and frequency settings. These multiplied a base frequency of 133MHz by 6 to give 798MHz, up to a multiplier of 12 which gave 1596MHz. The voltage varies between 0.940V for the lowest setting to 1.276V for the upper setting. The power supply chip interfaces to the Atom via a dedicated bus. The memory frequency remains constant across all CPU frequencies.

Energy measurements were taken using an Echidna at the AC-adapter input. The battery was removed. Triggering was performed via a USB-serial converter driving the DTR line.

The Atom includes two performance counters and a time-stamp counter. The events which can be monitored by the two counters are described in the Intel system programming guide [71]. A set of 113 events were deemed potentially relevant to the performance and power of the system. Note that the time-stamp counter in the Atom behaves differently to every other platform examined here: it counts real-time (in ns) rather than cycles.

The behaviour of the EEEPC 901 is shown in Figures A.35, A.36 and A.37. It is clear from Figure A.36 that the memory power consumption has a big effect on the over-all system power, with the memory-bound `art_1` and `art_2` benchmarks having a much higher power draw than the other less memory-bound benchmarks.

An initial examination of the energy use of an EEEPC 901 platform showed that, in the case of a 0-power idle mode, and no thermal problems, the optimal DVFS policy was trivial: run at the highest CPU frequency. However, when a typical idle power is subtracted from the total power and the energy calculated (see Section 4.5) it is clear that significant energy savings are possible as shown in Figure A.38

Of note on this platform is the high power when idle – it is nearly as high as the power required to run at full speed. Since the Atom was designed specifically for a low-power idle modes, the operating system is probably not making appropriate use of the available idle modes.

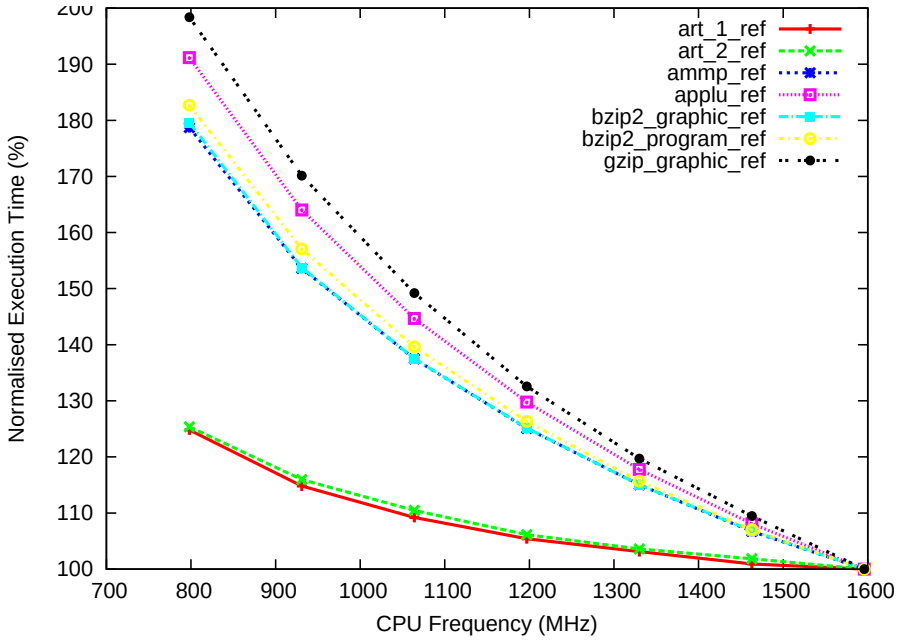


Figure A.35: Normalised execution time for a range of benchmarks on the EEEPC 901 Netbook.

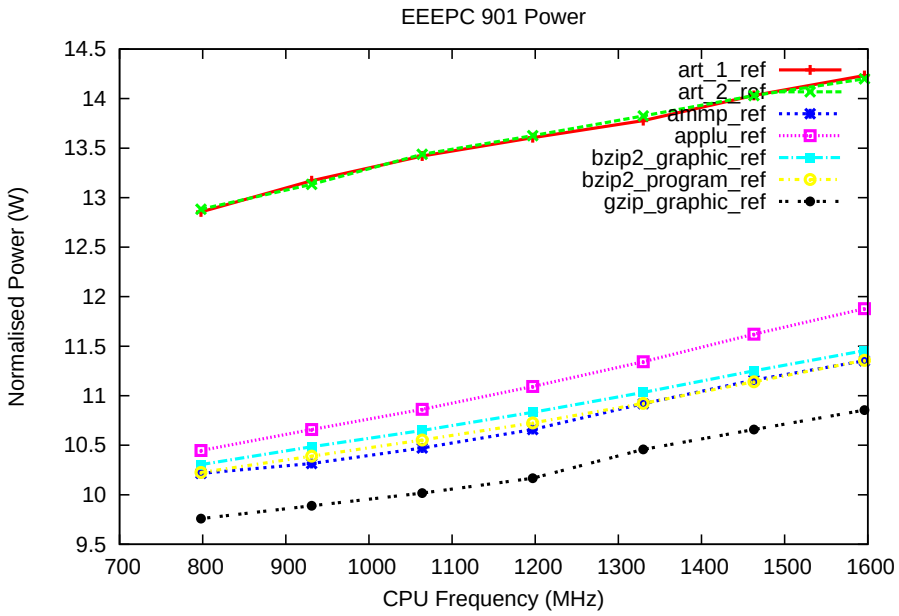


Figure A.36: Measured power for a range of benchmarks on the EEEPC 901 Netbook.

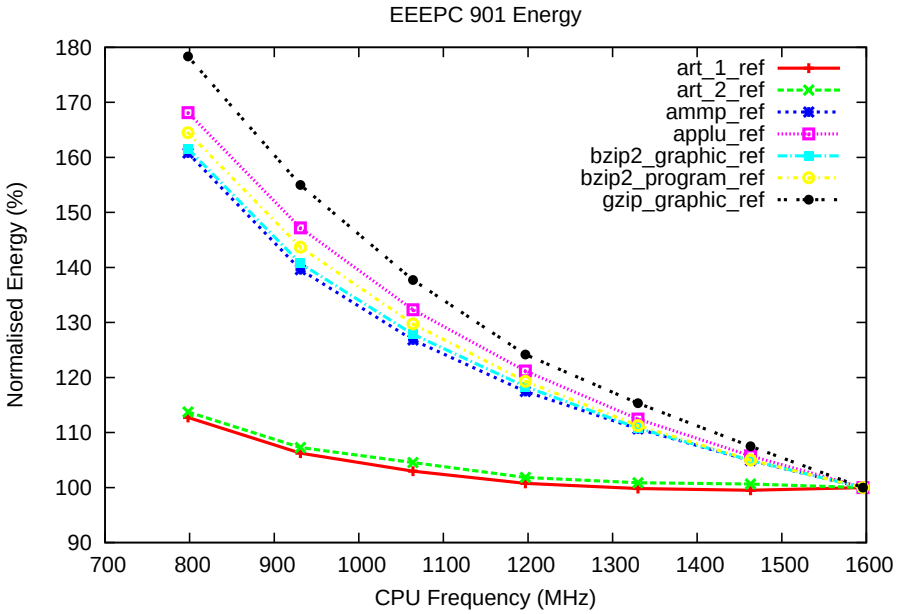


Figure A.37: Normalised Energy for a range of benchmarks on the EEEPC 901 Netbook.

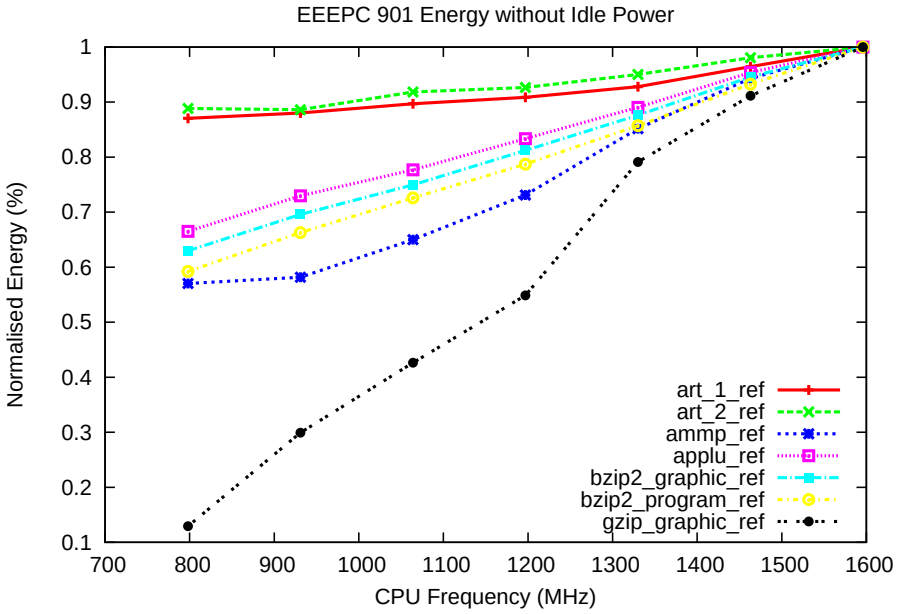


Figure A.38: Normalised Energy for a range of benchmarks on the EEEPC 901 Netbook, with the idle power subtracted.

A.8 Compucon K1-1000D (AMD Opteron 246) — *Opteron*

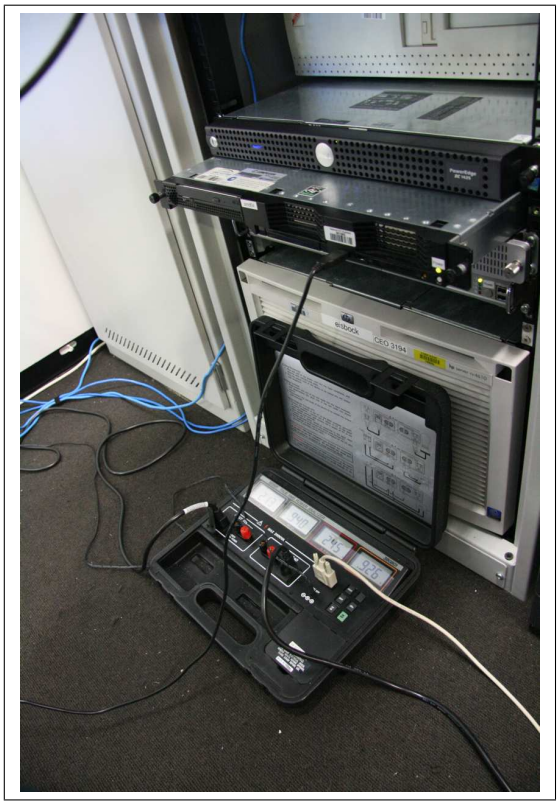


Figure A.39: Compucon K1-1000D Opteron-based server with Extech power analyser

The AMD Opteron-based server (a Compucon K1-1000D) is based on an AMD Opteron 246 processor clocked at 2 GHz. It has a 1 1MB cache interfaced to a 1 GHz hypertransport bus. It has 512 MB of DDR400 SDRAM by default (some experiments added a second 512 MB DIMM, allowing for dual-channel memory accesses).

The Opteron's voltage scaling system is coordinated by software. The result is a significant overhead for voltage and frequency changes, since the CPU is unavailable for the duration of the voltage/frequency switch. This is exacerbated by the Opteron's requirement that the voltage be scaled to the maximum value before the frequency switch can be performed, and subsequently to the target voltage. A driver was written by Etienne Le Sueur to scale the voltage and frequency, and the voltage was ramped much faster

Processor	AMD Opteron 246
Architecture	IA-32
Tested Settings	5
Variable Frequencies	2
Variable Voltages	1
Energy Measurement	Extech True RMS Power Analyzer 380801
Performance Counters	4 + cycle counter
Performance Events	$>$177

than the specification, resulting in a worst-case switching overhead of 140μs. The worst case when run within the specification was $\sim$2ms. The best case switching time (with the out-of-spec voltage ramp) was 15μs.

Five frequency and voltage pairs were chosen between 0.8 GHz at 0.9 V and 2.0 GHz at 1.5 V. It had been assumed that the memory frequency would remain constant for all CPU frequencies, but it was discovered experimentally (using a *digital storage oscilloscope* (DSO)) that the memory frequency lowers to 160 MHz at a CPU frequency of 0.8 GHz. At all other settings the memory frequency remains at 200 MHz.

A cycle counter and four user-configurable counters were available. The counters can measure several hundred events, of which we examined 177.

Power measurement was performed using a commercial AC power meter with an accuracy of 0.9% sampled at 2.5 Hz. This was inserted between the wall socket and the machine's power plug and thus measured the system's total AC power consumption (which is appropriate, since a reduction of the total AC power consumption being the main goal for server power management). The data was received via a serial connection by a program running on the machine itself. The power and performance overhead these measurements is considered negligible.

The basic results from the Opteron-based server are shown in Figures A.40, A.41 and A.42. The memory-bound benchmark is highly memory-bound with a 20% increase in execution time for a halving of the frequency. The CPU-bound benchmark is only slightly less than double the execution time for half of the frequency, as expected. The system power is clearly higher for the memory-bound benchmark due to the extra power drawn by the memory itself. At the lowest frequency, a much lower power is drawn for both benchmarks, and this is attributable to the lower memory frequency for that setting. In terms of the total energy used to run the benchmark, the CPU-bound benchmark increases to the left, indicating that the reduced power at lower-frequency settings does not compensate for the extra energy drawn due to the longer run-time. Since the memory-bound

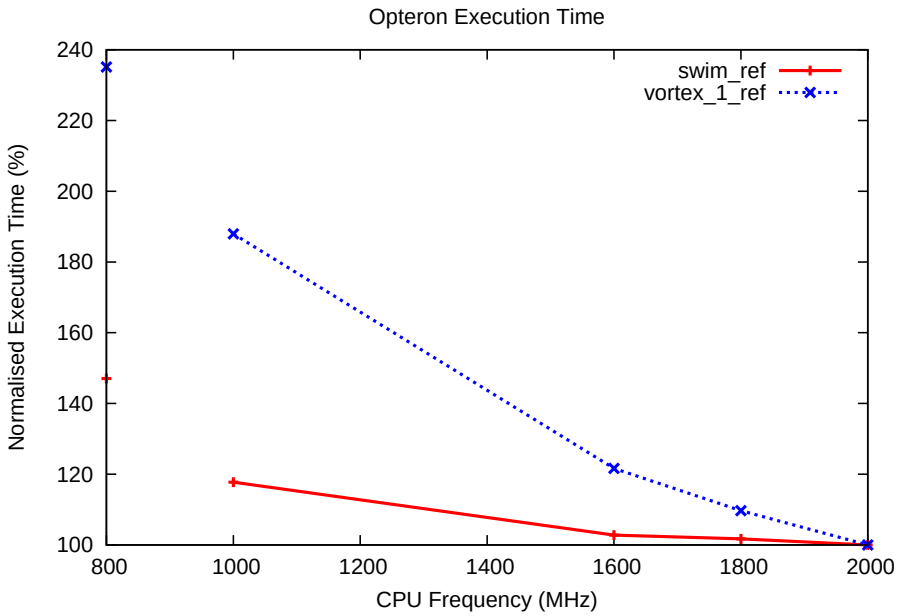


Figure A.40: Normalised execution time for memory-bound (`swim_test`) and CPU-bound (`vortex_1_test`) benchmarks on the Opteron-based server.

benchmark's run-time is nearly independent of CPU frequency, it exhibits an energy saving of more than 10% at reduced frequency settings.

When we subtract the energy which would be used to idle for the execution time of the benchmark, we see that the extra energy required to run the benchmark decreases with CPU frequency, and that the lowest frequency saves disproportionally more energy thanks to the reduced memory frequency.

The Opteron-based server was used underwent testing for nearly all aspects of this thesis.

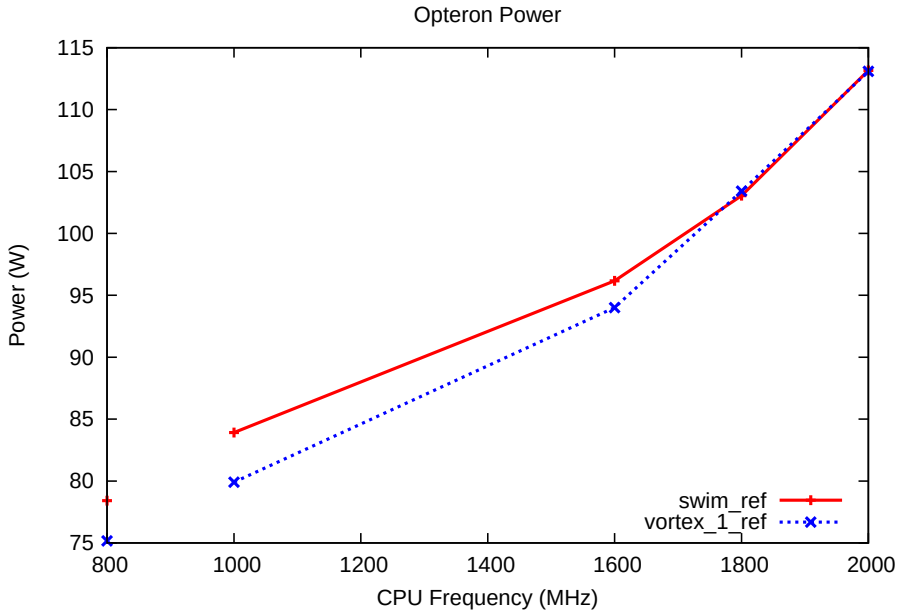


Figure A.41: Measured power for memory-bound (`swim_ref`) and CPU-bound (`vortex_1_ref`) benchmarks on the Opteron-based server.

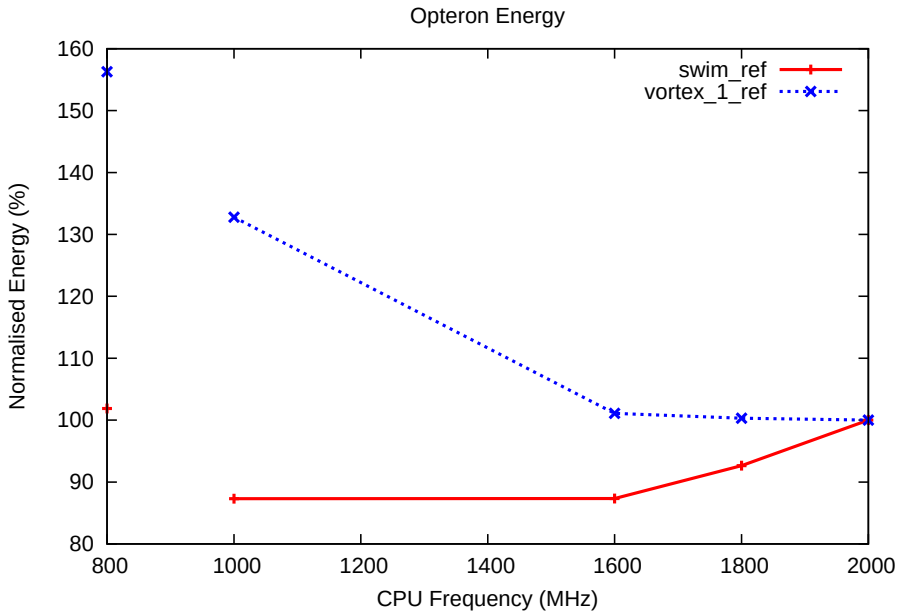


Figure A.42: Normalised Energy for memory-bound (`swim_ref`) and CPU-bound (`vortex_1_ref`) benchmarks on the Opteron-based server.

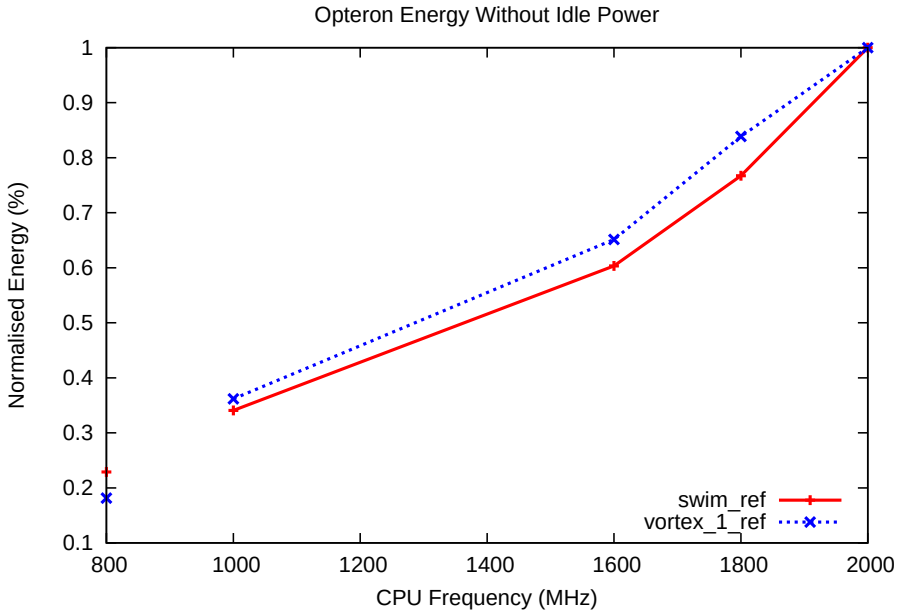


Figure A.43: Normalised energy for CPU-bound (`vortex_1_ref`) and memory-bound (`swim_ref`) benchmarks on Opteron, with the idle power component removed.

A.9 Intel Menlow Software Developer's Platform (Silverthorne) — *Menlow*

Some tests were conducted on an Intel Menlow developer's platform which used a pre-release version of the Atom processor (Silverthorne). The platform is a developer's system utilising integrated chipsets designed for ultra-mobile PCs. The details of this platform and its results are not discussed here due to a non-disclosure agreement, but in any case the system did not exhibit useful energy savings via frequency and voltage scaling using Koala.

A.10 Dell Server (Intel Xeon) — *Xeon*

A multi-core Intel Netburst-era Xeon-based system was tested. Unfortunately, with only three possible settings, this platform was considered un-interesting and therefore results are not presented here.

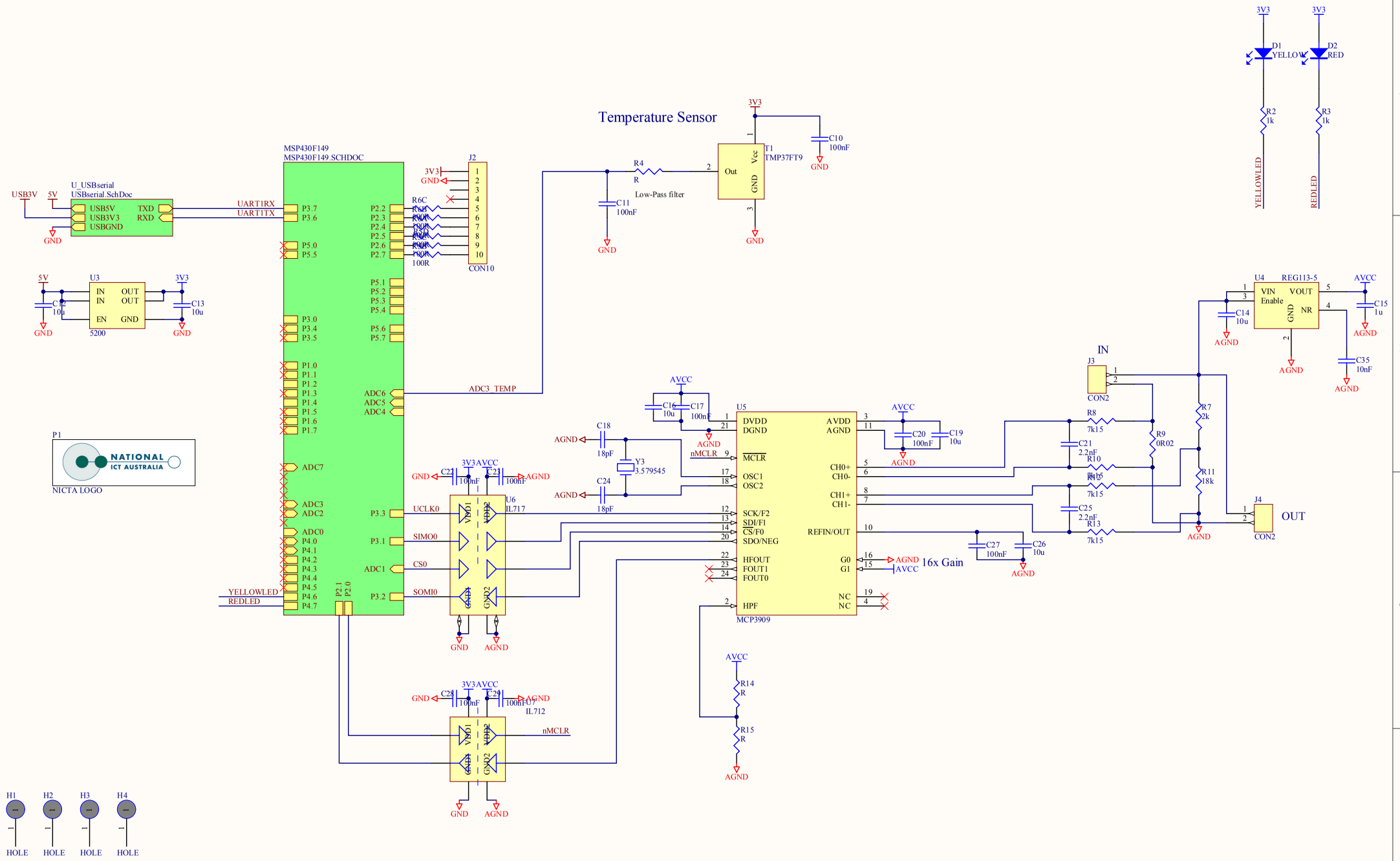
APPENDIX B

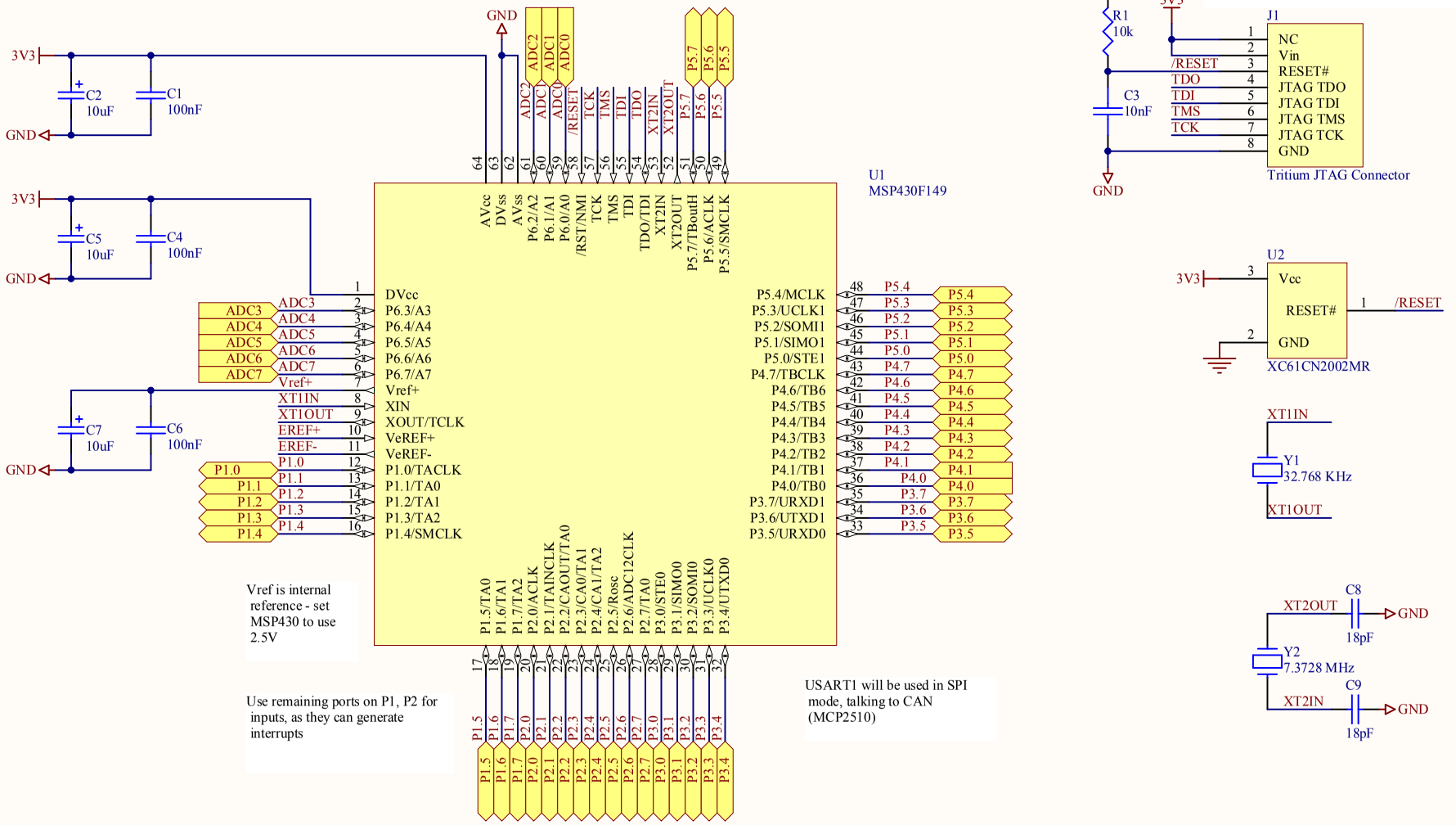
ECHIDNA

The Echidna is a simple circuit consisting of an FTDI 232R [52] USB to serial converter, an MSP430F149 microcontroller [153], and an MCP3909 energy measurement IC [103]. These provide USB connectivity, some programmable processing, and a simultaneously sampling, 16-bit sigma-delta ADC (with the appropriate signal conditioning circuitry for voltage and current measurements). Modifications were made to the basic circuit to scale the signals appropriately for different current and voltage ranges (for example, the Latitude D600 laptop uses a 16V input, whereas PLEB2 requires a 4.1V supply). Echidna has an accuracy of better than $1mA$ and $1mV$ for current and voltage respectively, giving a power accuracy of about $5mW$ or 0.4%. In some cases, the voltage accuracy was reduced when the voltage range was increased, however, the percentage errors remain less than 1%.

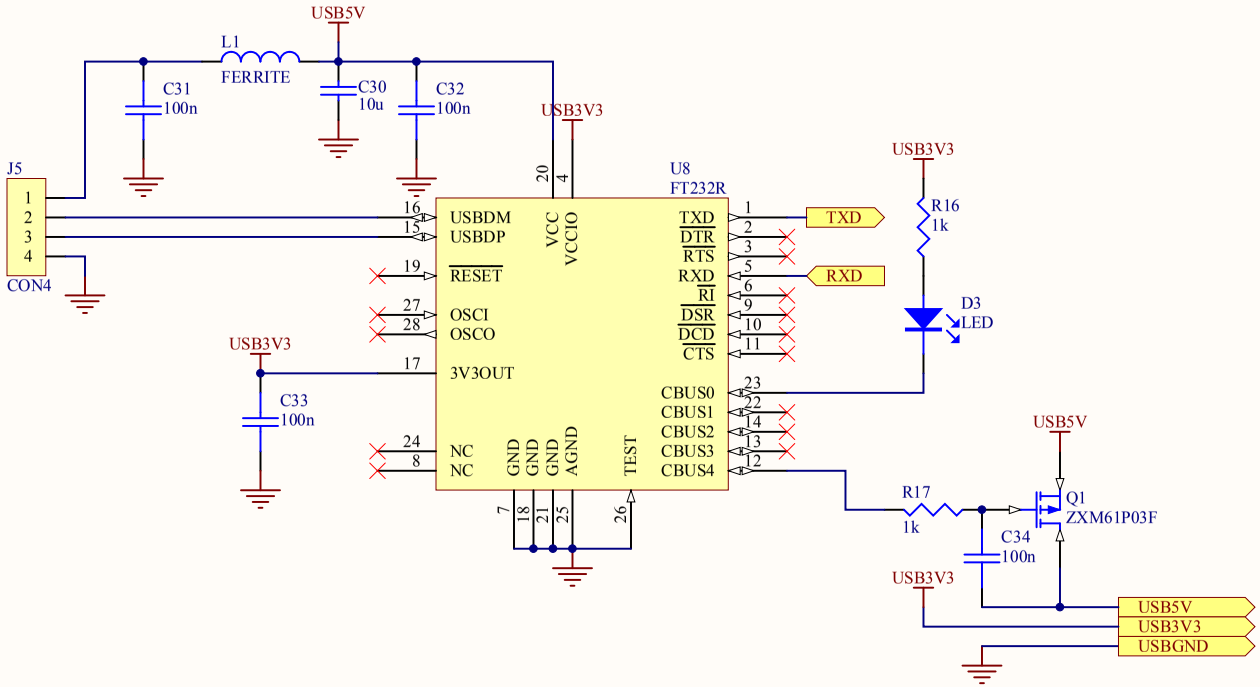
Echidna is supplied with a trigger signal in the form of a logic level (0, or 3.3V). A high signal indicates that the device should be measuring and accumulating data. A high-to-low transition indicates that it should print any accumulated data to the serial port and zero out the accumulators. The total energy is integrated while the device is accumulating, allowing for an accurate energy measurement. Echidna samples and accumulates at approximately 4kHz.

Triggers are provided from platforms via a number of means. Some are triggered via PC-style parallel ports, others by *general-purpose input/output* (GPIO) pins, others using the DTR line on either a real, or USB, serial port. Triggering via these hardware mechanisms allows for accurate timing — the measurement will begin very close to the start of a benchmark, and end very close to the end.


Temperature Sensor
MSP430F149
MSP430F149.SCHDOC
U_USBserial
USBserial.SchDoc
USB5V
USB3V3
USBGND
TXD
RXD
UART1RX
UART1TX
Low-Pass filter
ADC3_TEMP
MCP3909
IL717
IL712
16x Gain
IN
OUT
NICTA LOGO
HOLE
Title Echidna Top Level Schematic
Size: A4
Number:
Revision:
Date: 10/06/2009
Time: 5:20:06 PM
Sheet of
File: Z:\projects\phd\work\pm2\energymeas2\hardware\Documents\TopLevel.Sch
National ICT Australia
Level 4
223 ANZAC Pde
Kensington
NSW, 2052



Title	***MSP430F149 Microcontroller***		*National ICT Australia* *Level 4* *223 ANZAC Pde* *Kensington* *NSW, 2052*
Size: A4	Number:	Revision:	
Date: 10/06/2009	Time: 5:20:07 PM	Sheet of	
File: Z:\projects\phd\work\pm2\energymeas2\hardware\Documents\MSP430F149.SCHDOC			



Title	**USB-UART interface**		*National ICT Australia* *Level 4* *223 ANZAC Pde* *Kensington* *NSW, 2052*
Size: A4	Number:	Revision:	
Date: 10/06/2009	Time: 5:20:07 PM	Sheet of	
File: Z:\projects\phd\work\pm2\energymeas2\hardware\Documents\USBserial.SchDoc