

An Analysis of the Effectiveness of Energy Management
on Modern Computer Processors

THE UNIVERSITY OF
NEW SOUTH WALES



SYDNEY • AUSTRALIA

Master of Science

School of Computer Science and Engineering

The University of New South Wales

Etienne Le Sueur

June 22, 2011

Abstract

Managing energy consumption has become a critical issue for computer system designers. End-users are forced to charge battery-powered devices on a daily basis because battery technology has not kept pace with the demands of the power-hungry processors and peripherals used in today's mobile devices. Thousands of servers in data-centres also use power-hungry processors which result in huge costs for power and cooling. This is impacting our environment by accelerating global climate-change.

To help mitigate these problems, processor manufacturers have implemented various power-management mechanisms, including *dynamic voltage and frequency scaling* (DVFS) and low-power idle modes, to reduce and manage the power consumption of their processors. These mechanisms are controlled by the *operating system* (OS) and in the past, their use could result in significant improvements in energy efficiency. However, changes in semiconductor technology are altering the effectiveness of these mechanisms.

This thesis shows that on recent platforms, using DVFS results in only marginal reductions in system-level energy consumption for realistic workloads such as MPEG video playback. However, we find that the system-level energy consumption of lightly loaded web-servers can be reduced without impacting throughput or response latency. This results in an improvement in energy efficiency.

However, several trends exist which are leading to diminishing returns from traditional power-management mechanisms. We analyse these trends to determine their impact on the effectiveness of the energy-management techniques used in systems today, and look into the future to see what might help to reduce the energy consumption of the computer systems of tomorrow.

Acknowledgements

Throughout the time spent working on this thesis I have had the opportunity to work with and learn from many of the brightest and most intelligent people I have ever known. My supervisor and the leader of the lab in which I spent most of my time, Gernot Heiser, deserves my utmost appreciation and thanks. Without his constant guidance and support, I would not have been able to complete this work. Furthermore, as the leader of the ERTOS group, I am constantly impressed at the level of enthusiasm and support that he gives to the group as a whole. My co-supervisor Peter Chubb also deserves my thanks for always being available to bounce ideas off.

I began as a research assistant working under Dave Snowdon on his PhD research. Much of the work in this thesis builds on the knowledge that I gathered during my time working with him. I take this opportunity to thank him for passing on some of his vast knowledge to me.

My fellow graduate students Aaron Carroll, Bernard Blackham and Godfrey van der Linden were also very much involved in my work and were always available to help me when I had a problem I didn't know how to solve.

My fiancé Romany was always willing to proof my writing before looming deadlines, which I greatly appreciate.

Numerous other people in the ERTOS lab have helped me over the last few years and I will take this opportunity to thank them as well.

Originality Statement

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.

Etienne Le Sueur, June 22, 2011

Publications

Slow Down or Sleep, That is the Question Etienne Le Sueur and Gernot Heiser, *Proceedings of the 2011 Usenix Annual Technical Conference, Portland, Oregon, USA, June, 2011.*

Dynamic Voltage and Frequency Scaling: The Laws of Diminishing Returns Etienne Le Sueur and Gernot Heiser *Proceedings of the 2010 Workshop on Power Aware Computing, HotPower'10, Vancouver, Canada, October, 2010*

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*

Contents

Abstract	i
Acknowledgements	iii
Publications	vii
Contents	xi
List of Figures	xv
List of Tables	0
1 Introduction and Motivation	2
1.1 Research Questions	4
1.2 Contribution	4
1.3 Thesis Overview	4
2 Background and Related Work	6
2.1 Introduction	6
2.2 Power Consumption Characteristics of Transistors	6
2.2.1 Transistor scaling	7
2.2.2 Dynamic power dissipation in CMOS circuits	7
2.2.3 Static power dissipation in CMOS circuits	9
2.3 CPU Power-management Mechanisms	9
2.3.1 Dynamic voltage and frequency scaling	10
2.3.2 Idle states (C states)	10
2.3.3 Intel's power-management unit and <i>TurboBoost</i>	12
2.3.4 Dynamic cache resizing	12
2.3.5 Adaptive voltage scaling	13
2.3.6 Discussion	14
2.4 Evaluating Energy-management Techniques	14
2.4.1 Measurement and run-time profiling	14
2.4.2 Simulation and trace analysis	15

2.4.3	Modelling and accounting	17
2.4.4	System-level analysis	21
2.4.5	Scheduling on multi-core systems	22
2.4.6	Discussion	23
2.5	OS Energy Management	23
2.5.1	Research frameworks	23
2.5.2	Linux	25
2.5.3	Discussion	27
3	The Impact of Memory Throughput	28
3.1	Introduction	28
3.2	The Effects of Aggressive Prefetching	30
3.3	Modelling Workload Performance Under Varying CPU Frequency	31
3.3.1	Model characterisation	32
3.3.2	Developing a synthetic characterisation workload	34
3.3.3	Prefetching effects	35
3.4	Memory Throughput and DVFS	39
3.5	Conclusions	40
4	Slow Down or Sleep?	42
4.1	Introduction	42
4.2	Compute-intensive Workloads	43
4.2.1	Methodology	44
4.2.2	Results	45
4.2.3	Discussion	45
4.3	Padding with Idle-energy	46
4.3.1	Energy-delay product	46
4.3.2	C state overview	47
4.3.3	Padding with different C states	51
4.3.4	Discussion	52
4.4	Bursty/Periodic Workloads	52
4.4.1	Artificially introducing slack-time into SPEC workloads	53
4.4.2	MPEG playback workload	56
4.4.3	Hardware acceleration of MPEG decode	59
4.4.4	Web-server workload	59
4.4.5	Discussion	64
4.5	Conclusions	65
5	The Laws of Diminishing Returns	68
5.1	Transistor Scaling	68
5.2	Memory Performance	69
5.3	Improved Idle Modes	70
5.4	Asynchronously Clocked Caches	70

5.5	Multi-core Processors	71
5.6	Conclusions	72
6	Conclusions	74
6.1	Summary	74
6.2	The Future of CPU Power Management	76
6.3	Future Work	76
6.3.1	DVFS on hardware acceleration units	76
6.3.2	Per-core DVFS	77
6.3.3	Modelling with improved performance counters	77
6.3.4	OS scheduler optimisation for improved C state usage	77
6.4	Conclusion	78
	Glossary	80
	References	81

List of Figures

2.1	High-level components forming an n-channel MOSFET.	7
2.2	Construction of a NAND gate from MOSFET devices.	8
2.3	Possible C state transitions for an Intel Core i7 processor. All transitions must go through C0, except to the special C1E state, entered when all cores enter C1.	11
2.4	Future generations of Intel processors will have automatic promotion/demotion of idle states, thus removing the requirement to transition through C0, minimising the number of wake-ups. Current generations are restricted to the state transitions shown in Figure 2.3.	11
2.5	Variability in semiconductor power consumption, as predicted by the ITRS.	13
2.6	Plot of model residuals against measured value for an execution time model with four parameters [R, 2011].	20
3.1	Runtime vs. frequency (top) and CPU cycles vs. frequency (bottom) for several SPEC CPU2000 benchmarks on a Pentium-M. All values are normalised to the maximum frequency, 1.8 GHz.	29
3.2	Change in execution time of the SPEC CPU 2000 workloads on a recent AMD Opteron with prefetching disabled . Note that the performance of 181.mcf actually improves with prefetching disabled.	30
3.3	Comparison of model estimated execute time (top) and model estimated energy consumption (bottom) on the Sledgehammer platform with a four-parameter model. Model accuracy is good for most benchmarks in the set, however lbm.test and mcf.test on the left have huge over- and under-estimations respectively.	33
3.4	Comparison of normalised cycle count vs. CPU frequency for several SPEC CPU2000 benchmarks on a more recent Core i7. 3.60 GHz represents a TurboBoost frequency, all values are normalised to this. Both <i>swim</i> and <i>mcf</i> show a non-linear relationship that is caused by hardware prefetching.	35
3.5	This figure shows which parameters are selected for the time model when prefetching is enabled and their correlation coefficients. Each column represents a model with one more parameter included up to a maximum of 4 parameters. Y axis is R^2	36

3.6	This figure shows which parameters are selected for the time model when prefetching is disabled and their correlation coefficients. Each column represents a model with one more parameter included up to a maximum of 4 parameters. Y axis is R^2	37
3.7	Comparison of normalised runtime vs. CPU frequency for several SPEC CPU2000 benchmarks on the Core i7. 3.60 GHz represents a TurboBoost frequency, all values are normalised to this.	39
4.1	Runtime (left) and energy consumption (right) of <i>mcf</i> on the Sledgehammer and Santa Rosa systems.	44
4.2	Runtime (left) and energy consumption (right) of <i>mcf</i> on the Shanghai system. . .	45
4.3	Padded energy consumption for one, two and four instances of <i>mcf</i> on Shanghai (left) and padded energy-delay product (EDP) (right). All values are normalised to the maximum CPU frequency.	46
4.4	Idle power consumption for the Core i7 (top left), Atom Z550 (top right) and OMAP 4430 (bottom middle) when all cores are put in the same state.	49
4.5	Power draw of the Dell Vostro when only some of the cores of the Core i7 area placed in C6. When all cores enter C6, the processor enters the <i>package</i> C6 state, achieving even lower power draw.	50
4.6	Energy consumption for <i>mcf</i> (left) and <i>gzip</i> (right) when padded for idle energy with various C states for <i>mcf</i> on the Core i7. This approach assumes a single idle state transition at benchmark completion. Data shown is normalised to the minimum frequency. 3.60 GHz is a TurboBoost frequency. The zoomed section shows that using C6, energy efficiency can be improved by up to 5 % if the CPU frequency is reduced.	51
4.7	Costs associated with transition between C0 and deeper C states every 10 ms for two SPEC workloads, <i>mcf</i> (memory-bound, bottom) and <i>gzip</i> (CPU-bound, top) on the Core i7.	53
4.8	Energy consumption for <i>mcf</i> (left) and <i>gzip</i> (right) on the Core i7. In contrast to Figure 4.6, this data includes costs for idle state transitions every 10 ms. Data shown has been normalised to the minimum frequency. 3.60 GHz is a TurboBoost frequency. The zoomed section shows that using C6, energy efficiency can be improved by up to 5 % if the CPU frequency is reduced.	55
4.9	EDP for <i>mcf</i> (left) and <i>gzip</i> (right) on the Core i7 when slack time is introduced at 10 ms intervals. Data shown has been normalised to the minimum frequency. 3.60 GHz is a TurboBoost frequency.	56
4.10	System load (top left) and normalised energy consumption (top right) for playback of a high-definition MPEG video stream on the Core i7 under various frequencies. Energy data has been normalised to 2.13 GHz using C6, which was the case where minimum energy consumption was observed. The bottom graph shows a zoomed version of the C6 line in which the data has been normalised to the TurboBoost point.	57

4.11 Left: Energy consumption for playback of an MPEG video stream on the fitPC (Atom) when the C6 C state is used. Values are normalised to the 2.00 GHz / C6 point. Right: Energy consumption for playback of an MPEG video stream on the Pandaboard (OMAP4430). Values are normalised to the 1.008 GHz / C1 point.	58
4.12 Left: Energy consumption for playback of an MPEG video stream on the fitPC (Atom Z550) when the PowerVR VXD MPEG decode unit is leveraged. Right: Energy consumption for playback of an MPEG video stream on the Pandaboard (OMAP4430) when the IVA-HD MPEG decode unit is leveraged.	59
4.13 Topology of systems and network for the web-server benchmarks	60
4.14 Overhead in L2 and L3 misses and runtime for different C states for the Apache web-server workload on the Core i7 at 2.93 GHz. All values are normalised to the C0 C state.	61
4.15 Top left: CPU load on the Core i7 (all four cores) when running the Apache workload under DVFS. Top right: System-level energy consumption of the Core i7 running the Apache web-server workload. All energy values are normalised to the lowest observed energy consumption at 1.20 GHz / C6. Since Apache is a multi-threaded workload, the TurboBoost frequency varies between 2.93–3.60 GHz. Bottom centre: request efficiency in requests per unit energy, normalised to again to 1.20 GHz / C6.	62
4.16 System load (left) and energy consumption (right) of the fitPC (Atom) when running the Apache web-server workload. Energy consumption values have been normalised to the minimum value observed at 1.33 GHz with C6.	63
4.17 System load (left) and energy consumption (right) for the Pandaboard (OMAP4430) when running the Apache web-server workload. Energy consumption values have been normalised to the minimum value observed at 0.3 GHz.	64
5.1 Die area breakdown of three generations of Opteron processors.	71

List of Tables

3.1	Specifications of the two systems we analyse for the impact of memory throughput.	40
4.1	Specifications of the three AMD Opteron processors and systems we analyse for the compute-intensive workloads. All systems have a single CPU package.	43
4.2	Specifications of the three processors and systems we analyse for the idling workload scenarios.	47
4.3	Characteristics of the different C states on the three test systems. On the OMAP, C states greater than C1 could only be used if the second core was disabled. . . .	48

Chapter 1

Introduction and Motivation

Energy consumption has become a critical issue for most computing platforms. Previously, efficient use of energy resources was a concern only for battery-powered devices, such as mobile phones and laptops. However, it has now become a critical design constraint for *all* classes of systems. For example, a report to the California Energy Commission found that the power required for cooling and infrastructure in California data-centres can be over 50 % of the total electrical load [Tschudi et al., 2004].

A large proportion of the energy consumed by computer systems is spent powering the *central processing-unit* (CPU). In a data-centre, the heat generated by the CPU must be removed by air-conditioning requiring even more energy. To help to reduce CPU energy consumption, manufacturers have implemented various mechanisms for managing the power consumption of their processors. Some of these mechanisms allow the *operating system* (OS) to trade performance for lower power use, while others allow the OS to place the processor in various deep *sleep states* with increasingly reduced power consumption coupled with longer latencies to wake up from sleep.

One such mechanism, *dynamic voltage and frequency scaling* (DVFS), allows the OS to reduce the frequency (and performance) of the CPU and thereby lower its power consumption. Much prior research on improving the energy efficiency of computing systems has focused on DVFS, producing a number of techniques that can be employed by the OS [Weiser et al., 1994].

Data-centres contain thousands of servers which consume vast amounts of electricity for supply and cooling. However, studies performed by Google show that many of those servers spend most of the time significantly under-utilised [Barroso and Hölzle, 2007]. Hence, some studies suggest that reducing the CPU's frequency may improve energy efficiency.

Furthermore, as battery-constrained embedded systems continue to grow in performance and ubiquity, the issue of energy efficiency is becoming even more important. As a result, new power-management mechanisms, such as low-power idle modes and specialised hardware acceleration

units, are being introduced into embedded processors in order to help improve their energy efficiency and prolong battery life.

The energy efficiency of computer systems can be optimised using DVFS and sleep states. However, they are competing mechanisms. On the one hand, the CPU can be run at its highest frequency (with high power consumption) in order to complete work in the shortest time. Once the work is completed, the CPU can then go into a deep sleep state, reducing power consumption. This approach is referred to as *race-to-halt*. On the other hand, the CPU can be slowed down using DVFS, spend a longer time active, but have its power consumption reduced during this period. This approach reduces the time the processor can spend in deep sleep states.

To determine which of these approaches is best for a particular system, it is necessary to understand how the performance of the CPU's *workload* is going to respond to a change in CPU frequency. Prior research has addressed this issue by applying mathematical modelling techniques to try to predict how much performance will be lost when the CPU frequency is decreased [Snowdon, 2010]. These modelling techniques have also been used to predict the power drawn by a processor at different frequencies. When these two predictions are combined, the energy used by a processor executing some workload at different frequencies can be estimated and a decision can be made about which frequency should be used to provide optimal energy efficiency.

In the mid 1990s, Mark Weiser performed trace-based simulations of OS scheduler invocations in order to determine whether a CPU could be slowed down to reduce its energy consumption when the system was not fully utilised. His conclusion was:

To put it simply, the tortoise is more efficient than the hare: it is better to spread work out by reducing cycle time (and voltage) than to run the CPU at full speed for short bursts and then idle [Weiser et al., 1994].

More recently, Snowdon et al. and others, have shown that systems based on processors such as the Intel XScale PXA255 and Intel Pentium-M could achieve higher energy efficiency by reducing the CPU's frequency with DVFS when certain workloads were detected [Snowdon et al., 2009; Weissel and Bellosa, 2002].

However, our research suggests that several trends, such as:

- the scaling of silicon transistor feature sizes,
- increasing memory performance,
- improved sleep/idle modes,
- asynchronously clocked caches and memory-controllers, and
- the complexity of multi-core processors

are reducing the effectiveness of traditional power-management mechanisms such as DVFS in all classes of platforms.

This thesis shows that the above trends are indeed causing DVFS to become less effective and, in some cases, even reducing energy efficiency instead of improving it. However, the effectiveness of DVFS is critically workload dependent. Therefore, we aim to cover a wide range of workload and platform classes to determine if this finding is recurrent.

1.1 Research Questions

In short, we aim to answer the following questions:

1. Are Mark Weiser's claims set in stone—can energy efficiency still be improved by using DVFS to slow down the processor?
2. What factors influence the effectiveness of DVFS?
3. How much energy can be saved by using hardware acceleration units instead of the standard, general-purpose applications processor?

1.2 Contribution

Past studies have leveraged the DVFS mechanism to improve the energy efficiency of computer systems, however few have acknowledged the trends that contribute to its decreasing effectiveness. This thesis critically analyses the effectiveness of DVFS on recent computer systems, ranging from server- to mobile-class systems and shows that use of DVFS has become questionable. We introduce and analyse five trends which we believe are the root causes of the diminishing returns from DVFS.

1.3 Thesis Overview

This thesis is structured as follows:

Chapter 2 provides an overview of power-management mechanisms on modern processors and an analysis of prior work in the research area.

Chapter 3 discusses the impact of memory throughput on the effectiveness of DVFS and our attempt to model the performance of a recent multi-core processor under DVFS.

Chapter 4 addresses the question: *slow down or go to sleep?* by looking at several workload types on a range of platforms.

Chapter 5 outlines the trends that are contributing to the diminishing returns of DVFS; and

Chapter 6 suggests future directions for power-management research.

Chapter 2

Background and Related Work

2.1 Introduction

This chapter begins by briefly discussing basic transistor theory to enable the reader to better understand the analyses in the following chapters. Section 2.3 provides an overview of power-management mechanisms that are available on modern CPUs. Section 2.4 discusses how power-management mechanisms are used and evaluates several software techniques that make use of these mechanisms. Section 2.5 provides an overview of several frameworks that have been developed for OSes, including those that *Linux* currently uses to leverage the mechanisms available on modern CPUs.

2.2 Power Consumption Characteristics of Transistors

Transistors form the basis of all modern digital systems based on integrated circuits. Today, a computer processor can be made up of millions, or even billions of transistors.

Contemporary CPUs are implemented with a technology called *complementary metal-oxide semiconductor* (CMOS) which uses transistors called metal-oxide semiconductor field-effect transistors (MOSFETs). There are two types of MOSFET, n-channel and p-channel. Their construction is differentiated by using different types of semiconductor material for the transistor's features. MOSFETs are composed of two silicon junctions on a substrate, as shown in Figure 2.1. The *gate* (G) forms a capacitor which, when a voltage is applied, will charge. It is insulated from the substrate by an oxide layer, which means that very little current will flow through the gate. Upon charging, a conduction channel will form between the *source* (S) and the *drain* (D) due to electric fields induced in the substrate. If the voltage is above the *threshold voltage*, V_t , current can flow from source to drain.

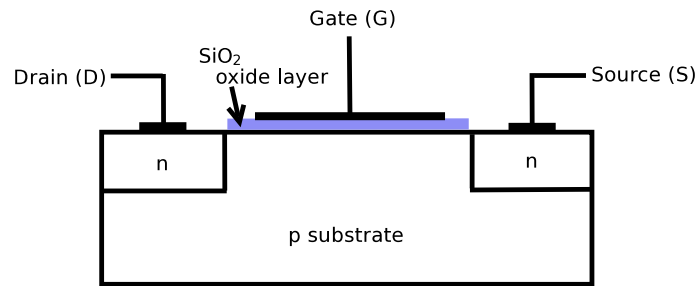


Figure 2.1: High-level components forming an n-channel MOSFET.
[see Floyd, 1984, Chapter 9, Section 9-3]

The power dissipation of CMOS integrated circuits is composed of both static and dynamic components. Static power dissipation arises from leakage and transistor bias currents, while dynamic power dissipation is due to the charge/discharge of capacitances and other switching activity in the circuit.

2.2.1 Transistor scaling

The size of the components in a transistor (e.g. the gate, source and drain), called the *feature size*, is determined by the manufacturing process. This is essentially a measure of the distance between the source and drain. In the early 1990's, the feature size was approximately $250\text{ }\mu\text{m}$. However, over time, processor manufacturers have worked to build smaller transistors because they can switch at a faster rate, and consume less power. Following *Moore's Law*, which dictates how often processors are released with a smaller feature size, this has lead to recent processors being manufactured from transistors with a feature-size of 32 nm . The *international technology roadmap for semiconductors* (ITRS) working group predicts that transistor feature-size will be as small as 9 nm within the next twelve years [ITRS, 2009].

Feature size is a critical characteristic for energy consumption for several reasons. Firstly, smaller transistors can operate at lower voltages and have reduced gate capacitance, which reduces dynamic power dissipation (explained in the next section). Secondly, as feature-size is scaled down, static leakage power *ris*es exponentially (explained in Section 2.2.3).

2.2.2 Dynamic power dissipation in CMOS circuits

CMOS logic gates, such as the NAND gate shown in Figure 2.2, are constructed from complementary pairs of n-channel and p-channel MOSFETs. The combination is such that in a steady state, one of the pair is in the off state, meaning that no current flows other than leakage current (explained in the next section). When one of the gate's inputs (A or B) switches, some time later, the output of the gate will settle on the result, and the next logic gate in the circuit may then transition based on its new inputs. A processor is made up of thousands of these logic gates which form functional units

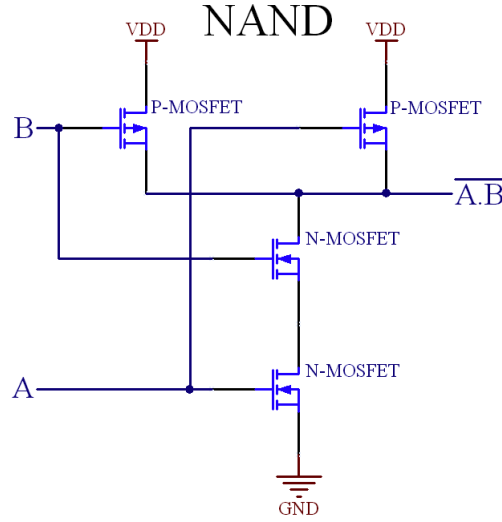


Figure 2.2: Construction of a NAND gate from MOSFET devices.
[see Mano and Kime, 1997, Chapter 2, Section 9]

such as integer or floating point units. These functional units are synchronised with a global clock, operating at the frequency of the processor.

The dynamic power drawn by a processor is made up of two components. Firstly, when switching occurs on a gate input, the two complementary MOSFETs are momentarily in a partial conduction mode, allowing *short-circuit* current to flow. Secondly, as the gates of MOSFETs form capacitors, there are energy costs associated with their charge and discharge due to

$$E = \frac{1}{2}CV^2 \quad (2.1)$$

where E is the energy stored in the capacitor, C is the capacitance in Farads, and V is the voltage that is applied to the FET's gate. As a transistor is scaled to a smaller feature size, the physical size of the gate is reduced, resulting in a reduction in its capacitance. Furthermore, as a smaller feature size allows reduced supply voltage, short-circuit current is also reduced. Hence, the total *dynamic* power drawn by a CMOS device is related to these characteristics:

$$P = \alpha C f V^2, \quad (2.2)$$

where P is the dynamic power consumption of the circuit (or processor), C is sum of the capacitance of the transistor gates (determined by the feature size), V is the supply voltage and f is the frequency at which the circuit is being clocked. The α value represents the level of switching activity in the circuit, because not all transistors will switch every clock cycle. For stable operation at a particular frequency, the supply voltage must be high enough such that the gate of each transistor in the circuit

has enough time to charge to above the threshold voltage. Hence, if the frequency is reduced, the voltage can also be reduced, resulting in a reduction in the power drawn.

2.2.3 Static power dissipation in CMOS circuits

Several factors contribute to the total static power consumption of a processor. Of these, the most substantial is *weak-inversion* (or sub-threshold) leakage current that flows from the drain to the source of each transistor, regardless of the state of the transistor. As feature size gets smaller, through the process of transistor scaling, the weak-inversion leakage current increases exponentially as

$$I_D \approx I_{D0} e^{\frac{V_{GS} - V_{th}}{nV_T}} \quad (2.3)$$

where I_D is the leakage current of a single transistor, V_{GS} is the gate–source voltage, I_{D0} is the leakage current when $V_{GS} = V_{th}$, n is a factor determined by feature-size and V_T is the *thermal voltage*, which depends on the temperature of the transistor.

Because of the exponential relationship between sub-threshold leakage and feature-size, in recent processors, static power dissipation contributes significantly to the total power consumption.

In the past, static power consumption has been small in comparison to dynamic power [Burd and Brodersen, 1995]. However, other factors contributing to leakage, including quantum effects such as gate-oxide layer tunnelling, are becoming increasingly important with shrinking feature size. Roy et al. discuss these and techniques for their mitigation [Roy et al., 2003].

Scaling of silicon transistor technology will slow at some point in the future due to the constraints of atomic dimensions. However, this will not happen before static power consumption (due to leakage current) rises well above dynamic switching power.

2.3 CPU Power-management Mechanisms

With the increasing importance of managed power consumption, modern CPUs have evolved to include numerous power-management mechanisms, some of which allow the OS to trade performance for reduced power consumption. Here, we describe the most common mechanisms and how they have been used in prior research.

2.3.1 Dynamic voltage and frequency scaling

DVFS is a mechanism that is available on most modern processors. It allows the core operating frequency of the CPU to be decreased allowing a corresponding reduction in the supply voltage, as per the relationship in Equation 2.2. This reduces the power drawn due to both dynamic and static components. Furthermore, temperature variation (due to varying heat dissipation or ambient temperature) can cause static power consumption to vary due to the dependence of sub-threshold leakage current on temperature as shown in Equation 2.3.

DVFS has the potential to not only reduce power draw, but also to improve energy efficiency. As will be explained in Chapter 3, some workloads respond differently to changes in CPU frequency. For some, performance scales linearly with frequency, while for others, performance scales sub-linearly and energy efficiency may improve with reduced CPU frequency.

A frequency change is not instantaneous—two steps are required, their order depending on whether the frequency is being increased or decreased. For stable operation, supply voltage must be increased to run at a higher frequency. Therefore, if a frequency increase is requested, the supply voltage must be *ramped* up to the value required to run at the requested frequency. After the supply voltage settles, the *phase-locked loop* (PLL) that generates the clock signal can be *relocked* to the new frequency. If a frequency decrease is requested, the PLL is relocked first, and then the supply voltage is ramped down. The combined latency for these steps is of the order of 10–100 μ s, depending mostly on the supply voltage change that is required.

This mechanism has been studied at length in the literature, and, in the past, its use has achieved significant improvements in energy efficiency, allowing battery-powered devices to last longer. Its use has been less pervasive in the data-centre mostly due to the perceived increase in latencies and reduction in *quality-of-service* (QoS) resulting from frequency changes.

Studies using DVFS are discussed in more detail in Section 2.4. In Chapter 3 and 4, we show that DVFS is becoming much less effective at improving energy efficiency. Indeed, we show that its use sometimes negatively impacts energy usage.

2.3.2 Idle states (C states)

A CPU can be thought of in terms of a *package*, *cores* and *threads*. A package can consist of multiple cores and a core can have one or two thread *contexts* which Intel has called *HyperThreads*.

When the OS has no tasks to schedule on a core, it will usually invoke the *idle thread* and parts of processor can be put into an *idle state* (or C state). As this thread does no useful work, it makes sense to put the CPU to sleep.

Modern CPUs tend to offer multiple C states, denoted by C_x where x is a number from 0 to some maximum. Higher x values yield a *deeper* idle state, resulting in lower power consumption, but

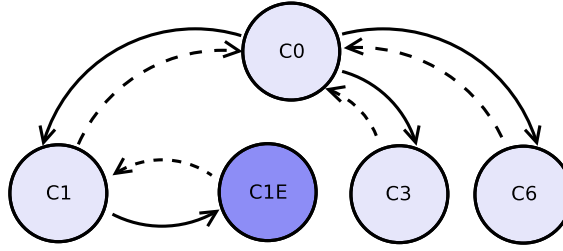


Figure 2.3: Possible C state transitions for an Intel Core i7 processor. All transitions must go through C0, except to the special C1E state, entered when all cores enter C1.

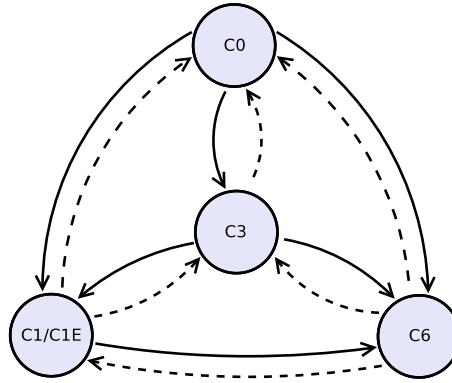


Figure 2.4: Future generations of Intel processors will have automatic promotion/demotion of idle states, thus removing the requirement to transition through C0, minimising the number of wake-ups. Current generations are restricted to the state transitions shown in Figure 2.3.

higher entry and exit latencies. Intel defines C states at the thread, core and package level, with C0 being the operating state in which a core is executing instructions. However, other manufacturers do not provide specifications for C states, allowing OS designers to choose the steps taken when entering different C states.

Because a single processor can have multiple cores and multiple hardware thread contexts (Hyper-Threads), constraints exist between thread, core and package idle states, for example, a *package* C state will not be entered unless all *cores* enter the same C state. There are also constraints on state transitions, as shown in Figure 2.3 for the Core i7 processor, where all transitions between C states must go through the active state, C0. The C1E state is a special exception, which is entered automatically when all cores enter C1.

Some C states are entered automatically, by the processor *promoting* an OS request to transition from a particular C state to a deeper C state. This includes the C1E state on Intel processors, and the C5 state on the Atom Z550, which are described in Section 4.3.2.

Future generations of Intel processors will also include a feature called C state *auto-demotion* where, depending on C state residency history (recorded by the processor), a request by the OS to enter an idle state will be automatically demoted to a lower level (lighter sleep) to minimise the negative effects of high-frequency C state transitions, resulting from naïve use. This is shown in Figure 2.4.

C states are discussed in greater detail in Chapter 4.

2.3.3 Intel's power-management unit and TurboBoost

Intel's power-management unit, not to be confused with the performance-monitoring unit, was introduced with the Nehalem architecture in 2008. This unit monitors the temperature and power requirements of the processor and determines when TurboBoost (described below) is initiated and when idle state requests can be promoted and should be demoted. This requires that the processor be instrumented with current, voltage and temperature sensors at various points on the die. Unfortunately, the measurements taken using these sensors are not accessible by the OS. Snowdon and Esmailzadeh et al. suggest that manufacturers should expose these measurements to the OS. This way, software developers can benefit from the knowledge and insight that they provide about how energy is consumed in computer systems [Snowdon, 2010; Esmailzadeh et al., 2011].

Recent Intel processors based on both the Nehalem and Sandy Bridge micro-architectures have a feature called *TurboBoost*. If one or more of a processors' cores are powered down into a deep idle state, the clock frequency of other active cores can be boosted above the rated nominal frequency. For example, the Intel Core i7 870 has a nominal rated frequency of 2.93 GHz. However, if one or more cores are sleeping, the frequency of the remaining active cores can be increased. If multiple cores are active, their frequencies may be increased to up to 3.2 GHz, depending on power draw and thermal load. If only one core is active, its frequency can be increased to 3.6 GHz. This can give performance improvements for single-threaded workloads.

Charles et al. perform an analysis of this feature using SPEC CPU2006 on a 3.2 GHz Core i7-based system and find that TurboBoost provides, on average, a 6 % improvement in performance [Charles et al., 2009]. However, they predict that this would be coupled with an average 16 % increase in energy consumption by the CPU. This increase in energy consumption is predicted using a simple model for the *dynamic* power consumption of their processor based on Equation 2.2, hence it does not account for energy consumed because of static leakage power in the processor or energy consumed by the rest of the system.

Unfortunately, the TurboBoost feature is controlled entirely in hardware by the power-management unit. The OS has only the ability to enable or disable it. Scheduling policies could be developed to allow single-threaded workloads to take advantage of TurboBoost at the cost of forcing other cores to sleep, however this is not the focus of this thesis.

2.3.4 Dynamic cache resizing

Caches contribute a large proportion of total processor power, as they require vast amounts of transistors and processor die area, adding to static power draw. To address this issue, some processors implement a dynamically resizable *last-level cache* (LLC). Using this mechanism, static

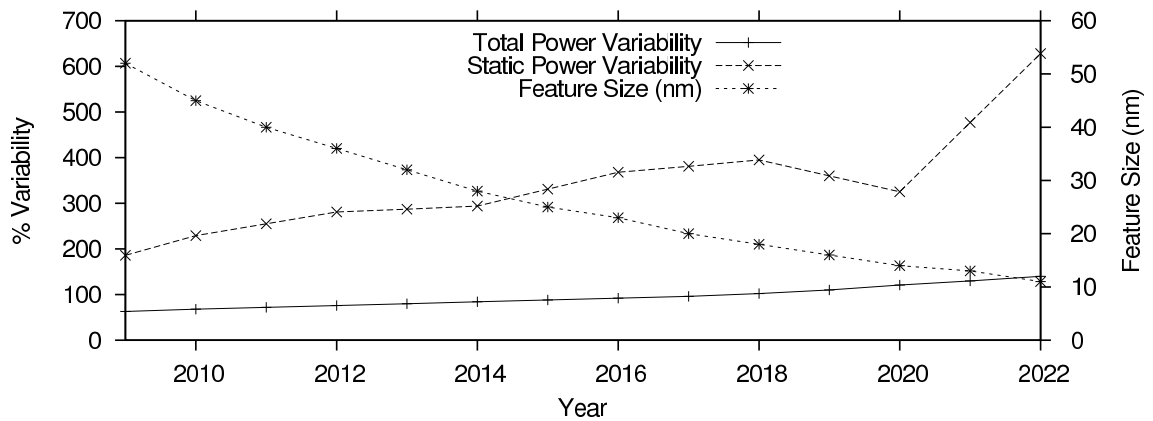


Figure 2.5: Variability in semiconductor power consumption, as predicted by the ITRS. [ITRS, 2009]

leakage power can be reduced when the entire cache is not required. Intel implements this feature with some of its mobile processors, such as the Atom.

Control of dynamic cache resizing is handled by the processor, however the OS can influence some decisions by changing thresholds, such as the CPU frequency below which cache resizing will be applied. Additionally, the OS can give *hints* to the processor when entering deep idle states suggesting that the cache should be resized.

2.3.5 Adaptive voltage scaling

The manufacturing techniques used to create processors cause variability in the transistor threshold voltages across supposedly identical devices. For this reason, at manufacture time, processors are tested to determine the voltage required to run at their rated frequency. The ITRS predicts that variability in static power consumption will increase dramatically over the next decade as feature sizes become smaller, as shown in Figure 2.5. Wanner et al. perform a study to measure how sleep (static) power varies in embedded processors which are supposed to be identical parts. They find that the variation in static power at room temperature can be as much as 500% [Wanner et al., 2010].

At run time, temperature variation (due to power dissipation and external effects) causes the threshold voltage of the transistors to change. Adaptive voltage scaling (AVS) tunes the supply voltage to improve energy efficiency under this temperature variation. The OMAP 4430 processor (which is analysed in Chapter 4) implements AVS by the way of its SmartReflex mechanism [Brian Carlson and Bill Gjolma, 2008]. However, like TurboBoost, the OS has little control over this feature. As a result we have not analysed it in detail.

2.3.6 Discussion

Manufacturers have implemented numerous mechanisms both to reduce processor power consumption when compute resources are not needed and allow to OS developers to design algorithms to trade power for performance. However, it is not immediately obvious that using these mechanisms will improve energy efficiency. Therefore, analysis should be undertaken before they are used for a practical purpose. This thesis focuses our analysis on DVFS and idle state usage as these mechanisms create the most interesting dynamic trade-offs. The other mechanisms discussed are usually simply enabled or disabled.

2.4 Evaluating Energy-management Techniques

OS developers are able to interact with the low-level hardware and have the ability to use the power-management mechanisms that processor manufacturers implement. In order to evaluate the effectiveness of a software power-management technique, the system being tested needs to be instrumented or modeled in some way. Because computers are complex systems, it is often difficult to analyse exactly where and how energy is used. This section discusses some methods that have been developed to use the available power-management mechanisms described above and evaluates their effectiveness.

2.4.1 Measurement and run-time profiling

Power consumption can be measured at various points in the system, resulting in measurements with different meanings. For example, the power lines leading directly into the CPU package can be instrumented, giving the power consumed only by the CPU [Esmaeilzadeh et al., 2011]. Alternatively, power consumption can be measured at the power supply, giving the total power consumed by the system and all of its peripherals.

Flinn and Satyanarayanan developed *PowerScope*, a tool to provide feedback to the developer about the energy consumption characteristics of their code [Flinn and Satyanarayanan, 1999]. The *system under test* (SuT) is instrumented with power-measurement apparatus and a separate system is used to record the data. Each time the measurements are sampled, the SuT is interrupted and the interrupt handler routine records the program counter and details of the currently running process. The data can be analysed at a later time to determine the energy efficiency of specific code sequences.

Most modern CPUs have a *performance-monitoring unit* (PMU), which enables online measurement of often hundreds of different events. The PMU consists of several configurable registers which count architectural events, such as cache-misses, unhalted CPU cycles, dispatch stalls, etc. These *performance-counter events* provide information about the performance of code running on the

CPU, helping to fine-tune and debug programs. They can also be used as parameters in models for estimation of power consumption and execution time.

Isci and Martonosi instrumented a Pentium-4-based system with a power meter and used the processor's PMU to determine the power consumed by individual functional-units within the processor [Isci and Martonosi, 2003]. The Pentium 4's *Netburst* architecture provides 18 hardware performance-counter registers allowing simultaneous measurement of up to 18 events. In contrast, processors based on Intel's more recent Pentium M, Core and Nehalem architectures can measure only two events simultaneously. Isci and Martonosi validate their model using SPEC CPU2000 benchmarks and report an average absolute error of 2 W, and a maximum error of 5.8 W. After interpreting the figures, this would seem to be about 4 % and 12 % error respectively. They also perform several real-world desktop workload benchmarks such as browsing web-pages and using a spreadsheet, but do not report specific results from these.

Measurement at run time is a concrete method to ensure that accurate data is obtained, however the situations in which most systems are deployed limit the feasibility of this approach. Profiling is useful to obtain accurate results while software systems are in development and testing phases, but when they are deployed on systems that differ from those used during testing, accurate results cannot be ensured. Processor manufacturers are starting to include power-measurement apparatus on the processor die, for example Intel's power-management unit includes current, temperature and voltage measurement sensors. However, at this point in time, these measurements are not made available to the user. If designing custom hardware, as is often the case for embedded systems, including power-measurement sensors is not difficult. The OpenMoko Freerunner, analysed by Carroll and Heiser, has component locations for sense-resistors on the main PCB, allowing individual sub-components, such as CPU and memory to be measured individually [Carroll and Heiser, 2010]. However, to achieve this, external data acquisition hardware was required. The *PLEB2*, a custom-designed, education oriented, embedded system included a separate micro-controller and sense-resistors to allow power measurement of several subsystems at run time and thereby enable energy-use profiling and online optimisation [Snowdon and Johnson, 2003].

2.4.2 Simulation and trace analysis

Simulation is a technique where a model of a system is developed and used to determine how a real system would react to changes in certain characteristics. Often to achieve this, a *trace* of a real system is recorded and then used (or *played back*) within the simulator to ensure that it remains coupled to reality. This technique can also allow comparison to ideal situations in which the real system lacks some information that a trace-based simulation can access (usually because that information is the result of a future decision). However, this can lead to the development of systems that rely too much on unrealistic, ideal models.

Brooks et al. modify *Turandot*, a cycle-accurate simulator for IBM POWER 4 processors to integrate *PowerTimer*, which provides a model for the power consumption of low-level functional units within

the CPU [Brooks et al., 2003]. They use the simulator and models to predict the effects of various architectural modifications on energy efficiency, for example, they find that when increasing the size of the L1 data cache, the performance improvements from lower miss rates are shadowed by a large increase in power consumption. Simulation at this level can be used to make informed decisions about which architectural features are important when designing new processors.

Isci et al. use Brooks' PowerTimer framework to analyse the effects of global versus per-core DVFS on a *chip multi-processor* (CMP) for SPEC CPU workloads. They found that with a global view of the tasks running on a CMP, they could better constrain the power consumption of the chip as a whole, while maintaining performance at a suitable level [Isci et al., 2006].

In 1994, Weiser et al. introduced the idea of varying the CPU frequency using DVFS based on the system load to optimise energy efficiency [Weiser et al., 1994]. He used a trace-based simulation to analyse the effects of changing CPU frequency at OS scheduler invocations on system energy consumption. A variant of this technique is now widely used, including in mainstream Linux which we described in Section 2.5.2.

Grunwald et al. performed an analysis technique developed by Weiser et al. to determine if it could be applied to real devices and workloads [Grunwald et al., 2000]. They used workloads such as MPEG playback and serving web-pages on a device with a Strong ARM SA-1100 processor. However, they were disappointed by the lack of energy savings that DVFS could realise in the realistic scenarios they tested. They also showed that the algorithms used by Weiser et al. were impractical because they require future knowledge only available during the trace-based simulation run.

Lorch and Smith describe a methodology called *PACE* designed to optimise such algorithms for real-time tasks with soft deadlines [Lorch and Smith, 2001]. *PACE* predicts the probability distribution of a task's work requirement (i.e. the number of cycles it will take to execute) by looking at past instantiations of the same task. It can then estimate a frequency that will minimise energy consumption while keeping the execution to the real-time schedule.

Simulation can allow theories to be tested, usually without significant implementation overhead. For example, simulating the effects of increasing L1 cache size before manufacturing a new processor design. However, care must be taken to understand the limitations of simulation, for example, during a simulation, the system being developed can use information that it would otherwise not have a priori, such as knowing how long a task is going to take to execute. Furthermore, simulators often make use of simplified system models in order to limit implementation complexity. This makes them less accurate than testing on real systems. For example, Snowden found that the efficiency of the CPU's voltage regulator on a Dell laptop changed when it switched between modes depending on CPU power requirements [Snowdon, 2010]. Without looking at the real system, the simulator would be unable to account for this change in efficiency.

For these reasons, we find that actual measurements of real systems give more reliable results for

our analyses. The use of actual measurements ensures that all the idiosyncrasies of the system are accounted for when measuring total system power at the *power-supply unit* (PSU).

2.4.3 Modelling and accounting

Modelling is a technique where some measurable characteristics of a system can be used to predict some other unmeasurable characteristic. In the scope of this thesis, modelling power consumption has been attempted numerous times in the literature, usually because computer systems do not have the ability to measure their own power consumption.

2.4.3.1 Modelling power consumption

Bellosa was the first to propose using performance-counter events to model processor power consumption [Bellosa, 2000]. Bellosa showed that a relatively simple linear model can accurately predict the power drawn by a processor when executing a piece of code. For the *process cruise-control* framework (described in Section 2.5.1) Weissel and Bellosa hand-picked several performance-counter events for use in their models.

Pusukuri et al. show that simple models for power consumption can be developed using a small number of performance-counter events as predictors [Pusukuri et al., 2009]. Their models are shown to be robust over a range of workloads from the SPEC CPU 2006 suite, however they find that some benchmarks have a high prediction error. Their model is frequency agnostic, however by using the SPEC CPU suites, they limit the generality of the model because the benchmarks are single-threaded and do not exercise a multi-core system to its' full extent. This is shown in the validation where the SPECjbb 2005 benchmark (a multi-threaded benchmark) has a high prediction error.

Snowdon et al.'s early work looked at several simple embedded systems such as the PLEB2, based on the Intel XScale PXA255 processor [Snowdon and Johnson, 2003; Snowdon et al., 2007]. This is the same processor used in Weissel and Bellosa's *process cruise-control* framework described in Section 2.5.1. The models developed for this simple processor were highly accurate and relatively easy to characterise. This is because the PXA255 has only 15 different performance-counter events which may be measured. Furthermore, the PXA255 lacks complex architectural features such as hardware prefetching [see Intel Corporation, 2003, Section 6.1.1, page 55]. Complex features, as we will show, can cause effects that are difficult to predict from measurements of performance-counter events.

2.4.3.2 Modelling energy consumption

For power-management, an important aspect of modelling *energy* (as opposed to just *power consumption*) is predicting the **execution time** of a piece of code at various CPU frequencies, since:

$$Energy = Time \times Power \quad (2.4)$$

The execution time of a piece of code, T , can be broken down into the number of cycles spent in various frequency domains:

$$T = \frac{C_1}{f_1} + \frac{C_2}{f_2} + \frac{C_3}{f_3} \dots \quad (2.5)$$

These frequency domains (f_1 , f_2 and f_3) may be made up of the CPU itself, the L3 cache which runs at a different frequency and main memory which operates at yet another frequency. Each component in Equation 2.5 has a coefficient that represents the amount of time that the system spends waiting for that component. An example of such a topology is the AMD *Shanghai* Opteron processor, where the CPU cores, L3 cache and main-memory all operate at different frequencies.

Equation 2.5 then becomes:

$$T = \frac{C_{CPU}}{f_{CPU}} + \frac{C_{L3}}{f_{L3}} + \frac{C_{mem}}{f_{mem}} + \frac{C_{coherency}}{f_{coherency.link}} \quad (2.6)$$

The Opteron architecture is also an example of a *cache-coherent non-uniform memory access* (cc-NUMA) system with multiple CPU packages, each connected to its own bank of memory. The *nodes* are connected over a high-speed bus, such as *hyper-transport* (HT) as is the case with the AMD Opteron system. For workloads which share data, cache coherency traffic between nodes can be substantial and as a result the model must accommodate for the CPU cycles spent waiting for the coherency frequency domain.

Using the PMU, we can estimate the number of cycles spent in each frequency domain using a model such as

$$\begin{aligned} C_{L3} &= \alpha L2.cache_misses \\ C_{mem} &= \beta L3.cache_misses \\ C_{coherency} &= \gamma Remote_requests \end{aligned} \quad (2.7)$$

where α , β and γ are platform-dependent coefficients determined through a process called *model characterisation*.

Modern processors have a *cycle counter* which can be used to measure total CPU cycles during execution, C_{total} . In that case, Equation 2.6 can be rewritten as:

$$C_{CPU} = C_{total} - \frac{f_{CPU}}{f_{L3}} C_{L3} - \frac{f_{CPU}}{f_{mem}} C_{mem} - \frac{f_{CPU}}{f_{coherency}} C_{coherency} \quad (2.8)$$

The model in Equation 2.8 contains what may seem like obvious parameters (from Equation 2.7). However, in reality, parameter selection is not so simple.

Rajamani et al. develop a model for predicting the performance impact of frequency changes on a single-core Pentium M system [Rajamani et al., 2006]. They use a small set of synthetic benchmarks to characterise their model for execution time. They then validate their model with benchmarks from the SPEC CPU 2000 suite showing that it has, on average, 10% prediction error. Like Bellosa, they hand-pick the performance-counter events to use as parameters in their models.

Snowdon improved on Bellosa's and Rajamani's approach by developing a technique to systematically analyse and choose the best parameters for a model from the pool of potentially hundreds of performance-counter events available on recent processors. The outcome is the characterisation of two models, allowing for prediction of execution time and power consumption under varying CPU frequency.

Snowdon's approach has three stages:

1. choose a characterisation benchmark set and measure the system exhaustively;
2. perform a regression subset-selection to choose performance-counter events and their coefficients; and
3. evaluate (validate) the generated models on a different set of workloads.

The first stage involves running each benchmark workload at each available CPU frequency while measuring each of the available performance counter events, as well as the energy consumed by the system during each benchmark run. This requires running multiple iterations, not only for statistical reasons, but also because most recent CPUs can only measure two to four events at once.

The second stage is to use a computational statistics package such as *R* to perform a *regression subset-selection* [R, 2011]. This involves trying each of the performance-counter events measured in stage one in a model to see which ones provide the highest accuracy. This can be time-consuming, depending linearly on the number of benchmarks in the set, the number of frequencies and the maximum number of parameters to try in each model.

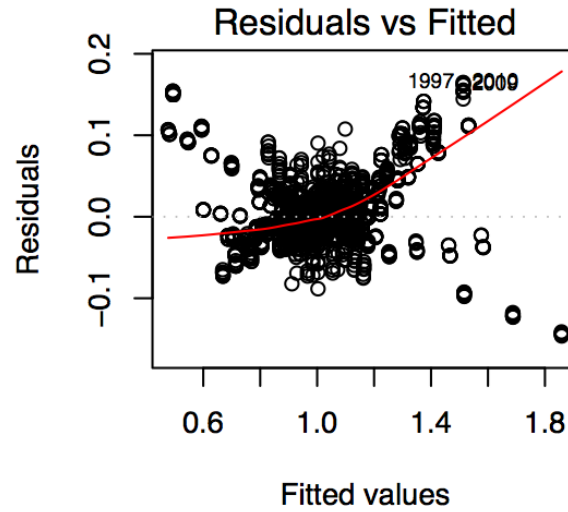


Figure 2.6: Plot of model residuals against measured value for an execution time model with four parameters [R, 2011].

Since only a small number of parameters can be measured at once, there are also constraints between the two models. If the execution time model uses four parameters then the power model must contain a subset of those four. Snowdon characterised the execution time model first and then used the parameters chosen in a subset-selection for the power model.

Part of the second stage is to visually inspect the residual plots to determine how the model is performing. Figure 2.6 shows a graph plotting residuals versus measured values. The value being predicted is actually the ratio of CPU cycles at the current frequency and CPU cycles at the target frequency (the frequency we are predicting)—this is shown on the x axis. The y axis shows the residuals, or error, in each prediction. Each circle on the graph represents an iteration of a benchmark workload being run at one frequency and the ratio of cycles being predicted at another possible frequency. From the graph, the maximum percentage error seen is approximately 10% ($0.2/1.8$).

The third stage involves validating the models generated in stage two. A different set of *validation* benchmarks are chosen (these should be different to those used during the characterisation) and the models are used to predict the energy consumed at run-time. The actual energy consumed is still measured using a power-meter. The predicted energy is compared to the measured energy to determine the accuracy of the models. See Chapter 3 for an example of such a validation.

Belloso pioneered the performance-counter based modelling approach, which Snowdon improved on. Several others have also used performance-counters to model both execution time and power consumption under varying frequency, resulting in reasonably accurate predictions for older, single-core processors. Our own work attempts to extend this approach to modern, multi-core processors.

2.4.4 System-level analysis

The work most closely related to our own was done by Miyoshi et al. on a laptop-class system [Miyoshi et al., 2002]. They showed that the decrease in slack-time resulting from running at a lower CPU frequency can offset any energy savings realised by using DVFS. Using a web-server workload they found that it was more energy-efficient to run at a high frequency (i.e. race-to-halt) for both high-utilisation and low-utilisation scenarios. However, the kind of systems available 9 years ago (when their study was performed) are very different to those available today. As discussed in Section 2.3.2, modern CPUs offer many low-power idle states and their usage results in significantly reduced power consumption.

Bircher et al. use performance-counters to develop models of an entire system, as opposed to just the CPU. Their technique uses the relationship between certain CPU performance-counter events and other activity in the system to attribute power consumed by subsystems such as the hard-drive, chipset and memory [Bircher and John, 2007]. They characterise their models with SPEC CPU benchmarks, a Java benchmark and a synthetic disk benchmark. The models they developed were shown to have an average error of less than 9%.

More recently, Bircher et al. have looked at DVFS on CMP systems such as those based on AMD's Opteron processors [Bircher and John, 2008]. Bircher et al. find that reducing the frequency of a single core has a negative effect on the performance of other cores operating at higher frequencies. This was found to be due to the Opteron's shared cache architecture. Cache coherency probes to cores that are running at low frequencies take longer, and as a result, workloads that share memory are affected by DVFS decisions in ways that older systems were not.

Carroll and Heiser analyse the power consumption of several smart-phones, firstly by fine-grained instrumentation of an OpenMoko Freerunner, an open-source mobile hardware platform [Carroll and Heiser, 2010]. They find that most power is consumed by components *other* than the CPU, such as radios and the LCD back-light. These findings are validated by measuring the total power consumption of two more recent smart-phones, the Google G1 (HTC Dream) and Google Nexus 1 (HTC Desire).

System-level analysis provides a complete picture of the power consumption characteristics of a computer system. It gives an indication of which components are the most *power hungry*, and therefore, suggests the best places to look for opportunities to improve energy efficiency. Miyoshi et al. perform a similar study to our own on a much older system [Miyoshi et al., 2002]. However, power-management and computer architecture have moved on since then. Carroll and Heiser's recent study finds that on embedded systems, the processor consumes a small proportion of total power relative to other components, such as LCD back-lights and wireless network radios [Carroll and Heiser, 2010].

Our work looks at the effectiveness of DVFS on more recent systems which have numerous low-

power idle states. In the past, processors either had no idle states or, only a single idle state with relatively high power consumption when compared to the processors of today.

2.4.5 Scheduling on multi-core systems

Systems based on multi-core processors present opportunities to choose tasks to *co-schedule*. Merkel and Bellosa study the effects of co-scheduling tasks with different memory-intensity. This is achieved by sorting (by memory-intensity) the run-queues that the OS scheduler maintains for tasks executing on the system. They find that DVFS only saves energy when memory-bound tasks are co-scheduled together [Merkel and Bellosa, 2008]. However, in doing this, the performance of these tasks is severely impacted. Their conclusion is that the optimal scheduling strategy for maximum energy efficiency is to co-schedule memory-intensive tasks together with compute-intensive tasks. This allows the memory-intensive task full use of the memory hierarchy, with minimal impact on the performance of the compute-intensive task, which has a memory-footprint that fits inside the core-local cache.

Asymmetric multi-processor (AMP) systems were proposed as a more power-efficient way to design multi-core processors [Kumar et al., 2003, 2004]. Currently, multi-core processors include many cores of the same *instruction-set architecture* (ISA) and design, operating at the same frequency (i.e. *symmetric multi-processor* (SMP) systems). An AMP may contain cores which are *fast* or *slow*. Fast cores are characterised by deep pipelines and high issue width, out-of-order execution, aggressive branch prediction and high frequency. *Slow* cores, on the other hand, may have shallower pipelines, be in-order, single-issue, etc. The power-performance trade-offs differ greatly between these two core types. Certain workload types are better suited to slow cores, while others are more suited to fast cores. Because workloads may exhibit both compute-intensive and memory-intensive phases, the decision about which core to run on cannot simply be made at the beginning of program execution. Rather, the program must be monitored at run-time and a decision made (by the OS scheduler) about whether to *migrate* a task between cores.

Saez et al. developed a comprehensive scheduler for AMP systems [Saez et al., 2010]. Using hand-picked combinations of SPEC CPU workloads, they established a model which can predict the *speed-up factor* of running a task on a fast core compared to running on a slow core. Their model is shown to be accurate over a wide range of workload types, including those that are multi-threaded. On their test system, fast cores are characterised only by a high frequency. There are many different ways in which fast and slow cores can vary, such as those described above. They conclude that, regardless of the characteristics of the cores, AMP systems are a viable alternative to SMP systems.

As there are currently no commercial AMP systems available, they must be emulated. This can be achieved in various ways. Many have used DVFS to create performance asymmetry. Li et al. also study scheduling on AMP systems. However, instead of using DVFS, they create performance asymmetry by controlling the issue width on an x86 processor [Li et al., 2007]. This is done by putting one or more cores of a CMP into a special '*debug*' mode. They present *AMPS* an OS scheduler

that supports architectures with both SMP and NUMA style performance asymmetry. Using a range of workloads (SPEC OMP, SPECjbb, kernbench and Ogg Vorbis encoding) they show that AMPS is capable of improving performance by up to 44 % compared to the standard Linux 2.6.16 scheduler for some workloads. The focus of this paper is on improving performance rather than energy efficiency.

Scheduling presents many interesting opportunities for power management on multi-core systems. However, constraints between cores, such as operating frequency and shared resources limit the benefits achieved from mechanisms such as DVFS as Merkel and Bellosa show.

AMP systems are an interesting new direction for processor architecture and bring different trade-offs to power-management. Work in this area is mostly unrelated to our own, however our work could influence the design decisions made in these new processors, for example, whether to implement per-core DVFS or provide only chip-wide DVFS.

2.4.6 Discussion

Researchers and OS developers have used power-management mechanisms in numerous ways to achieve many different goals. The primary purpose of their work is to improve the energy efficiency of computer systems, allowing increased longevity for battery-powered devices and reduced power/cooling bills for data-centres. Much of the prior work in this area has used DVFS, however, few have analysed how its effectiveness has changed over time.

2.5 OS Energy Management

The OS plays a critical role in power management. Many previous researchers have developed novel ways to leverage power-management mechanisms in order to improve energy efficiency or to attribute energy consumption to particular system activities. Power management is an active area of research in academia and in the industry. However, while academia thrives on publication, industry is often secretive about techniques that give a competitive advantage. Battery life is a critical driver for power-management research in the industry, and, often the results of such research are not published. However, the power-management frameworks used in *Linux* are open-source. Firstly, we discuss several research frameworks, and secondly, we discuss those within *Linux*.

2.5.1 Research frameworks

Researchers have studied OS power-management for many years, resulting in the development of many frameworks. The primary goals of these frameworks is to attribute energy consumption to tasks running on a system and allow intelligent power-management decisions to be made.

For example, Weissel and Bellosa developed *process cruise-control*, a framework that is based on modelling the response of workloads to changes in CPU frequency [Weissel and Bellosa, 2002]. This was achieved by correlating hardware performance-counter events (measured by the PMU) with performance and energy consumption measurements. Performance-counter measurements are recorded at run-time for each process running in the system. The expected performance loss is then calculated for each frequency the processor can run at, and a frequency is chosen that will give no more than 10 % performance degradation. This system was able to achieve energy savings of up to 22 % for memory-intensive applications when frequency *alone* was scaled. It was further claimed that their system would achieve savings at least 37 % if the voltage was scaled as well.

More recently, the *Koala* framework, which builds on the ideas pioneered in process cruise-control, was developed by Snowdon et al. [Snowdon et al., 2009]. Snowdon surmised that in order to make intelligent power management decisions, one must first know how the system will respond. Using the techniques described in Section 2.4.3, two mathematical models (one for execution time and one for power consumption) were used to predict the energy consumed by a workload at various processor frequencies when using DVFS. At run time, OS scheduler invocations were used as points to run the models, and data was collected from the performance-counters during each time-quantum. A policy framework was constructed around the models allowing the system to respond to different user requirements. The *generalised energy-delay* policy is represented by

$$\eta = P^{1-\alpha} \times T^{1+\alpha} \quad (2.9)$$

where η is the value to be minimised, P is power and T is time. This gives the user a single variable parameter, α , which has a range of $-1 \leq \alpha \leq +1$. The range is continuous, however some values result in the more well-known policies:

$\alpha = 1$ gives maximum performance (no reductions in CPU frequency)

$\alpha = 0.33$ gives minimum energy-delay product (maximises MIPS/W, i.e. computational efficiency)

$\alpha = 0$ gives minimum energy

$\alpha = -1$ gives minimum power (i.e. the lowest frequency)

At each time-quantum, this policy was used to determine the frequency at which a task should run. By using the OS's process model, each task is treated separately, allowing a different frequency to be chosen for each workload.

High-level policies can also use the *generalised energy-delay* policy. For example, the system may be battery powered, and as the battery charge depletes, longevity may become *more* important, so α may be lowered to reduce energy consumption.

The *Koala* framework was shown to improve the energy efficiency of some workloads by up to 24 %, while having little impact on performance [Snowdon et al., 2009].

However, the *Koala* framework focused only on the CPU and memory. Zeng et al. developed *ECOSystem*, a framework that attempts to fairly share energy resources amongst tasks running on a system [Zeng et al., 2003; Zeng and Lebeck, 2005; Zeng et al., 2002]. Zeng et al. introduce the idea of *currentcy*, which represents energy. Currentcy is allocated to tasks when they begin, and continues to accumulate over time. Currentcy is consumed when a task runs on the processor, or has the OS perform other tasks on its behalf, such as disk I/O. If a task runs out of currentcy, it may no longer be scheduled on the CPU. A task's currentcy will continue to accumulate over time, allowing it to be scheduled again once enough has been accumulated. The CPU model they use is simplified by assuming that the CPU consumes a fixed amount of power when doing work.

Cinder is similar to *ECOSystem*, in that it allocates energy resources to tasks [Roy et al., 2011]. However, *Cinder* achieves this by using the separation that a microkernel provides. *Reserves* are stores of energy, and *taps* control the rate of energy use. The battery is the root reserve, from which all energy is sourced. *Cinder* supports:

- *isolation*, which forces tasks to share the energy resources fairly;
- *delegation*, which allows a task to specify some portion of its allocated energy resources to a system-wide task, such as a network stack; and
- *subdivision*, which allows one task to share some of its energy resources with another.

The authors claim that the ARM processor used in their test system (an HTC Dream) lacks the ability to measure performance events. For this reason, they assume that tasks executing on the processor consume a fixed amount of power. This sort of framework is useful when tasks need to activate a costly resource, such as a network interface. On a mobile phone, many tasks may want to perform network operations, but alone, a task may not have enough allocated energy to activate the costly wireless radio needed to perform network operations. However, if multiple tasks pool their energy resources by collaborating (using *Cinder's delegation* mechanism), the cost of activating the radio can be amortised over several tasks, thereby limiting the rate at which the radio is activated and improving energy efficiency.

2.5.2 Linux

Linux is unique from the well-known commercial OSes by being completely *open-source*. This makes Linux an obvious choice for research purposes, because it is easy to alter and understand.

The kernel sources for other OSes such as Windows and Mac OS X are mostly unavailable. Even when source-code is provided, such as with Apple's *Darwin* (XNU) kernel, the code which implements the

power-management control is excluded. Presumably, this is because the creators believe that the technology used is proprietary and keeping it secret gives them a competitive advantage.

The Linux kernel has two major subsystems which control power-management, *cpufreq* and *cpuidle*, which respectively control DVFS and idle mode decisions.

2.5.2.1 The cpufreq framework

The *cpufreq* framework includes algorithms that decide what frequency the CPUs cores should operate at, based on some heuristics measured by the system.

Several *governors* can be loaded into cpufreq. These governors apply a policy for choosing an appropriate CPU frequency.

The *ondemand* governor monitors the load on the CPU, and reduces the CPU frequency by one step if the load falls below a certain threshold. Conversely, if load rises above a pre-defined threshold it will immediately increase the frequency to the maximum.

The governor checks the load every 80 ms by default thereby allowing the CPU frequency to track the system load closely, hence the name *on demand*.

The *conservative* governor is similar to the *ondemand* governor, however, when load is detected above the threshold, it will only increase the frequency by one step, rather than immediately to the maximum.

There are also two more simplistic governors, *performance* and *powersave*, which respectively hold the CPU at the highest and lowest frequencies.

Lastly, there is a *userspace* governor which allows the user to specify which frequency each core of the CPU should operate at. This also allows a userspace application to implement some other arbitrary policy to control frequency changes.

2.5.2.2 The cpuidle framework

The *cpuidle* framework uses algorithms to decide which depth of sleep the processor should be put in when there are no tasks to schedule.

Like cpufreq, there are several governors which can be loaded into cpuidle which apply different policies to choose a sleep state to enter when the CPU is idle..

The *ladder* governor has two thresholds, one for promotion and another for demotion. It chooses a progressively deeper C state if the previous residency is maintained above the threshold for promotion, and chooses a lighter sleep state if the residency is below the demotion threshold.

The *menu* governor uses information provided by processor vendors about C state entry/exit costs and the next known timer wake-up event to estimate the amount of time that the CPU will be idle. It then corrects its estimate by checking if there is outstanding disk I/O that may cause the CPU to wake up early. Using its estimate of idle time, it chooses a C state that has a break-even energy cost (i.e. it will choose a deep sleep state only if it thinks that energy will be saved).

The effectiveness of these governors has not previously been analysed quantitatively in the literature. In Chapter 4 we present some results from use of these governors.

2.5.3 Discussion

Many of these frameworks suffer from high complexity. The frameworks tend to be difficult to apply to real-world systems as they use complex models that depending on a system have defined characteristics. If a model is used on a system that does not possess these characteristics, it may not achieve any energy savings and may actually increase energy consumption.

For example, Linux uses comparatively simple, heuristic-based algorithms that make assumptions about the use of DVFS which may adversely affect system energy efficiency. A common fallacy that users of Linux often fall for is assuming that just because DVFS is implemented on their processor, using it will improve their laptop's battery-life, when in fact, it may reduce it.

Chapter 3

The Impact of Memory Throughput

3.1 Introduction

To determine whether it is best to ‘*slow down*’ or ‘*go to sleep*’, it is necessary to understand how changing the CPU’s frequency will impact the performance of a workload. This, in turn requires an understanding of the basic architecture of modern computer systems.

Most modern systems are based on the *von Neumann* architecture. At the heart this architecture is a CPU connected to a block of memory, usually consisting of *dynamic random access memory* (DRAM) from which both data and instructions are *fetch*ed to be executed on the CPU. The DRAM (often called *main memory*) and the CPU operate asynchronously at different clock rates, hence there is a performance disparity between tasks that are *CPU-bound*, (i.e. few instructions require operands sourced from main memory) and tasks that are *memory-bound* (i.e. many instructions require operands sourced from main memory). Other architectures, such as the *Harvard* architecture which has separate memory for instructions and data, are less common. For this reason, we focus on the von Neumann architecture.

In the past, CPU performance has increased at a greater rate than memory performance, resulting in a growing gap. This has resulted in a *bottleneck*, often referred to as the *von Neumann bottleneck*, where many tasks require access to a shared, unified memory over a single shared bus. As multi-core processors with many cores accessing a single, shared main memory become more common, this gap will continue to exist. However, for the still common class of workloads that are single-threaded and use only a single core, this gap has recently begun to shrink as CPU core frequencies have plateaued and memory performance has continued to improve. When running alone, these workloads can see significant benefits from the much greater memory throughput available on recent systems.

The range of *memory-intensity*, from CPU-bound to memory-bound, is continuous. Where a particular task falls on this continuum depends critically on the type of workload that is running

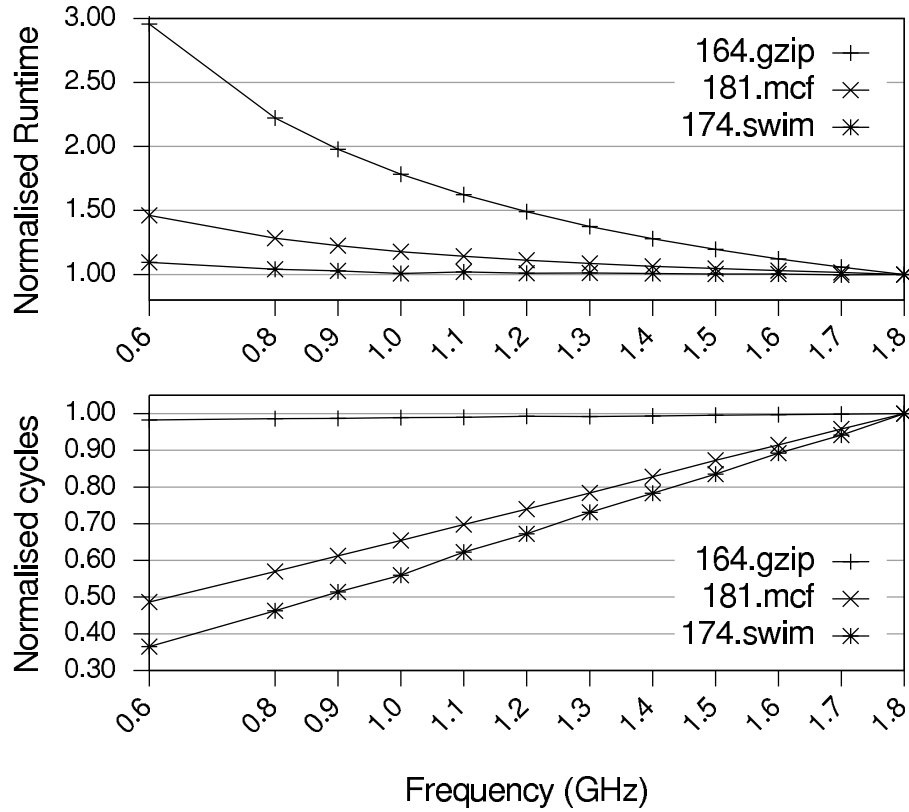


Figure 3.1: Runtime vs. frequency (top) and CPU cycles vs. frequency (bottom) for several SPEC CPU2000 benchmarks on a Pentium-M. All values are normalised to the maximum frequency, 1.8 GHz.

and the patterns in which data are accessed. It also depends on the number of tasks that are running. For example, workloads that are CPU-bound when running in isolation may become more memory-bound when co-scheduled together on a multi-core processor. This is because some resources such as L3 caches, are shared amongst all cores.

The effectiveness of DVFS at improving energy efficiency is also workload dependent. The runtime of CPU-bound workloads, such as *gzip* increases significantly as CPU frequency is reduced, as shown in the top graph of Figure 3.1. This is because the number of CPU cycles required to execute this type of workload is approximately constant, regardless of CPU frequency. This is shown in the bottom graph in Figure 3.1, where the line for *gzip* is approximately flat.

However, the other workloads, *mcf* and *swim*, have very different characteristics. They are examples of *memory-bound* workloads and require significantly fewer CPU cycles to execute at lower CPU frequencies, also shown in Figure 3.1. This is because when executing memory-bound workloads, the CPU issues many instructions into the pipeline that require operands from main-memory, saturating the memory hierarchy. This causes the pipeline to stall while waiting for operands to become ready, before the pipeline can be restarted and execution may continue.

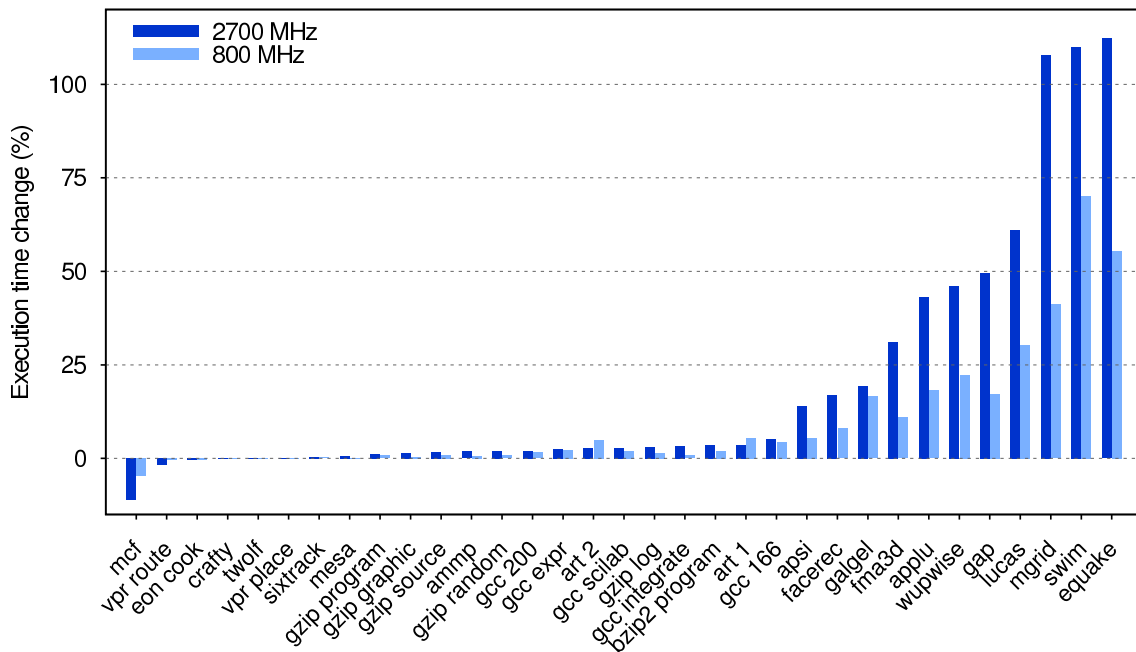


Figure 3.2: Change in execution time of the SPEC CPU 2000 workloads on a recent AMD Opteron with prefetching **disabled**. Note that the performance of 181.mcf actually improves with prefetching disabled.

The disparity between memory and CPU performance can create power-management opportunities, e.g. using DVFS to slow the CPU down when stalling is detected. On the Pentium-M-based laptop, *swim* is almost 100% memory-bound. At one third the CPU frequency, almost one third of the CPU cycles are required to execute the workload. This corresponds to a 9% performance degradation when compared to the maximum frequency. Snowdon found that by reducing the CPU frequency for this workload, energy efficiency could be improved by approximately 24% [Snowdon et al., 2009].

The bottom graph in Figure 3.1 shows that the relationship between CPU frequency and normalised CPU cycles is clearly linear for all three workloads. Hence, the number of CPU cycles required to execute a workload can be estimated, provided there is data that can be measured to predict the memory-intensity of the workload. In the past, as described in Section 2.4.3, *performance-counter* events, such as LLC misses, have been used as a means to predict a workload's memory-intensity. In Section 3.3, we look at how such a model can be constructed.

3.2 The Effects of Aggressive Prefetching

Prefetching is a technique used to improve memory throughput for workloads that access memory with predictable patterns. Modern processors implement a *stride detector* which analyses the memory addresses that the CPU calculates and issues to the memory controller. If a pattern of accesses with a

consistent distance between addresses (called stride size) is detected, the cache lines containing the data are *pre-fetched* from main-memory into a local cache, before the processor issues the instruction which requires the data. The *prefetch distance* is the number of cache lines that will be prefetched when a strided access pattern is detected. With raw increases in memory throughput from new memory technologies (DDR2, DDR3, etc.) prefetch distance is continually being increased.

Figure 3.2 shows how prefetching affects the performance of the complete SPEC CPU2000 benchmark suite on a recent quad-core AMD Opteron processor codenamed *Shanghai*. When prefetching is disabled, most of the workloads experience a significant increase in execution time. However, some, such as 181.mcf (shown far left in the figure) actually benefit from having prefetching disabled. Overall, prefetching gives an impressive boost in performance.

3.3 Modelling Workload Performance Under Varying CPU Frequency

To determine when the CPU's frequency can be scaled without sacrificing performance, we need to predict how a workload will perform at different CPU frequencies.

As described in Section 2.4.3, Snowdon's *Koala* framework uses two predictive models, one for execution time and another for power consumption. These provide an estimate of the energy consumed by the CPU when executing a task. Choosing parameters and their coefficients requires a one-time model-characterisation on the system that is being modeled. The models are then used at runtime to provide energy consumption estimates which drive power-management (DVFS) decisions.

Snowdon et al. analysed several systems. However, the two systems for which models were built and validated were less complex than the systems available today. The Pentium-M- and first-generation AMD Opteron-based systems that were modeled are single-core systems. Today, all systems have at least a dual-core processor, including embedded systems such as the iPad 2 and Pandaboard. In addition, Snowdon et al. simplified the models by making certain assumptions about the systems he was modelling. For example, he assumed that the workloads being modeled did not perform disk I/O.

To improve Snowdon's *Koala* power-management framework, thus allowing it to be used on more recent systems with *multi-core* processors, would require relaxing some of the assumptions made during its development and improving it so that it has:

- a reduced reliance on the SPEC CPU benchmark suites for model-characterisation;
- the ability to accurately model workloads that perform I/O to mass-storage;
- the ability to accurately model workloads that cause slack-time in the system; and
- the ability to accurately model multiple CPU cores with shared resources.

Multi-core processors present several interesting complications for this type of modelling when compared to their more simple, single-core predecessors. For example, the individual cores of modern a *chip multi-processor* (CMP) share an on-die cache, which allows multi-threaded workloads to share data efficiently. A performance-counter event that measures the number of accesses to main-memory needs to be broken down into per-core event counts in order to model the characteristics of tasks on each core separately and the models need to be adapted to handle these situations.

Initially, we focus on the first point, by developing a methodology to characterise the models using a synthetic workload.

3.3.1 Model characterisation

Snowdon's work lead to the development of the models for a single-core AMD Opteron-based system codenamed (Sledgehammer). The models were characterised using SPEC CPU2000 and validated using SPEC CPU2006, as per the stages described in Section 2.4.3.

Characterising the two models (i.e. execution time and power consumption) is a complex process. On the one hand, the characterisation set should exercise the entirety of the CPUs architectural features, giving the model a wide coverage and ability to predict general workloads. On the other hand, the *performance-monitoring unit* (PMU) has many hundreds of events that can be chosen to parameterise the models. In addition, modern CPUs are very complex devices with many (often measurable) architectural features designed to improve performance. A larger characterisation set will exercise a wider range of these performance-counter events, however the number of simultaneously measurable counters is currently limited to two or four, depending on processor generation. The Pentium-M processor allows simultaneous measurement of two events, while AMD's Opteron processors allow four events to be measured. In the past, the Netburst architecture on which the Pentium-4 was based allowed 18 events to be measured simultaneously. The small number of simultaneously measurable events on recent processors makes modelling *general* workloads difficult, because the complete set of behaviours of a large characterisation set cannot be predicted to high accuracy with only two to four parameters. We found that characterising a two to four parameter model with a very large characterisation set resulted in a model that performed poorly for all workloads.

Bircher and John found that a model based on L3 cache misses alone did not predict the energy consumed by an AMD quad-core Opteron system to a high accuracy. They found that when running multiple instances of the SPEC CPU2000 181.mcf workload, memory power increases while L3 misses actually slightly decrease [Bircher and John, 2007]. This is a perplexing result, suggesting that the Opteron processor would be difficult to model using performance counter events.

Snowdon et al. had more success, finding a balance between characterisation-set-size and model generality for a relatively simple single-core processor, the AMD *Sledgehammer* Opteron. Figure 3.3 shows a comparison between model-estimated versus actual execution time (top) as well as model-

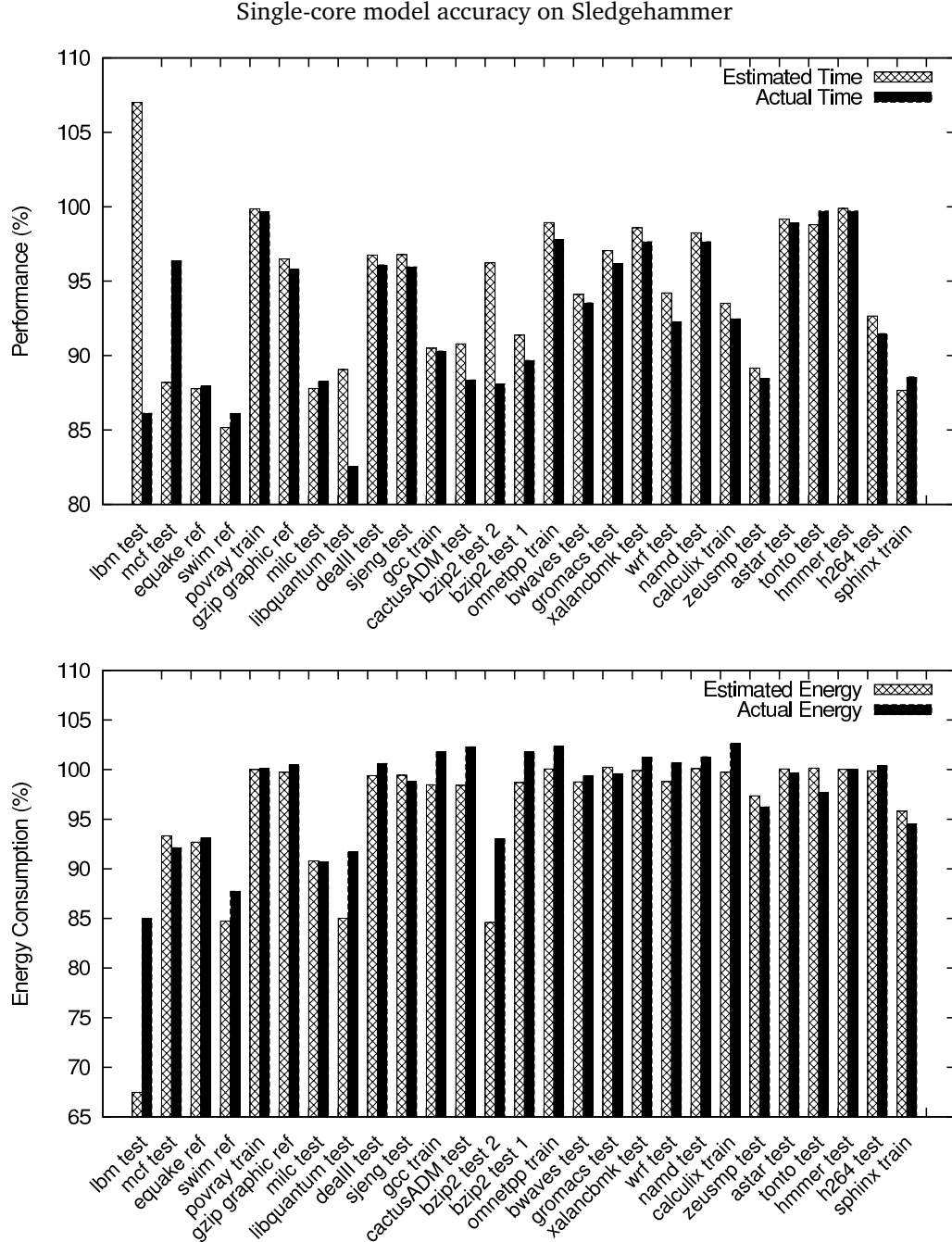


Figure 3.3: Comparison of model estimated execute time (top) and model estimated energy consumption (bottom) on the Sledgehammer platform with a four-parameter model. Model accuracy is good for most benchmarks in the set, however **lbm test** and **mcf test** on the left have huge over- and under-estimations respectively.

estimated versus actual measured energy consumption (bottom). The figure shows that the models predict well for a large number of the workloads in the benchmark suite. However, the models predict poorly for several workloads, such as **lbm_test** and **mcf_test**.

Performance-counter events are designed to measure very specific events and Snowden et al's results suggest that the performance-counter events used in their models were unable to characterise a model with the ability to predict accurately for general workloads. Jotwani et al. found that 95 parameters are required to model the power consumption of a recent AMD processor to within 2% error, for a wide variety of workloads [Jotwani et al., 2010].

3.3.2 Developing a synthetic characterisation workload

To improve the models in the situation where we can only measure up to four parameters, we surmised that a synthetic benchmark, tuned to exercise different architectural features of a CPU core, could be used to characterise a model that performs better when applied to general workloads.

We constructed a synthetic benchmark with model characterisation in mind. Such a benchmark should:

- have a controllable runtime, to be able to manage the time for characterisation,
- have controllable memory-intensity, to cover the continuum of memory-intensity,
- be able to exercise the main functional units of a CPU core, to cover a wide range of workload types,
- be able to run on multiple cores, and
- be statistically sound.

We designed the benchmark to copy data between two buffers of variable size. To vary memory-intensity, several parameters can be tuned, namely:

- Buffer size: This parameter can be varied to determine whether the benchmark will fit inside, or cause misses, at the various levels of cache available on a processor. If it is increased beyond the size of the LLC, resulting memory accesses will be forced to main-memory.
- Stride size: This parameter can be varied or randomised to determine the effectiveness of the hardware prefetch unit. Stride-size can also be used to vary, in a controlled way, the percentage of accesses that will miss in the caches. An access pattern which has a constant stride-size (the distance between memory access addresses) will allow the prefetch unit to actively fetch operands *before* the CPU actually issues the instructions that require them.

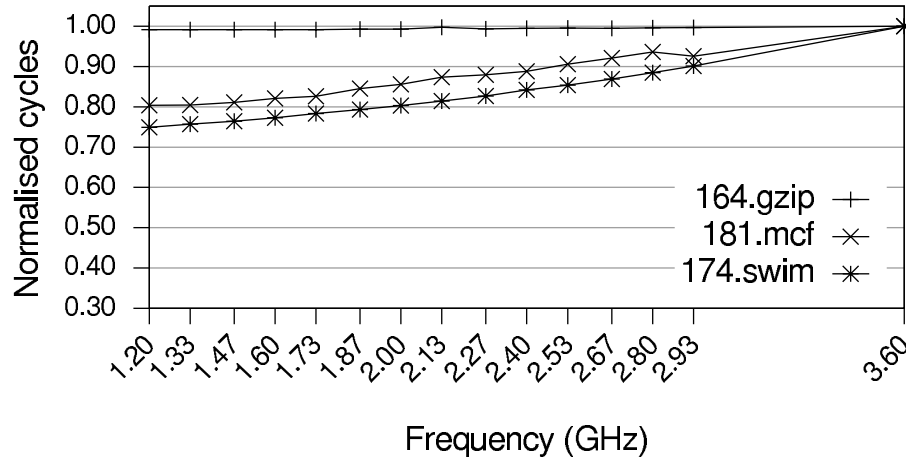


Figure 3.4: Comparison of normalised cycle count vs. CPU frequency for several SPEC CPU2000 benchmarks on a more recent Core i7. 3.60 GHz represents a TurboBoost frequency, all values are normalised to this. Both *swim* and *mcf* show a non-linear relationship that is caused by hardware prefetching.

- Compute operations: A variable number of compute operations (e.g. adds, multiplies, divides etc.) or nops (no operation instructions) can be introduced to increase the time between memory operations.

We use this workload in the next section.

3.3.3 Prefetching effects

The graph in Figure 3.4 shows an unusual result that was observed on several of our platforms. As can be seen, the memory-intensity of the *mcf* and *swim* workloads on this platform varies non-linearly with CPU frequency. Using a synthetic workload, we determined that this was caused by prefetching. A reduced CPU frequency creates longer latency between memory requests, thus allowing the prefetch-unit more time to satisfy memory requests for operands before the processor issues the instructions requiring them.

Our attempts to model the effects of prefetching, were, ultimately, not successful. We believe that this was primarily due to limitations in the events that the PMU can observe. We tried to model the effects on an AMD Opteron system (based on a quad-core Shanghai processor) using the synthetic benchmark described above. Figures 3.5 and 3.6 show the parameters that were selected for the execution time model when prefetching was enabled and disabled. In these figures, each vertical column represents a different model. A black or grey rectangle means that the corresponding parameter on the left has been used as a predictor in that model. Moving toward the right, more parameters are added, up to a total of four, which is the maximum number of discrete performance-counter events that can be measured at once (on the AMD Shanghai processor). The intercept is

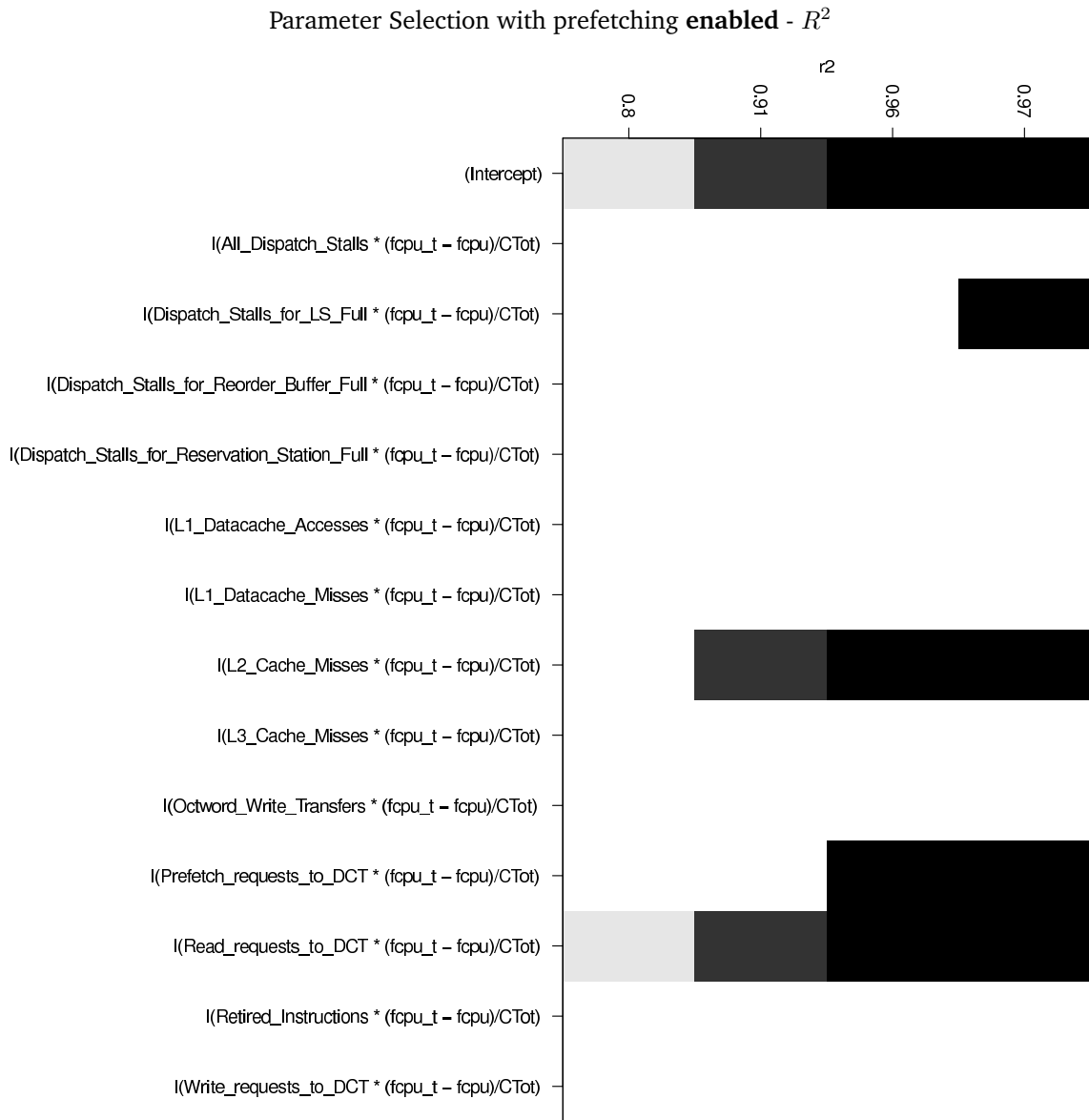


Figure 3.5: This figure shows which parameters are selected for the time model when prefetching is **enabled** and their correlation coefficients. Each column represents a model with one more parameter included up to a maximum of 4 parameters. Y axis is R^2 .

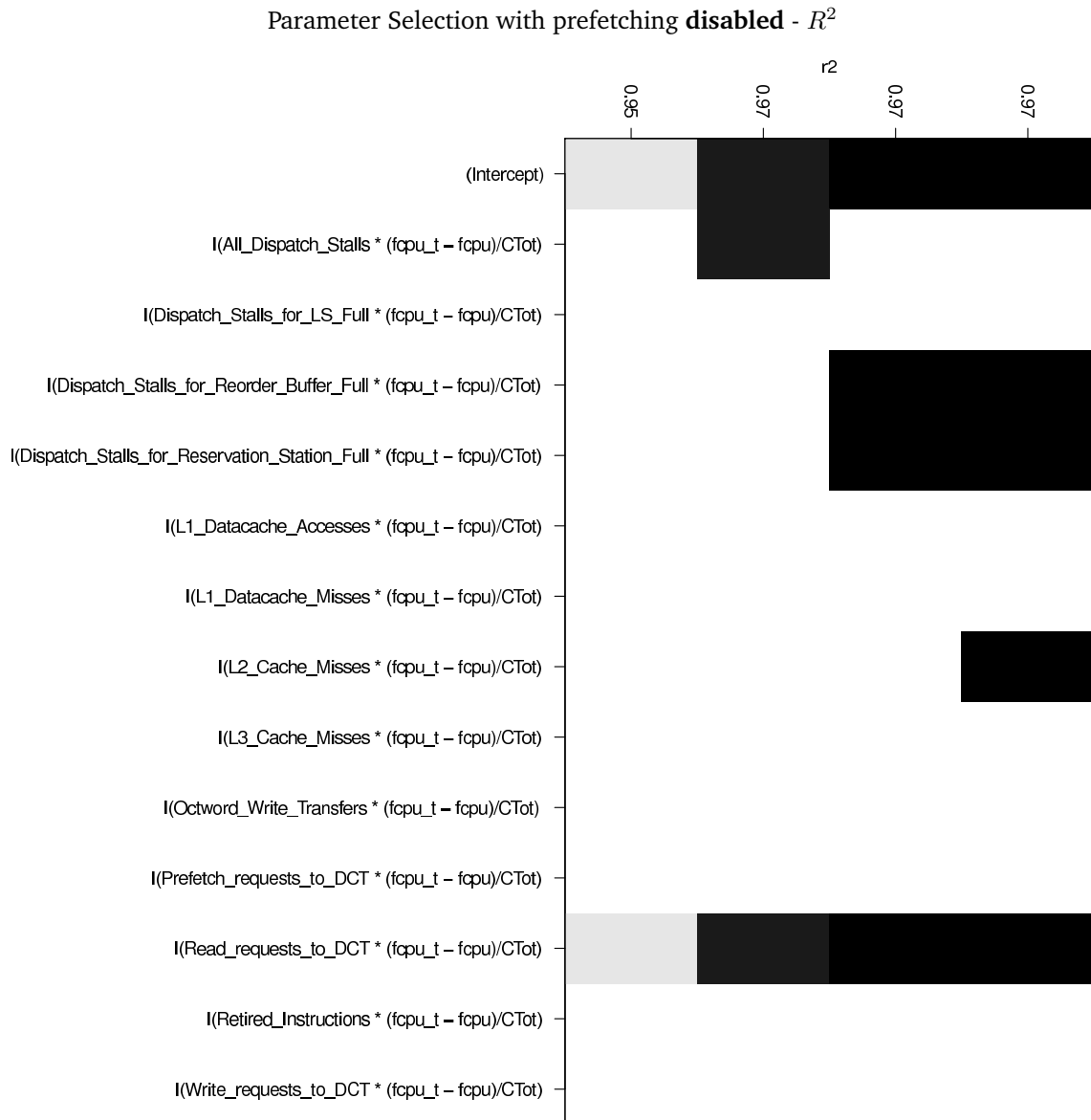


Figure 3.6: This figure shows which parameters are selected for the time model when prefetching is **disabled** and their correlation coefficients. Each column represents a model with one more parameter included up to a maximum of 4 parameters. Y axis is R^2 .

simply a constant value that is included in the model, regardless of the predictor values. Hence, it is included in all models.

In both cases, a single-parameter model (the left-most vertical column) selects the READ REQUESTS TO DRAM controller (DCT) event as its only predictor. However, where prefetching is **enabled**, the resulting model has a correlation coefficient (R^2) of just 0.8. This represents a model that can explain approximately 80% of the characterisation set data. If prefetching is **disabled**, the single parameter model has a significantly improved R^2 of 0.95. This difference in model accuracy is due to the single performance-counter event not being tightly correlated to actual memory bus utilisation.

When a two-parameter model is used (the second column), the regression subset-selection chooses

READ REQUESTS TO DCT; and

ALL DISPATCH STALLS

when prefetching is **disabled**, and

READ REQUESTS TO DCT; and

L2 CACHE MISSES

when prefetching is **enabled**. We would expect the PREFETCH REQUESTS TO DCT event to be picked to allow the model to predict the effectiveness of the prefetcher. Instead, the number of L2 cache misses is chosen as the better predictor, suggesting that the number of prefetch events is less significant. This two-parameter model has an R^2 of 0.91 when prefetching is enabled, compared to 0.97 with prefetching disabled. The ALL DISPATCH STALLS event would seem to be a good choice if hand-picking predictors for a model, however as it turns out, when prefetching is enabled, this event is never picked by the automatic regression subset-selection.

If we increase the number of parameters to three (the third column), we see that the PREFETCH EVENTS TO DCT event is now chosen and the model has an improved R^2 of 0.96 when prefetching is enabled. However, this is still not as good as the case where prefetching is disabled, where R^2 is 0.97.

If we increase to four parameters, the model gets a slightly improved R^2 when prefetching is enabled. However, other problems begin to appear with many-parameter models, such as over-fitting and poor generality. With prefetching disabled, the model does not improve past two parameters, suggesting that prefetching is the root cause of the inaccuracy and that the other parameters have little statistical significance.

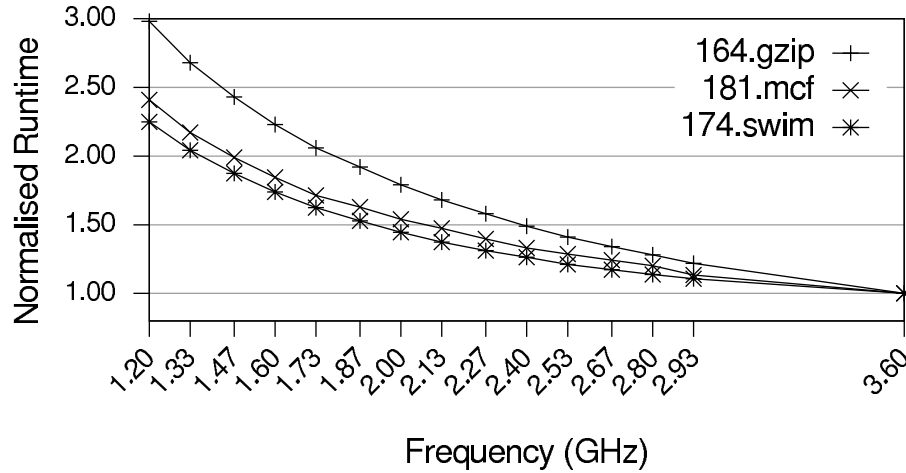


Figure 3.7: Comparison of normalised runtime vs. CPU frequency for several SPEC CPU2000 benchmarks on the Core i7. 3.60 GHz represents a TurboBoost frequency, all values are normalised to this.

3.4 Memory Throughput and DVFS

While observing the effects of prefetching on workload performance, we also noted that increasing memory throughput reduced memory-related stalls in the processor pipeline, causing some workloads to become less memory-bound. This has an effect on the way in which workload performance scales under DVFS.

Figure 3.7 shows runtime of the three SPEC CPU2000 workloads *gzip*, *swim* and *mcf* on a Core i7-based desktop under varying CPU frequency. All three workloads experience significant increases in runtime at reduced CPU frequencies, in contrast with the Pentium-M where the memory-bound workloads *swim* and *mcf* experience only marginal increases in runtime.

Figures 3.1 and 3.4 show normalised CPU cycles for these workloads on the Pentium-M-based laptop and the Core i7-based desktop. On the laptop, the memory-bound workloads require significantly fewer CPU cycles when the processor is clocked at a lower frequency. For example, *swim* requires almost 64 % fewer cycles at 0.6 GHz compared to 1.8 GHz. This corresponds to approximately 4 % degradation in performance at one third the CPU frequency, suggesting that *swim* is a very *memory-bound* workload. In contrast, on the desktop, *swim* requires only 25 % fewer cycles at one third the maximum CPU frequency. That is, *swim* is significantly *less* memory-bound on the desktop compared to the laptop.

The same can be said for *mcf*. The reason for this change in memory-intensity is because memory throughput and latency has improved significantly in the time between the manufacture of the two test platforms. Characteristics of the two systems are compared in Table 3.1. As the table shows, the DRAM used in the Pentium-M laptop is *double-data-rate* (DDR) at a frequency of 200 MHz, which

System	Dell Vostro 430s	Dell Latitude D600
Processor	Intel Core i7 870	Intel Pentium-M 745
Die Codename	Nehalem	Dothan
ISA	64-bit x86	32-bit x86
Class	Desktop	Laptop
Cores	4	1
Threads	8	1
Frequency (GHz)	1.2–2.93	0.6–1.80
TurboBoost (GHz)	3.6	-
Voltage (V)	0.65–1.40	0.95–1.34
Process feature-size	45 nm	90 nm
TDP	95 W	21 W
L2 cache	4×256 KiB	2 MiB
L3 cache	8 MiB (shared)	-
Memory	4 GiB DDR3 1,333 MHz	1 GiB DDR 400 MHz
DRAM channels	3	2

Table 3.1: Specifications of the two systems we analyse for the impact of memory throughput. Thermal design power (*TDP*) is the maximum processor power dissipation expected. [ARK, Intel Corporation, 2011]

gives an effective frequency of 400 MHz. The 1 GiB of memory is configured as 2x 512 MiB modules, allowing it to run in *dual-channel* mode, giving a maximum theoretical throughput of 3.2 GiB/s. In contrast, the memory of the Core i7 desktop is DDR3 which has more *internal* prefetching. In contrast to the prefetching technique described in Section 3.2, this kind of prefetching is inside the chips on the memory modules and amortises the *column access strobe* (CAS) latency over many bytes. The memory controller of the Core i7 uses three channels for improved throughput, giving it a theoretical maximum throughput of 10.667 GiB/s, more than three times the throughput of the laptop. However, the two processors also have different a number of cores. The Pentium-M is a single-core processor and its memory bus needs only to handle the requests from a single workload at a time. In contrast, the Core i7 has four cores all sharing the same main memory and memory-bus hence the memory hierarchy needs to handle a much greater potential throughput if all cores are making requests for data at once. However, single-threaded workloads, like those used above are still the predominant workload type on desktop systems, hence single-threaded memory performance is still important.

3.5 Conclusions

CPU workloads sit on a continuum of memory-intensity. At one end, the runtime of CPU-bound workloads, like *gzip*, significantly increases as CPU frequency is decreased. At the other end of the continuum, the runtime of memory-bound workloads like *swim* and *mcf* increases less dramatically with reduction in CPU frequency. Snowdon et al. have shown that in the past, the energy efficiency

of systems running memory-bound workloads can be improved significantly by reducing the CPU frequency [Weissel and Bellosa, 2002; Snowdon et al., 2009].

However, we found that the memory-intensity of single-threaded workloads, such as those that are part of the SPEC CPU suites, has been significantly reduced due to the large increase in memory throughput in the time between the release of our two test systems. When these workloads are run on systems with multi-core processors, such as the Core i7-based desktop, memory-intensity is reduced by up to 50 % when compared with the older Pentium-M-based system. As shown in the next chapter, even running multiple concurrent instances of these workloads results in increased energy consumption with DVFS.

Furthermore, we find that the performance-counter events available on the Shanghai Opteron system are not capable of accurately predicting the effectiveness of the prefetch unit. This is a critical problem when characterising the execution time model required for Snowdon's *Koala* power-management framework. As shown in Figure 3.2, prefetching gives a significant improvement in execution time for almost all of the SPEC CPU2000 workloads and the execution time model must be able to predict its' effect. However, the model must also predict the slowdown experienced by *mcf*, (as shown in Figure 3.3) which, even with a relatively simple single-core processor, it is incapable of predicting accurately.

These findings suggest that there is a trend toward diminishing energy efficiency gains from DVFS. However, this factor is one of many. The rest of this thesis aims to enumerate and analyse other factors contributing to the diminishing returns from DVFS and to answer the central question: *slow down or go to sleep?*

Chapter 4

Slow Down or Sleep?

4.1 Introduction

Over the years, processor manufacturers have introduced many new power-management mechanisms. Section 2.3.2 introduced *idle modes* or C states, which are low-power modes that the cores of a CPU can be put into when they are not needed for execution.

Idle modes are important for workloads that create *slack-time* in a system. Unfortunately, unmodified SPEC CPU workloads do not allow the system to idle at all during their execution, thus the idle modes of more recent CPUs are not exercised when these benchmarks are used.

Before we can finally answer the question *slow down or sleep?*, we must understand the way workloads use CPU resources on a system and the costs associated with transitions between C states.

Much of the prior research into DVFS has used the SPEC CPU benchmark suites to analyse the trade-offs in power-management [Standard Performance Evaluation Corporation, 2006]. These suites are comprised of single-threaded, *high-performance computing* (HPC) workloads, designed to stress the CPU and memory systems. In order to stress modern *multi-core* systems, many studies simply run two or more instances of these workloads. Some choose workload combinations randomly, while others hand-pick benchmarks to co-schedule. This methodology is sound, however incomplete, because the workloads are still single-threaded and do not stress the cache-coherency framework that CMP systems employ.

In this chapter, we analyse the effectiveness of DVFS by looking at a range of workload classes. Firstly, we look at several SPEC CPU workloads that are compute-intensive and run the CPU at 100 % utilisation until completion. Next, we use a specialised framework to artificially introduce slack-time into these SPEC workloads, therefore making them act like more bursty, real-world workloads. Lastly, we look at several actual real-world workloads—MPEG playback and serving

System	Compucon K1-1000D	Dell PowerEdge SC1435	HP ProLiant DL365 G5
CPU model	246	2216	2384
CPU die codename	Sledge-hammer	Santa Rosa	Shanghai
Year	2003	2006	2009
Core count	1	2	4
Frequency (GHz)	0.8–2.0	1.0–2.4	0.8–2.7
Voltage (V)	0.9–1.55	0.9–1.35	1.0–1.35
TDP	89 W	95 W	75 W
Feature size	130 nm	90 nm	45 nm
Die size	193 mm ²	230 mm ²	285 mm ²
Transistor count	106 M	243 M	463 M
L1 cache	64 KiB I & 64 KiB D per core		
L2 cache (per core)	1 MiB	1 MiB	512 KiB
L3 cache	-	-	6 MiB (shared)
Line-size	64 bytes at all levels		
Memory	512 MiB DDR 400 MHz	4 GiB DDR2 667 MHz	8 GiB DDR2 667 MHz
DRAM channels	1	2	2
Prefetch distance (in cachelines)	2	2	3
System idle power	74 W	145 W	133 W

Table 4.1: Specifications of the three AMD Opteron processors and systems we analyse for the compute-intensive workloads. All systems have a single CPU package. Thermal design power (*TDP*) is the maximum processor power dissipation expected [Advanced Micro Devices, 2010]

web-pages. On the one hand, MPEG playback is a single-threaded, bursty workload which is common on many system classes. On the other hand, a web-server workload is also inherently bursty, but is also multi-threaded, running on all the cores that are available in a CMP. Therefore, a web-server workload will exercise the complete cache hierarchy of a modern CMP, including the cache coherency system. These workloads provide a wide range of characteristics allowing us to analyse the use of DVFS and C states under varying conditions.

4.2 Compute-intensive Workloads

Compute-intensive workloads (as opposed to I/O intensive workloads) are those that run the CPU at 100 % until completion. At no time is the CPU able to *idle* during their execution. Memory-bound workloads may still cause the CPU pipeline to *stall* during execution, but the CPU **cannot** be put into an idle state during these periods of pipeline stalling.

Firstly, we look at three generations of server-class AMD Opteron-based systems released over a span of 7 years. The newer generation processors are built from smaller transistors, have more cores, larger caches and are attached to faster main memory. The characteristics of these systems are shown

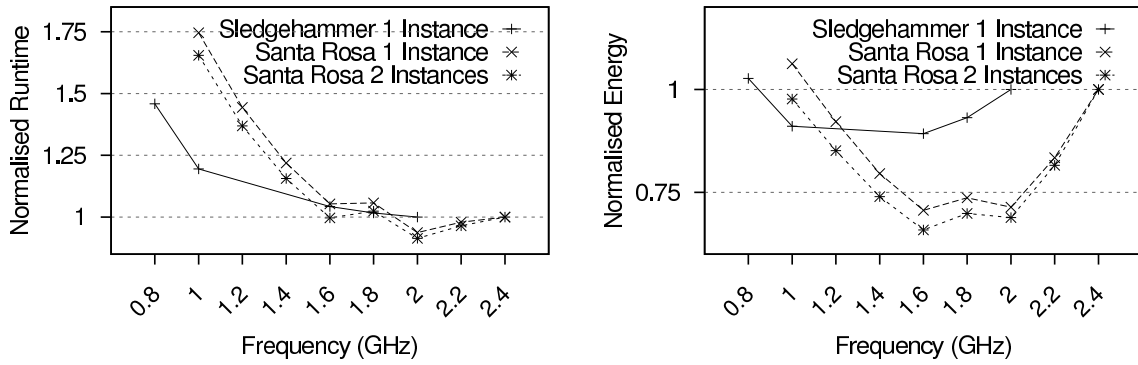


Figure 4.1: Runtime (left) and energy consumption (right) of *mcf* on the Sledgehammer and Santa Rosa systems.

in Table 4.1. Henceforth, they will be referred to by their processor code-names, *Sledgehammer*, *Santa Rosa* and *Shanghai*.

As discussed in Chapter 3, memory-bound workloads offer the best opportunities to improve energy efficiency using DVFS because the CPU can potentially be slowed down without significantly sacrificing performance. Therefore, to test the best case for DVFS being effective at improving energy efficiency, we selected the most memory-bound benchmark workload on all our systems, the 181.MCF (*mcf*) workload that is part of both SPEC CPU2000 and CPU2006. There is little difference between the two versions other than the total runtime of the benchmark.

By running multiple instances of this workload, we are able to stress the high-throughput main memory present in modern CMP systems allowing us to test the best-case opportunities for improving energy efficiency with DVFS.

4.2.1 Methodology

The three systems were instrumented with a power meter [Extech Instruments Corporation, 2008] at the PSU measuring the **total power** drawn by each system. The systems were set up to run Linux with kernel version 2.6.35. All tests were carried out from a RAM-backed file system to ensure that there was no disk activity, as this would perturb our measurements. The same 64-bit benchmark binary was used on all three platforms, compiled with GCC 4.2 with high optimisation enabled.

Each test was run 10 times then averaged to ensure the results were statistically sound. All results we present have a standard deviation of less than 1% of the mean.

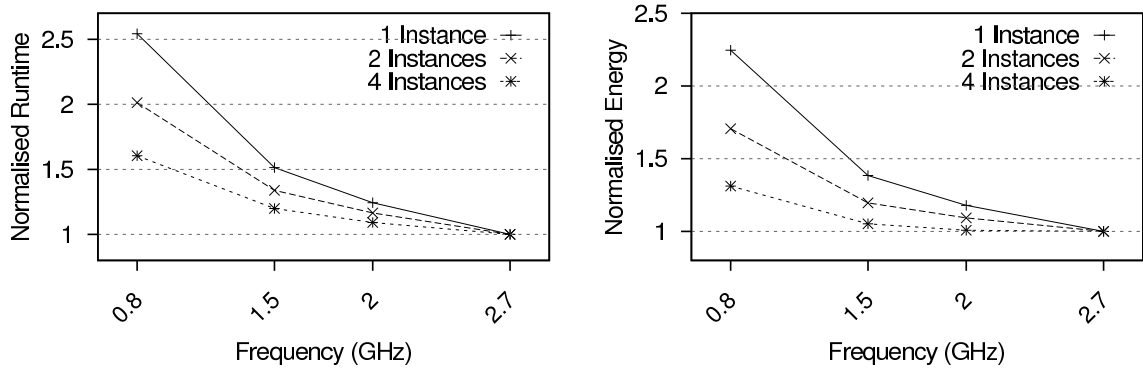


Figure 4.2: Runtime (left) and energy consumption (right) of *mcf* on the Shanghai system.

4.2.2 Results

We first looked at the two older platforms, one based on Sledgehammer, a single-core processor running at 2.0 GHz, built at the 130 nm feature size with DDR main-memory, and the other based on Santa Rosa, a dual-core processor built at a feature size of 90 nm running at 2.4 GHz with DDR2 main-memory. Figure 4.1 shows runtime and energy consumption vs. CPU frequency (both normalised to the maximum CPU frequency of each platform). As we expected, the reduced frequency causes runtime to increase, as shown in the left graph. However, energy consumption (shown in the right graph) decreases when the CPU frequency is reduced. On the Santa Rosa system, energy efficiency can be improved by up to 30 % when executing two instances of the *mcf* workload. Hence, on the first and second generation Opteron-based platforms, DVFS is effective at improving energy efficiency for this memory-intensive workload.

Next, we looked at the most recent platform based on the Shanghai processor. This processor is constructed from a 45 nm process, has four cores sharing a large L3 cache and runs at 2.7 GHz at the same voltage as the Santa Rosa.

Figure 4.2 shows that energy consumption of the Shanghai system only increases with reduced CPU frequencies. Even if four instances of *mcf* are run simultaneously on separate cores, energy consumption is still increased significantly. This is in stark contrast to the two previous generations, where up to approximately 30% energy could be saved.

4.2.3 Discussion

These initial results show that simply comparing energy consumption between different frequencies on a more recent system always results in higher energy consumption. However, some introspection reveals that this approach is limited, because the SPEC workloads are not representative of those that are run on real systems. Indeed, very few real-world systems run HPC workloads. Therefore, the methodology needs to be expanded to include more realistic workload scenarios.

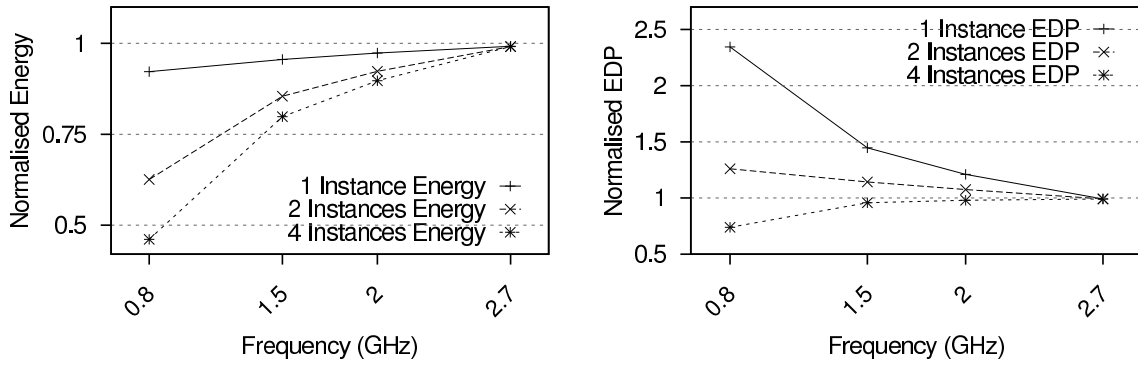


Figure 4.3: Padded energy consumption for one, two and four instances of *mcf* on Shanghai (left) and padded energy-delay product (EDP) (right). All values are normalised to the maximum CPU frequency.

4.3 Padding with Idle-energy

Reducing the CPU's frequency lengthens the execution time of tasks, which causes the system to consume disproportionately more static power at low frequencies. For this reason, energy consumption comparisons cannot be made fairly between multiple frequencies. A benchmark that was run at a high frequency is unfairly biased, because there is an implicit assumption that once a benchmark completes, the system is switched off. This is not what happens in reality. The system will stay powered on, albeit in an idle state. To account for this, we can *pad* the faster, high-frequency benchmarks with the energy consumed while the system would have been idle. The amount to pad depends on the difference between the execution time of the longest-running benchmark (at the lowest frequency) and the benchmark being padded.

The left graph in Figure 4.3 shows energy consumption for one, two and four instances of *mcf* on the Shanghai system when shorter-running benchmarks are padded with idle energy calculated from the idle power values given at the bottom of Table 4.1. Here, we can see that energy consumption is reduced when the CPU frequency is reduced.

4.3.1 Energy-delay product

Often, performance is still an important factor. However, simply minimising energy consumption can result in a significant reduction in performance for meagre energy savings. For this reason a better metric for comparison is the energy-delay product (EDP), which can be used to optimise MIPS/W (i.e. computational efficiency):

$$EDP = E \times T, \quad (4.1)$$

System	Dell Vostro 430s	fit-PC2	Pandaboard
Processor	Intel Core i7 870	Intel Atom Z550	OMAP 4430 Cortex A9
ISA	64-bit x86	32-bit x86	32-bit ARMv7
Class	Desktop	Embedded	Embedded
Cores	4	1	2
Threads	8	2	2
Frequency (GHz)	1.2–2.93	0.8–2.0	0.3–1.008
TurboBoost (GHz)	3.6	-	-
Voltage (V)	0.65–1.40	0.75–1.1	0.93–1.35
Process	45 nm		
TDP	95 W	2.4 W	Unknown
L2 cache	4×256 KiB	512 KiB	1 MiB (shared)
L3 cache	8 MiB (shared)	-	-
Memory	4 GiB DDR3 1,333 MHz	1 GiB DDR2 533 MHz	512 MiB LPDDR2 800 MHz
DRAM channels	3	2	2
Storage	500 GiB 3.5" SATA hard-drive	80 GiB SATA 2.5" hard-drive	64 GiB USB SSD 4 GiB SD
Graphics Acceleration	None	SGX 535 and VXD	SGX 540 and IVA-HD

Table 4.2: Specifications of the three processors and systems we analyse for the idling workload scenarios.

Thermal design power (*TDP*) is the maximum processor power dissipation expected. [ARK, Intel Corporation, 2011; OMAP 4430 Technical Reference Manual, 2011]

where E is energy consumed and T is execution time. The right graph of Figure 4.3 which plots energy-delay product (EDP) shows that if execution time is accounted for, the optimal frequency is still the maximum, unless four instances of *mcf* are running simultaneously.

4.3.2 C state overview

The AMD Opteron-based systems used in the experiments above implement only a single idle state, making them uninteresting for the following experiments. Table 4.2 outlines the characteristics of three systems which implement multiple levels of idle states. We use these systems for the experiments in the remainder of this chapter. The rest of this section describes the available C states on these processors.

As discussed in Section 2.3.2, constraints exist between package-, core- and thread-level C states. On the Intel processors (Core i7 and Atom), C1 is the only state which can be entered by an individual

C state	Details	Intel Core i7 870	Intel Atom Z550	OMAP 4430 Cortex A9
C1		available		
	Entry	via HALT or MWAIT instruction		wait-for-interrupt (WFI) instruction
	Clocking / Voltages	some clock-gating is applied		
	Notes	Cores respond to remote cache snoops for coherency		some of MPU in data retention state
C1E		available	unavailable	
	Entry	automatic when all cores in C1		
	Clocking / Voltage	frequency and voltage are reduced		
C2		unavailable	available	
	Entry		via MWAIT instruction	explicit shutdown plus WFI
	L1 caches			flushed
	L2 caches			flushed
	Notes		Interrupts from system-controller hub (SCH) masked	co-processor state must be explicitly saved
C3		available	unavailable	available
	Entry	via MWAIT instruction		explicit shutdown plus WFI
	L1 caches	flushed		
	L2 caches	flushed		
	Clocking / Voltage			more of MPU in retention
C4		unavailable	available	
	Entry		via MWAIT instruction	explicit shutdown plus WFI
	Clocking / Voltage		PLLs are stopped, voltage is reduced to allow cache retention	MPU switched off, other parts still active
C5		unavailable	available	
	Entry		automatic if L2 cache was flushed before entering C4	explicit shutdown plus WFI
	L1 caches		flushed	
	L2 caches		flushed	
	Clocking / Voltage			MPU switched off, other parts off
C6		available		unavailable
	Entry	via MWAIT instruction		
	Clocking / Voltage	core clocks stopped and core voltage reduced to near-zero		
	Notes	L3 cache is still active and snoopable		

Table 4.3: Characteristics of the different C states on the three test systems. On the OMAP, C states greater than C1 could only be used if the second core was disabled.

[Intel Corporation, 2011b,a; Texas Instruments, 2011]

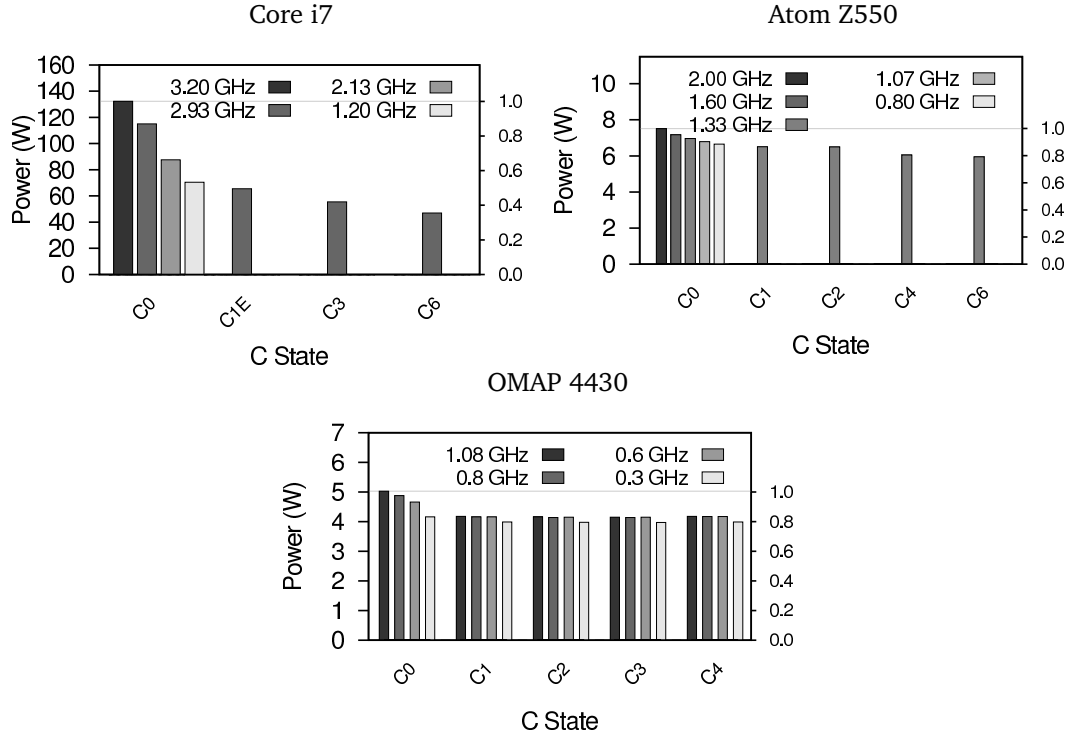


Figure 4.4: Idle power consumption for the Core i7 (top left), Atom Z550 (top right) and OMAP 4430 (bottom middle) when all cores are put in the same state.

thread (HyperThread), all other C states can only be entered by a whole core. Except for the C1E state, there are no constraints between C states entered by different cores.

The OMAP 4430 microprocessor unit (MPU) contains two Cortex-A9 cores, the L1 caches and their controllers, the local interrupt controller and the snoop-control unit (SCU) in one package. Its C states are not defined by the manufacturer, instead, the OS designer must create MPU-wide C states which selectively put different processor subsystems into low-power states. Here we describe the C states defined by Linux 2.6.35. The OMAP 4430 also has several other compute elements, including two Cortex M3 micro-controllers, an image and video accelerator (IVA), a 3D accelerator (SGX 540) and several other components.

On the OMAP, C1 is the only C state which can be entered by an individual core, all other C states are only entered if requested by both cores.

Table 4.3 provides an overview of the different C states that are available on the test systems characterised in Table 4.2. Effects are additive down columns leading to the deepest C state, C6 at the bottom of the table.

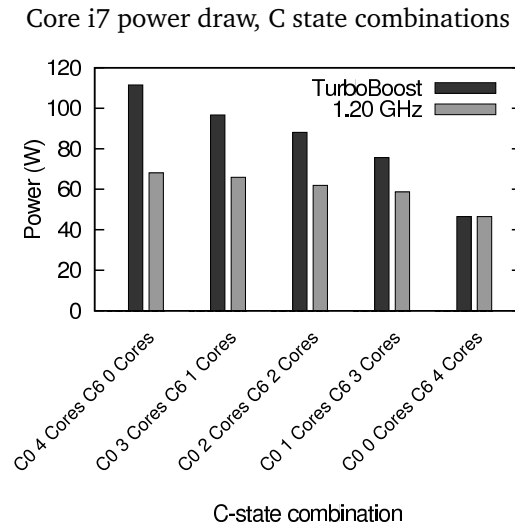


Figure 4.5: Power draw of the Dell Vostro when only some of the cores of the Core i7 area placed in C6. When all cores enter C6, the processor enters the *package* C6 state, achieving even lower power draw.

4.3.2.1 C state power consumption characteristics

In Figure 4.4, we show the total system power drawn by each of the three platforms when each of the available C states are invoked. For the platforms with multiple cores, all cores are put in the same C state. As shown, for C0, the power consumption is correlated to the CPU frequency. C0 is essentially the same as a naïve idle thread, where the CPU just spins in a tight loop and no effort is made to reduce power consumption. On the Core i7, the steps from C0 to C1E, C3 and C6 all result in reasonable reductions in power consumption. However, on the Atom and OMAP processors, the reductions achieved from using the deep sleep states are relatively small compared to the Core i7. This is because the power drawn by these processors is a small portion of total system power.

Additionally, some processors, like the Intel Core i7, implement special *package* C states, which are only used when all cores are in the same state and other system activity (such as DMA and interrupts) are disabled. When in these special package C states, the processor can achieve even lower power draw. Figure 4.5 shows total system power draw when different combinations of cores are put in the deep C6 C state. The dark bars represent the case where the processor is able to use TurboBoost mode (i.e. high frequency) and the light bars are when the processor frequency is reduced to 1.20 GHz using DVFS. The left most group is when all cores are active. As we allow each additional core to enter C6, the power draw drops, until all cores are in C6 (the right-most bars), and a larger drop in power draw is achieved.

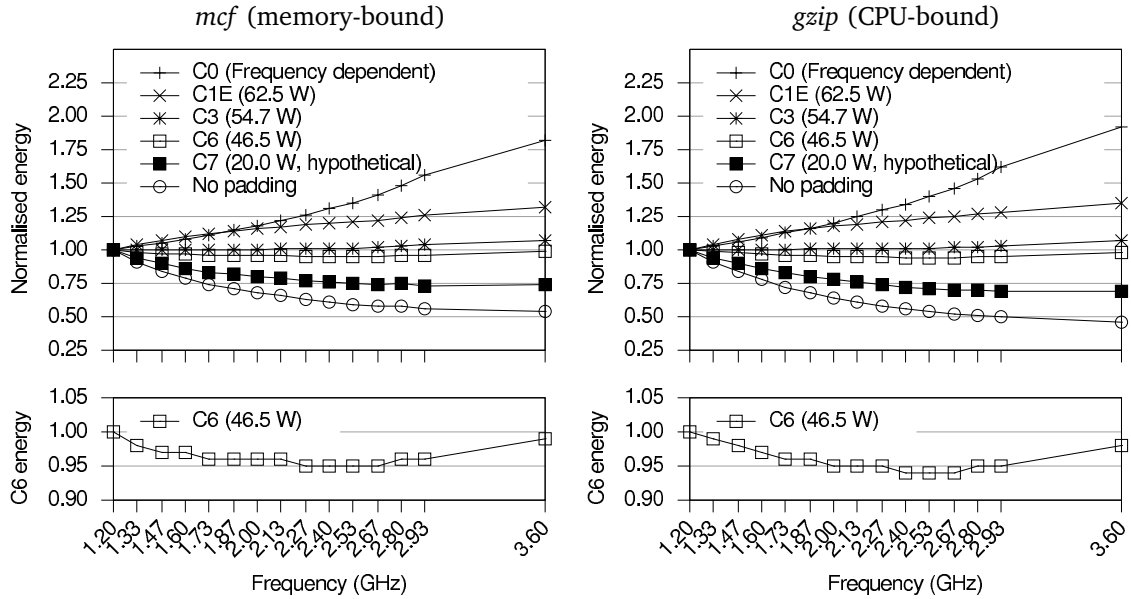


Figure 4.6: Energy consumption for *mcf* (left) and *gzip* (right) when padded for idle energy with various C states for *mcf* on the Core i7. This approach assumes a single idle state transition at benchmark completion. Data shown is normalised to the minimum frequency. 3.60 GHz is a TurboBoost frequency. The zoomed section shows that using C6, energy efficiency can be improved by up to 5 % if the CPU frequency is reduced.

4.3.3 Padding with different C states

Given that these systems have multiple C states, we can now apply the same padding methodology, but using the power drawn in each of the C states to calculate the idle energy. Figure 4.6 shows energy consumption normalised to the minimum frequency for *mcf* and *gzip* on the Core i7. Each line represents a different C state used to *pad* benchmarks that are executed at high frequencies with idle energy. The bottom line in the graph shows energy consumption when a hypothetical 0 W idle mode is used (i.e. no padding) and the top line represents padding with C0 idle energy, the lightest C state, in which the power drawn depends on the CPU's frequency. The lines in between represent padding with the various C states (C1E, C3, C6 and the hypothetical C7).

These results suggest that with the light sleep states (C0, C1E, C2, C3), the frequency at which energy consumption is minimised is always the minimum frequency. With the deepest sleep state currently implemented on the Core i7 (C6), the trade-off essentially disappears. The energy-frequency relationship is almost flat, with only a marginal energy consumption improvement of approximately 5 % for some mid-range CPU frequencies. We also show a line which represents a hypothetical idle state, C7, which has roughly half the power consumption of C6. In this case, the maximum frequency becomes the energy-optimal operating point.

4.3.4 Discussion

Padding shorter-running (high CPU frequency) benchmarks with idle energy allows system-level energy consumption at different CPU frequencies to be compared fairly. Using this methodology, Figure 4.3 showed that total system energy consumption could be reduced at low CPU frequencies on the Shanghai platform. However, the system's *energy-delay product (EDP)* was significantly impacted at the low CPU frequencies due to the increase in runtime as shown in Figure 4.2.

Figure 4.6 showed that different C states create very different energy consumption characteristics when padding with idle energy. In most cases, energy consumption is minimised either at the lowest, or highest CPU frequency. However, if the C6 C state was used, we observed a meagre reduction in energy consumption for both CPU-bound and memory-intensive workloads. We showed that with a hypothetical C state (C7) with half the power draw of the deepest available C state (C6), that energy consumption would be minimised at the highest CPU frequency. It is not unreasonable to expect that we will see such a scenario in the near future.

However, this padding methodology is still incomplete because it does not account for more frequent idle state transitions that occur when *bursty* workloads create slack-time. In the next section, we address this issue by introducing slack time into SPEC workloads and analysing several real-world workloads that inherently cause slack time in a system.

4.4 Bursty/Periodic Workloads

In the previous two sections, the workloads being analysed did not allow the CPU to idle during the execution phase of the benchmark. Here, the term *idle* means that there is no task to execute on the CPU, in contrast with the CPU executing a memory-bound instruction stream which causes it to stall waiting for operands. When the CPU is idle and there are no real tasks for the OS to schedule, the *idle thread* is scheduled instead. In a naïve implementation, the idle thread simply spins in a tight loop. However, as described in Section 2.3.2, modern CPUs often implement several low-power idle modes (C states), which the OS may invoke to reduce the CPU's power consumption while idle.

Some workloads are inherently bursty, causing *slack-time* in a system. A good example of such a workload is MPEG video decoding and playback. Groups of frames must be decoded before a deadline when they must be displayed to the viewer. If the decoding task is completed before the deadline, the CPU may be allowed to enter an idle state. Another example is a web-server responding to requests for web pages. The time between requests is variable with some random distribution, resulting in bursts of work for the CPU to complete before returning to an idle state.

4.4.1 Artificially introducing slack-time into SPEC workloads

The approach taken to padding for idle-energy (as was done in Section 4.3) is too simple. Applying this methodology assumes that there is only a single state-transition at the end of the benchmark execution. Workloads which create slack-time will allow the CPU to enter and exit idle modes much more frequently. Furthermore, different C states will have different costs, depending on what actions must be taken before entering and exiting the C state. For example, for a core to enter the C6 C state on the Core i7, its L1 and L2 caches must be flushed to the shared L3, and the core's supply voltage must be turned off. The magnitude of these costs will also be workload dependant, for example a workload that has a large cache footprint will have a larger exit cost as cache-lines are refilled from upper levels of the memory hierarchy when the core wakes up from sleep.

As discussed in Section 4.1, the SPEC workloads are normally compute-intensive. They do not allow the CPU to idle during execution. However, to force the CPU to transition between C states more frequently, we can artificially introduce slack-time into these workloads. This allows us to analyse the overheads resulting from frequent C state transitions.

4.4.1.1 C state transition overheads

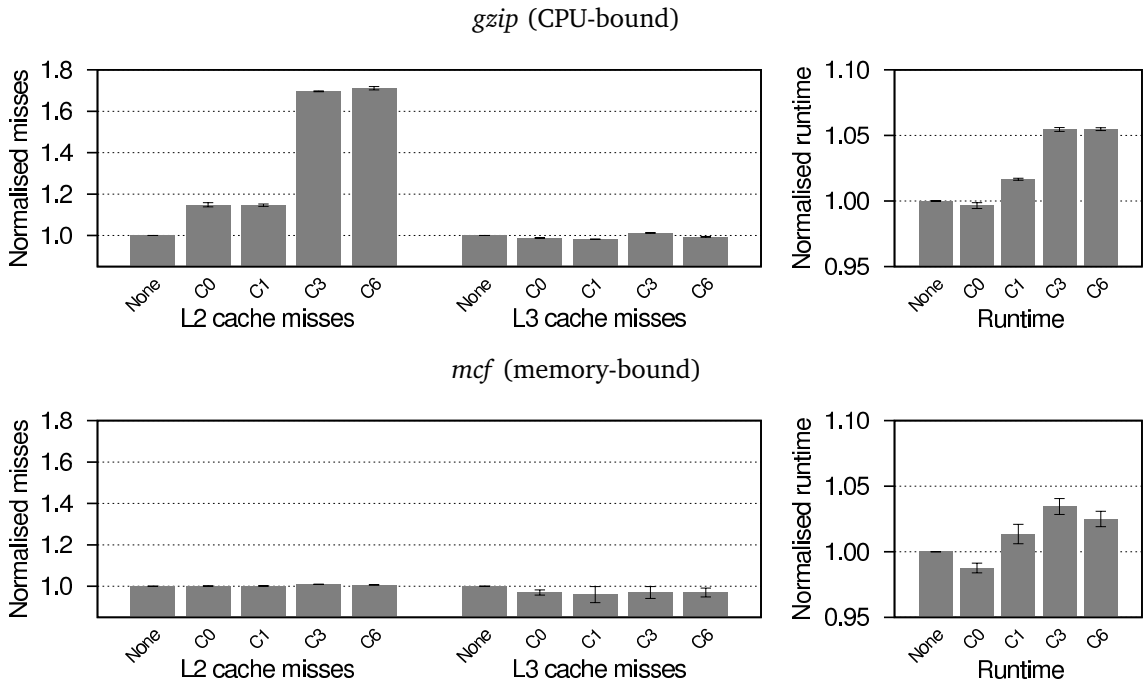


Figure 4.7: Costs associated with transition between C0 and deeper C states every 10 ms for two SPEC workloads, *mcf* (memory-bound, bottom) and *gzip* (CPU-bound, top) on the Core i7.

To measure the overheads associated with frequent transitions between C states on the Intel CPUs, we designed a framework using the *performance-monitoring unit* (PMU) allowing us to introduce

slack-time into arbitrary benchmark workloads, providing opportunities for the CPU to enter a C state.

To determine what C state transition frequency should be tested, we analysed the C state transition characteristics of a real-world MPEG playback workload. We found that the CPU was able to enter a C state approximately every 10 ms. Additionally, OSes often use 10 ms as the time-quantum for time-sharing CPU resources. Therefore, a 10 ms time-quantum is used in this analysis.

We used the PMU to generate interrupts at fixed instruction intervals. A timer is started at the beginning of the benchmark and when an interrupt occurs at the end of an interval, if the timer has counted less than 10 ms, the CPU is allowed to enter a C state for the remainder of the 10 ms time-quantum. When the CPU awakens, the timer is restarted and the next time-quantum begins.

We chose the number of instructions to execute in an interval such that the work was completed close to the deadline when the CPU was at its minimum frequency.

As a result, the SPEC workloads can be transformed into periodic tasks with soft deadlines, much like an MPEG video or audio playback task. If the time taken to complete the work in a time-quantum is small, then there is slack-time in the system and the CPU is able to enter a C state. If the time taken to complete the work in a time-quantum is large, then either the deadline is missed or there is only a small amount of slack-time in the system. The consequences for missing deadlines is task specific, however we tuned the system such that very few deadlines were missed.

At high CPU frequencies, there was significant slack-time in the system. We found that a small number of deadlines were missed at the low CPU frequencies, however, we measured the time accumulated from these missed deadlines and padded all benchmark executions out to the maximum execution time we observed at the lowest frequency. The runtime at the highest frequency was always within 1 % of the lowest frequency, hence the amount of padding required was small.

To allow us to choose which C state the CPU should use when idle, we implemented a new *userspace governor* for the Linux cpuidle framework. The C state is specified using the sysfs interface to cpuidle.

We measured the overhead of our framework to be minimal. However, firstly, we run the benchmarks within the framework, but skip the `SLEEP` system-call. All other values are normalised to this 'None' bar in the graphs. This allows us to measure the overheads of the C state transition alone excluding any overhead introduced by our framework.

Figure 4.7 shows the overheads in L2 and L3 misses and runtime associated with transitions between C0 and deeper C states at intervals of 10 ms on the Core i7 at 3.6 GHz. We present results from two SPEC CPU workloads, *gzip*, a CPU-bound workload (top) and *mcf*, a memory-bound workload (bottom).

As shown, the use of C states has variable overhead depending on the type of workload—*gzip*, has a much higher L2 miss-rate and slightly elevated L3 miss-rate when the deeper C states are used. This

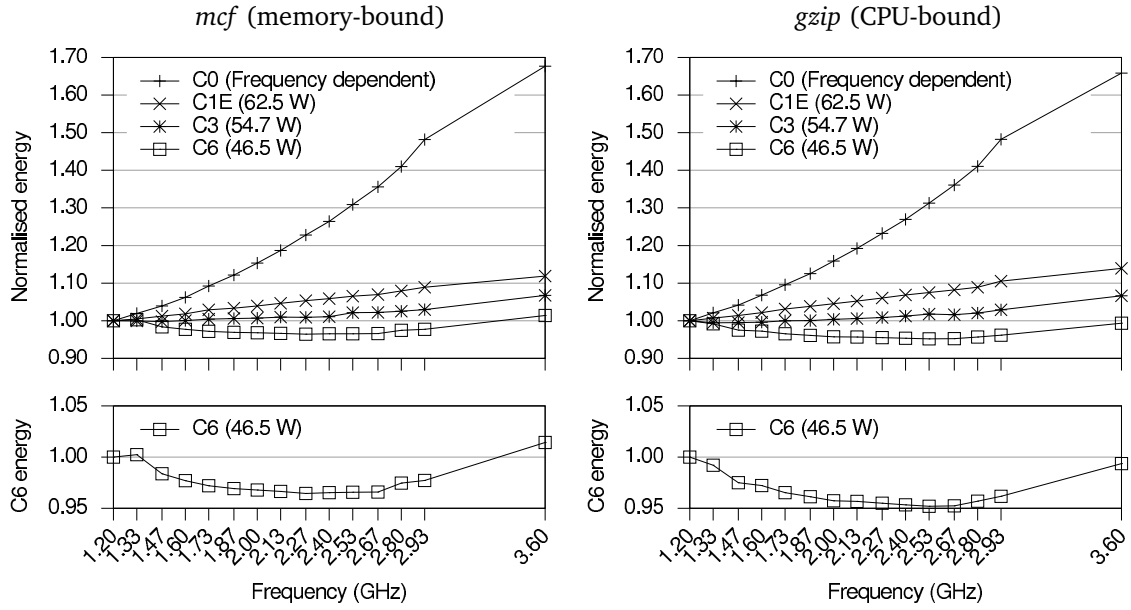


Figure 4.8: Energy consumption for *mcf* (left) and *gzip* (right) on the Core i7. In contrast to Figure 4.6, this data includes costs for idle state transitions every 10 ms. Data shown has been normalised to the minimum frequency. 3.60 GHz is a TurboBoost frequency. The zoomed section shows that using C6, energy efficiency can be improved by up to 5 % if the CPU frequency is reduced.

is because the L2 cache must be flushed to the L3 cache before the core may enter the C3 or C6 C states. In contrast, *mcf*, has almost no increase in L2 or L3 miss-rate when deep C states are used.

The caches in the Core i7 are *inclusive*. This means that cache lines stored in the L1 cache are a subset of those stored in L2, and those stored in the L2 cache are a subset of those in the L3 cache. Therefore, unless the data stored in the L1 or L2 cache is modified, the cost of flushing the caches on entering a deep C state will be small. The L1 and L2 caches must still be reloaded when a core wakes up, which is what causes the overhead in L2 cache misses.

However, as the graphs show, the large increase in L2 cache misses for *gzip* results in a marginal 5 % increase in runtime—the cache and memory performance of the Core i7 platform is sufficient to hide most of the costs associated with the entry and exit of the deep C states states when they are invoked at this rate.

The runtime overhead for *mcf* must be caused by something other than L2 or L3 cache misses. For example, the deep C3 and C6 sleep states require that the core voltage is switched off upon C state entry and on upon C state exit. This will cause some latency.

Figure 4.8 shows energy consumption for the same two workloads under varying CPU frequency. These graphs are similar to Figure 4.6, however, in these figures, slack-time has been introduced into the workloads at 10 ms intervals using the framework described above. The zoomed section shows that when the C6 C state is used, energy consumption for *mcf* can be reduced by up to 3 % if

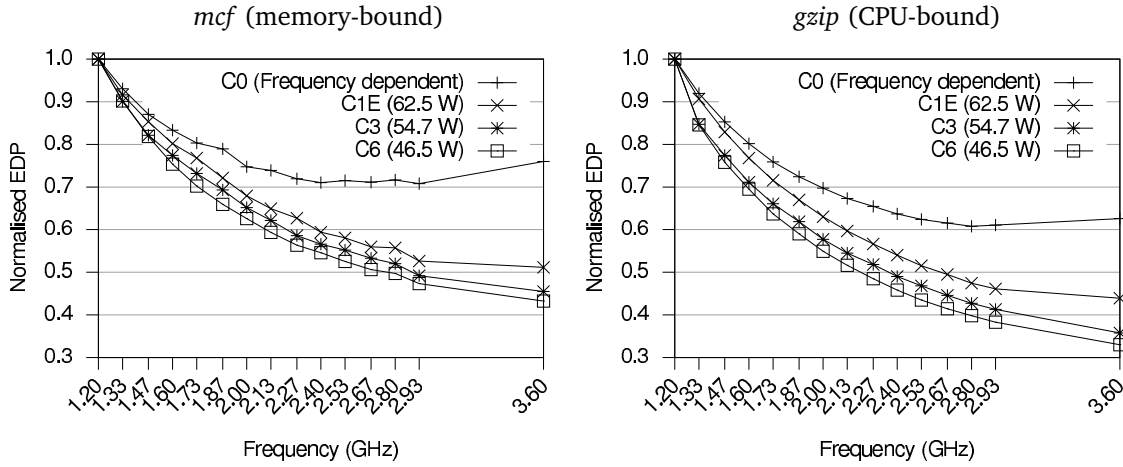


Figure 4.9: EDP for *mcf* (left) and *gzip* (right) on the Core i7 when slack time is introduced at 10 ms intervals. Data shown has been normalised to the minimum frequency. 3.60 GHz is a TurboBoost frequency.

the CPU frequency is reduced. Comparing the line for C6 to Figure 4.6, where approximately 5 % energy could be saved, the energy consumption overhead for the C6 state with *mcf* is approximately 2 %. Similar results are observed for *gzip*.

However, at reduced CPU frequencies, runtime increases dramatically and, as shown in Figure 4.9, the *energy-delay product* is still minimised at the maximum frequency in each case except when TurboBoost (3.60 GHz) is used with no C state (C0).

The Atom implements dynamic cache resizing for its L2 cache, as described in Section 2.3.4. Normally, the Atom's L2 cache is dynamically resized by a state-machine within the processor hardware based on C state residency. However to force deterministic results, we force the cache to be resized to zero when entering the C6 state, by using a *hint* with the MWAIT instruction. This effectively forces the L2 cache to be flushed immediately upon entry to the C6 C state. We measured similar overheads for these two workloads on the Atom.

Our framework was designed for x86 only, hence it could not be used on the OMAP4430-based Pandaboard. However, our experiments with MPEG playback on the Pandaboard in the next section show that the entry and exit costs of C states on the OMAP4430 are considerably more expensive than the Core i7. This highlights a difficulty when designing general-purpose power-management algorithms such as the *menu* and *ladder* cpuidle governors, as the trade-offs are often very different between platforms.

4.4.2 MPEG playback workload

Playback of MPEG video streams is a common workload for desktop-, laptop- and embedded-class systems. This requires a certain number of video *frames* to be decoded and displayed in a fixed

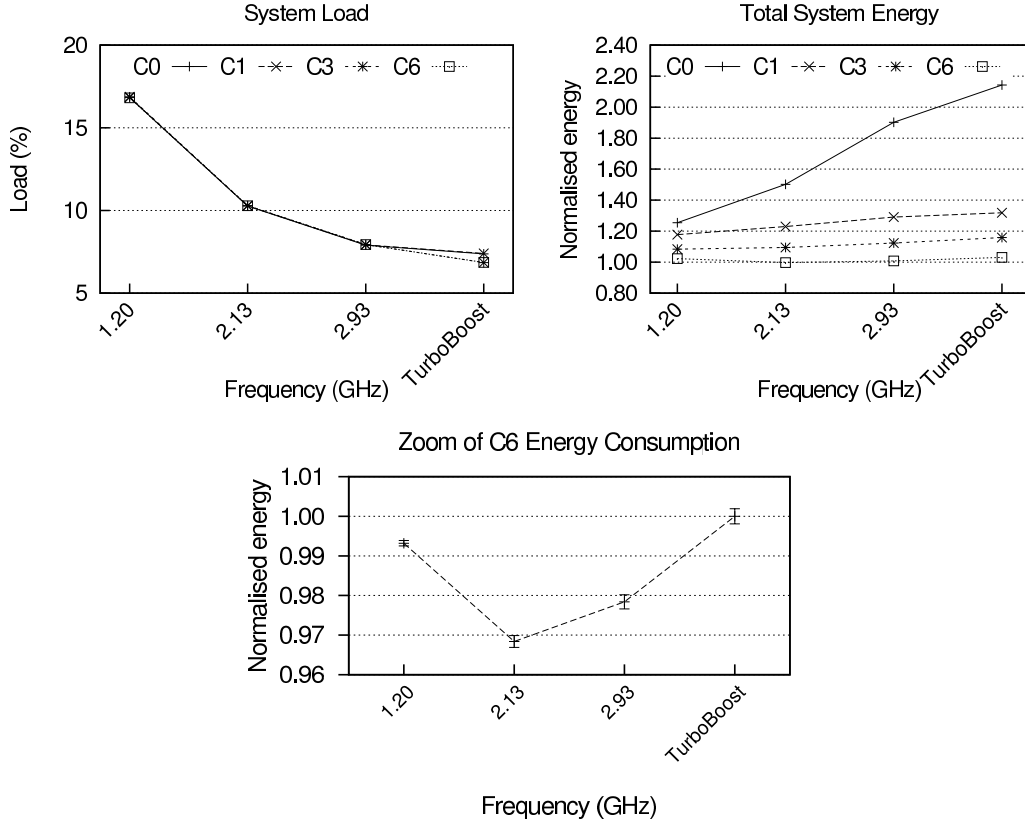


Figure 4.10: System load (top left) and normalised energy consumption (top right) for playback of a high-definition MPEG video stream on the Core i7 under various frequencies. Energy data has been normalised to 2.13 GHz using C6, which was the case where minimum energy consumption was observed. The bottom graph shows a zoomed version of the C6 line in which the data has been normalised to the TurboBoost point.

period of time in order to maintain the correct *frame-rate*. This is necessary for several reasons: firstly, the user can visibly detect skipping if the frame-rate is too low, and secondly, synchronisation between audio and video must be maintained, otherwise the user can detect lag between what they see and what they hear. This workload is essentially a soft real-time task—there is a deadline for when a group of frames must be decoded and the CPU can idle if the work is completed before the deadline. This is an interesting case for *slack-time minimisation* with DVFS. The CPU can be slowed down such that the level of performance is just above that required to decode the frames before the deadline.

We use *mplayer* a Linux utility designed to decode and display video streams to a user. If *mplayer* determines that the audio and video streams are losing synchronisation it will *drop* frames, i.e. it will sacrifice smooth playback in order to keep the two streams synchronised. *Mplayer* is an open-source program, which we modify to keep a count of the total number of dropped frames.

First, we use the Core i7 to play a high-definition (h.264, 1080p) video—a trailer from the recent *Wall Street* movie. The first 60 seconds of the movie are played while measuring the energy

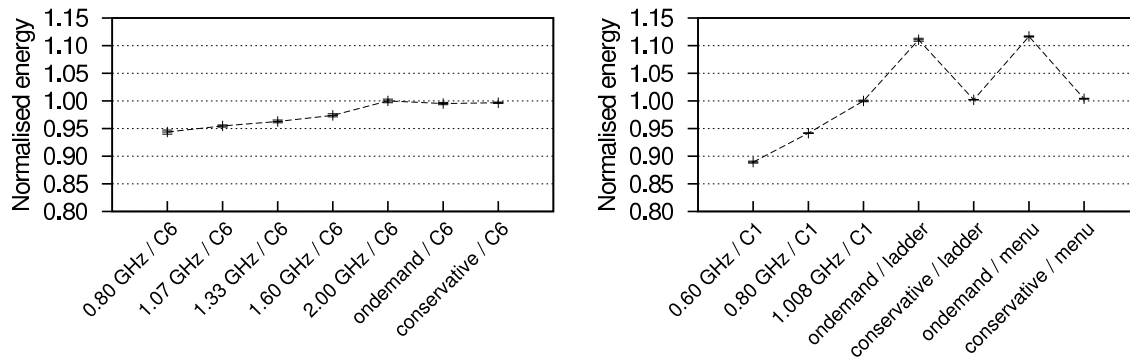


Figure 4.11: Left: Energy consumption for playback of an MPEG video stream on the **fitPC** (Atom) when the C6 C state is used. Values are normalised to the 2.00 GHz / C6 point. Right: Energy consumption for playback of an MPEG video stream on the **Pandaboard** (OMAP4430). Values are normalised to the 1.008 GHz / C1 point.

consumed by the total system. Figure 4.10 shows that at most 3 % energy is saved when the CPU frequency is reduced to 2.13 GHz. No frames were dropped or synchronisation lost on this system at any frequency. This workload does not tax even a single core of the quad-core Core i7 processor. Even at the lowest frequency, 1.20 GHz, there are still opportunities for the CPU to idle and enter a C state. Using DVFS gives only marginally better energy efficiency for this workload on this platform.

Next, we look at the Atom (fitPC) and OMAP (Pandaboard) platforms, the characteristics of which are shown in Table 4.2. Neither of these platforms were powerful enough to decode the high-definition *Wall Street* MPEG stream without dropping considerable frames to maintain video/audio synchronisation. For this reason, we use an h.264 MPEG stream with a reduced resolution and bit-rate, *The Elephants Dream* at 1024x576. The left graph of Figure 4.11 compares the energy consumed by the Atom Z550 based **fitPC** at various frequencies when the C6 C state is used. Similarly to the Core i7, marginal energy efficiency improvements (approximately 5 %) are observed when reducing the CPU frequency to 0.80 GHz. No frames were dropped at any of the CPU frequencies on this platform.

The right graph of Figure 4.11 shows the same results, but on the **Pandaboard**, based on the Texas Instruments OMAP 4430. We found that use of the deep sleep states (as shown by the points where the cpuidle *ladder* governor is used) on this platform has significant energy overhead, due to the high cost of L2 cache-misses resulting from the cache being flushed to main memory—energy consumption is increased by 10 % when these are used. However, energy efficiency can be improved by approximately 10 % with no noticeable reduction in frame-rate or user-experience when the CPU frequency is reduced to 0.6 GHz.

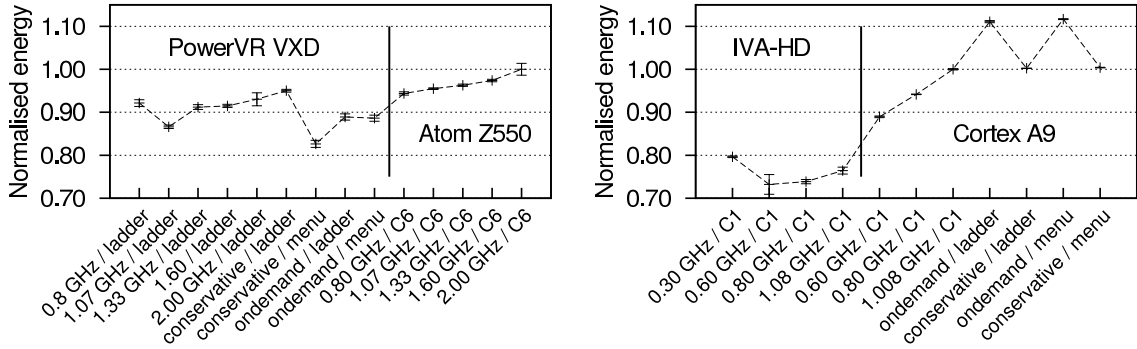


Figure 4.12: Left: Energy consumption for playback of an MPEG video stream on the **fitPC** (Atom Z550) when the PowerVR VXD MPEG decode unit is leveraged. Right: Energy consumption for playback of an MPEG video stream on the **Pandaboard** (OMAP4430) when the IVA-HD MPEG decode unit is leveraged.

4.4.3 Hardware acceleration of MPEG decode

Both the Atom Z550 and OMAP4430 have hardware accelerated MPEG decode units on the same die as the CPU cores. Using these units allows the main CPU cores to idle more frequently and for longer periods of time, because it is only required to transfer the data between storage and main memory. The accelerating units are connected to the main memory bus allowing them to perform *direct-memory-access* (DMA). This can reduce the energy consumption of the platform when playing MPEG video streams.

The Atom Z550 has a PowerVR VXD MPEG decode assist unit [Imgtec, 2011], which enables it to decode full HD MPEG streams, while the CPU itself cannot, due to performance constraints. In Figure 4.12, we show the same data as in Figure 4.11, but also include the total energy consumed by the fitPC when *The Elephants Dream* is played using *mplayer's* video acceleration API (VAAP) which uses the VXD unit. We found that using the *menu* cpuidle governor and the *conservative* cpufreq governor, energy consumption can be reduced by up to 17 %.

The OMAP4430's image and video accelerator (IVA-HD) unit is similar to the Atom's VXD unit. It provides hardware accelerated MPEG decode assistance. The right graph in Figure 4.12 compares the use of this unit with simply using the ARM Cortex A9 to decode and play the MPEG video stream. By using the IVA-HD and reducing the Cortex A9's frequency to 0.6 GHz, total system energy consumption can be reduced by up to 27 %.

4.4.4 Web-server workload

Serving web-pages is one of the most common server workloads. It is also inherently bursty, due to the random nature of web requests coming from the Internet. Servers must be provisioned to accommodate high request rates during times of high load, however, Barroso and Hölzle found that many of the web servers in Google's data-centers operate under-utilised most of the time. From

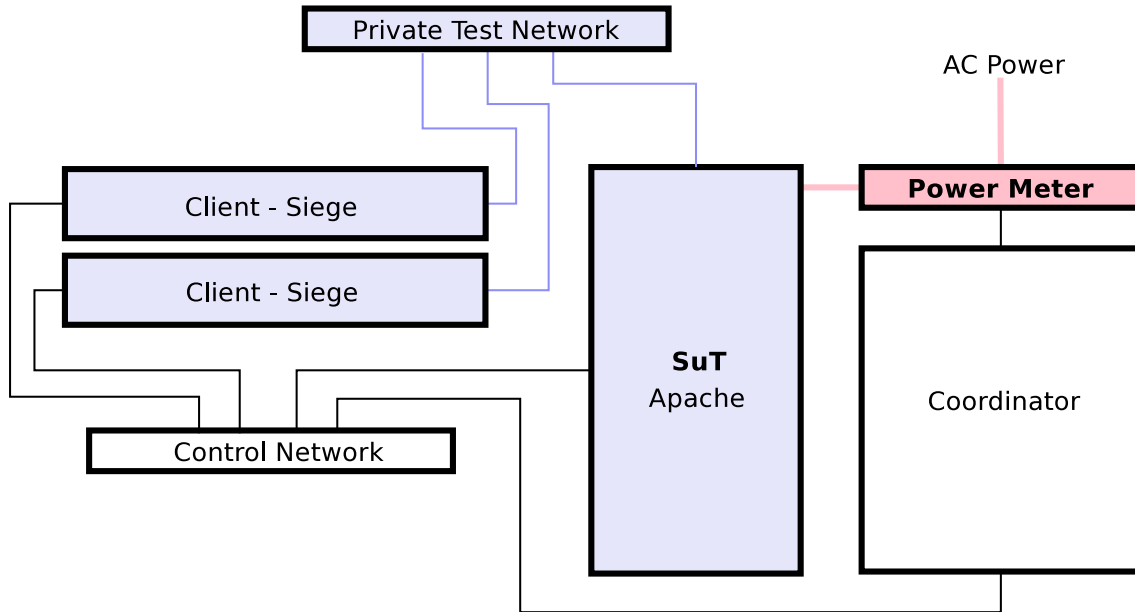


Figure 4.13: Topology of systems and network for the web-server benchmarks

an analysis of more than 5,000 servers, they found that the CPU utilisation of these servers was mostly between 10–50 % [Barroso and Hölzle, 2007]. They also discuss the differences in workload characteristics between laptop- and server-class systems, claiming that while laptops experience bursts of CPU utilisation then long periods of idleness, servers are rarely completely idle. However, the time-scale that they analyse is too coarse. As we have shown, many workloads allow the CPU to transition to idle states at the millisecond scale. To determine whether C states should be used for server workloads, we performed an analysis of a web-server workload.

The topology of our test framework is shown in Figure 4.13. The SuT is either the Core i7-based Dell Vostro 430s, the Atom Z550-based fitPC or the OMAP4430-based Pandaboard. None of these are server-class systems. However, the desktop is equivalent to some low-end servers that are often used in data-centres. The clients sending web requests to the SuT are two HP DL365 G5 servers, based on AMD Opteron processors. There are two networks, firstly a control network that allows set up routines to be initiated, and secondly, a separate private test network through which web requests and responses are sent between the clients and the SuT.

The web-server software we use is Apache 2.2, which hosts a copy of the ERTOS public website (<http://www.ertos.nicta.com.au>) which includes a combination of small-sized static HTML pages and larger PDF documents. We use the *siege* HTTP load generator to generate requests to the web-server software running on the SuT [Siege, 2010].

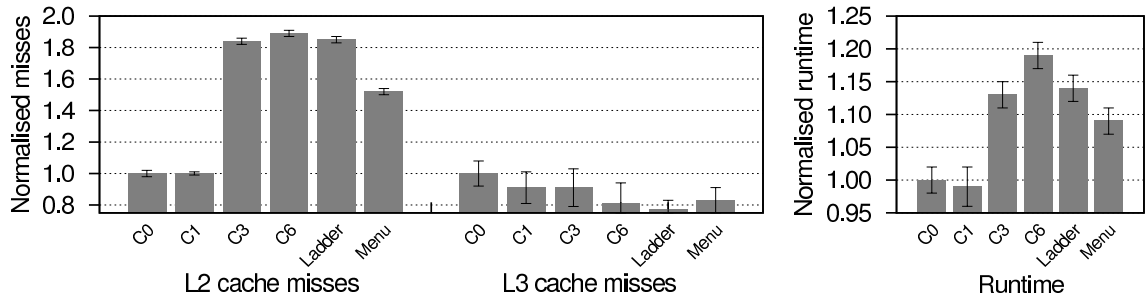


Figure 4.14: Overhead in L2 and L3 misses and runtime for different C states for the Apache web-server workload on the Core i7 at 2.93 GHz. All values are normalised to the C0 C state.

4.4.4.1 Results from the desktop

Each Siege client is configured to make up to 1,000 concurrent connections to the SuT. This allows the clients to generate approximately 90,000 requests in a 60 s period, giving a combined request rate of approximately 3,000 requests per second. As shown in the top left graph of Figure 4.15, the system load (CPU utilisation) generated by the two clients accessing the web-server is quite small, ranging between 12–28 % depending on the CPU frequency and C-state used. As shown, using the deeper sleep states has additional overhead.

Apache is a multi-threaded web-server, making use of all cores available on the system. The SuT is instrumented with a power meter at the power supply, measuring total system power which the coordinator uses to record the energy consumption. At the end of each run, each Siege client generates a report with data about the average response latency, total number of requests, etc. This data is recorded on the coordinator to be later analysed.

We observed that while *gzip*, *mcf* and *mplayer* all spend most of their time in userspace, the Apache workload spends much of its time in kernel, resulting from a high rate of system calls and network activity causing interrupts. Furthermore, when running the Apache workload, the CPU transitioned between idle states much more frequently. We observed state transitions approximately every 1 ms, ten times that of the MPEG playback workload.

The left graph in Figure 4.14 shows the overhead in L2 and L3 cache misses when using each of the available C states on this system while the Apache workload is running. Similarly to the CPU-bound *gzip* workload from Section 4.4.1.1, significant overhead in L2 cache-misses is observed when the deep C3 and C6 C states are used. The number of L3 cache-misses vary considerably due to the absolute values being relatively small and the error is quite high. Runtime is shown on the right and is the sum of both user time (time executing user-level code) and system time (time spent in system calls). When the deep C states are used, we see a significant increase in active CPU time.

A high performing web-server is not only characterised by the request throughput it can handle, but also by the average response latency. This is critical because users requesting web-pages are latency sensitive. Siege reports the average response time for each request over the length of the

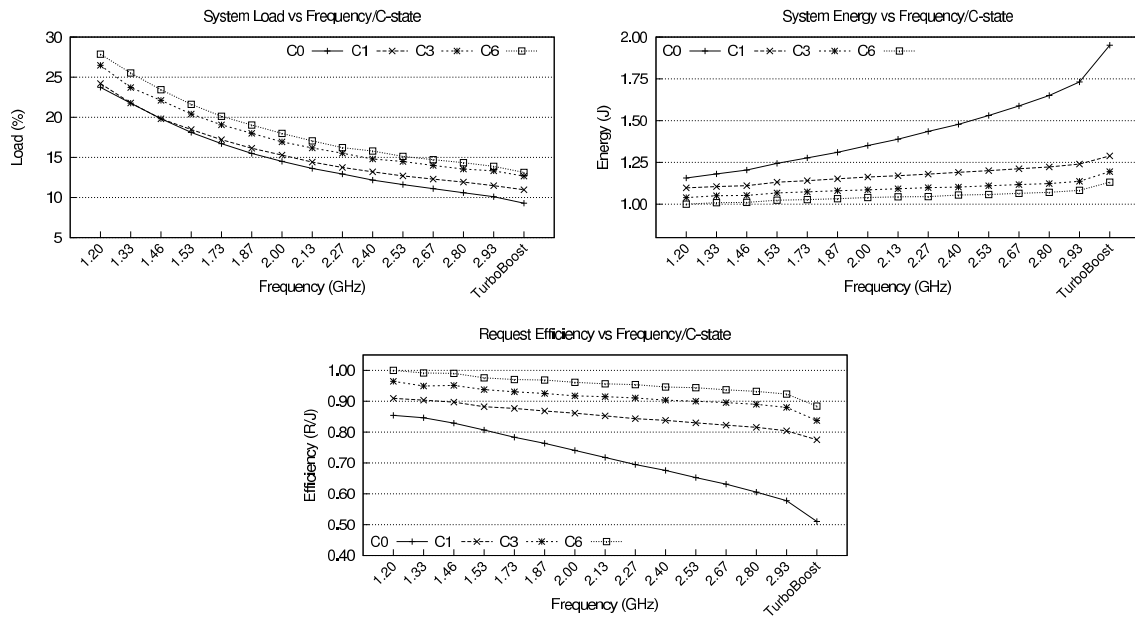


Figure 4.15: Top left: CPU load on the Core i7 (all four cores) when running the Apache workload under DVFS. Top right: System-level energy consumption of the Core i7 running the Apache web-server workload. All energy values are normalised to the lowest observed energy consumption at 1.20 GHz / C6. Since Apache is a multi-threaded workload, the TurboBoost frequency varies between 2.93–3.60 GHz. Bottom centre: request efficiency in requests per unit energy, normalised to again to 1.20 GHz / C6.

benchmark. In our tests, this was approximately 120 ms regardless of the CPU frequency or C state that was used.

As shown in the top right graph of Figure 4.15, energy consumption is minimised at the lowest CPU frequency (1.2 GHz), using the deepest C state, C6. The middle figure shows that energy-efficiency (in requests per Joule) is also maximised at 1.2 GHz using C6, giving a 12 % improvement compared to running at the maximum frequency using TurboBoost.

This workload is very different to the SPEC workloads which we introduced slack-time into in Section 4.4.1 and the MPEG workload used in Section 4.4.2. The Apache workload allows the CPU’s cores to enter sleep states at a much higher frequency than the 10 ms time-quantums we used in Section 4.4.1—we observed C state transitions every 0.75 ms. Additionally, opportunities for the processor to enter *package* C states are diminished with the multi-threaded Apache workload. Since requests are received sporadically and are evenly distributed to different cores, the time at which the cores can enter C states is not synchronised, hence the processor is very unlikely to be able to enter a package C state. This is partially responsible for DVFS still being beneficial for this type of workload on this system. Future research could address this by modifying the OS scheduler to delay request processing, and allow all the cores to enter the same state, thus allowing a package C state to be achieved. However, this approach could have an impact on response time and would need to be analysed in depth to determine whether it would have a positive effect on energy efficiency.

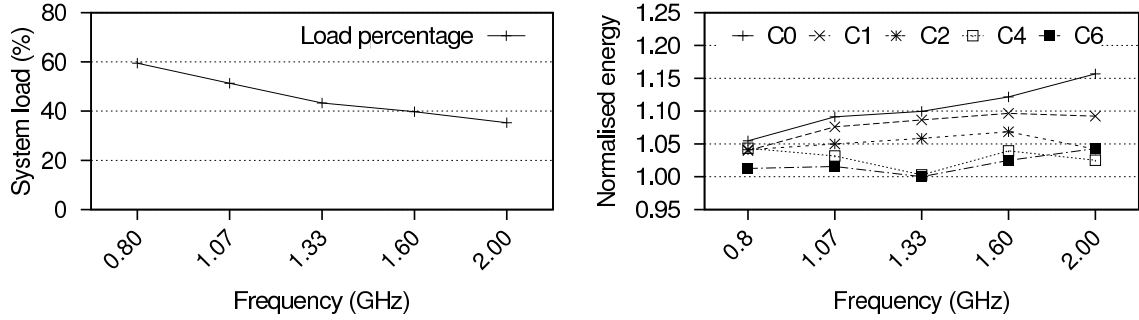


Figure 4.16: System load (left) and energy consumption (right) of the fitPC (Atom) when running the Apache web-server workload. Energy consumption values have been normalised to the minimum value observed at 1.33 GHz with C6.

4.4.4.2 Results from the fitPC

The use of embedded-class systems in the data-centre environment is currently a hot topic of research. The Atom-based fitPC is an example of such a system. However, the performance of this system does not match the Core i7-based desktop. In order to generate a reasonable load, we had to reduce the number of concurrent connections 1,000 to just 75 from each Siege client. This resulted in system load ranging from 60–35 % and a request rate of 276 requests per second at 2.00 GHz. Additionally, the fitPC has only a single network interface. Therefore, we use the private test network for control as well as for data-transfer between the server and clients. Control messages are only sent and received before and after the benchmark, hence there should be no additional overhead.

Figure 4.16 shows system load (left) and total system energy consumption (right). On this platform, energy consumption was minimised at a different frequency depending on the C state used. If C4 or C6 were used, the system energy consumption was minimised at 1.33 GHz. Energy-efficiency (Joules per transaction) was also minimised at this setting. On this platform, reducing the frequency resulted in a slight drop in request rate, up to 6 % at 0.8 GHz. Response latency also varied with CPU frequency, from 10 ms at 2.0 GHz to 40 ms at 0.8 GHz. However, this is lower than the response rate observed on the Core i7-based desktop.

4.4.4.3 Results from the Pandaboard

Similarly to the fitPC, the OMAP4430 processor on which the Pandaboard is based is incapable of handling the same load as the Core i7. Therefore, we reduced the number of concurrent connections for each client from 1,000 to 100, resulting in a load between 68–29 %. Like the fitPC, the Pandaboard has only a single network interface connected to the processor via USB. Hence, like with the fitPC, we use the private network for both control and data.

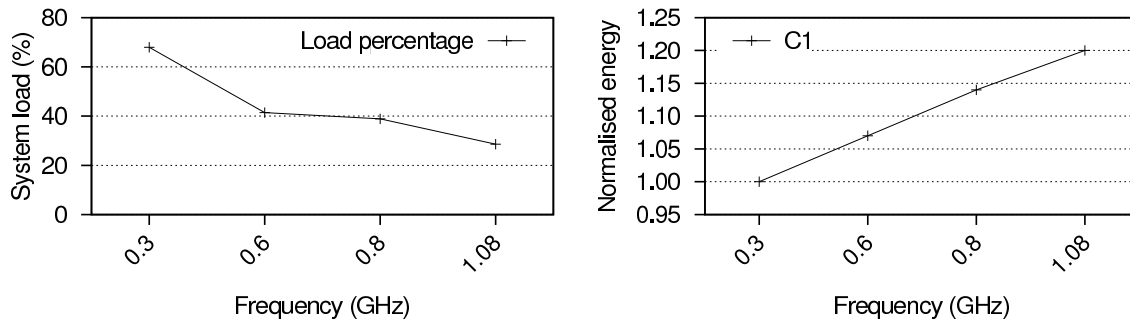


Figure 4.17: System load (left) and energy consumption (right) for the Pandaboard (OMAP4430) when running the Apache web-server workload. Energy consumption values have been normalised to the minimum value observed at 0.3 GHz.

Figure 4.17 shows system load (left) and system energy consumption (right) for the Pandaboard when running the Apache web-server workload. On this system, the 100 connections from each client generated a request rate of approximately 305 transactions per second. However, at 0.3 GHz, this was reduced by 15 % to approximately 260 transactions per second, as the system was incapable of sustaining the necessary throughput. Total energy consumption was minimised at 0.3 GHz. However, energy-efficiency (in Joules per transaction) was minimised at 0.8 GHz. Request latency was approximately 120 ms at all frequencies, the same as for the Core i7.

We show only results from the C1 state, due to limitations in the implementation of C state entry procedures in the Linux kernel which require the second core to be disabled if the OS wishes to use deeper sleep states. The OMAP4430 has complex clock and voltage domain control systems, which have complex inter-dependencies. This makes certain C state combinations (for multiple cores) impossible. In contrast, entering a C state on an Intel processor requires only issuing an `MWAIT` instruction—the processor handles all of the constraints internally.

The Pandaboard is a cutting-edge development system, and, as such, some parts of the system may be sub-optimal. For example, the network interface, being USB-connected, relies on the USB sub-system, which also must handle access to the SSD storage device where the web-server files are located. Furthermore, support for the OMAP4430 processor in the Linux kernel is still very experimental. The Pandaboard has only been commercially available since December 2010, and it is one of the first platforms based on the multi-core ARM Cortex A9 processor.

4.4.5 Discussion

Bursty workloads are very different to compute-intensive workloads. This is primarily because they allow the processor to idle during slack time that is created because of under-utilisation.

When comparing the results obtained from the padding methodology to those obtained when slack-time was artificially introduced into SPEC workloads suggested that there was little overhead in energy consumption from C state usage when the transition frequency was the order of 10 ms.

Furthermore, as was found with the Apache web-server workload, high C state transitions on a lightly loaded system seemed to have little impact on throughput and response latency. For the Core i7-based desktop, total system energy consumption was minimised by running at the lowest frequency using the deepest sleep state. The multi-threaded and bursty nature of the Apache workload means that the processor is unlikely to be able to enter a package C state, thus would not achieve such low power draw when idle. We believe that this fact explains why DVFS is still beneficial for this workload on this system.

Energy efficiency in Joules per transaction was also minimised using these settings. The fitPC and Pandaboard could not handle the same request rate as the desktop, because they are generally lower performing systems. On the fitPC, total system energy consumption and energy per transaction were minimised at a mid-range frequency, but still using the deepest C state. However, on the Pandaboard, system energy consumption was minimised at the lowest frequency, 0.3 GHz, but energy per transaction was minimised at 0.8 GHz.

The quad-core Core i7 achieved the best energy efficiency (Joules per transaction) out of the three platforms for the web-server workload. Despite this being a desktop-class system, this is an interesting finding considering the current market direction toward using embedded-class processors in the data-centre. However, the embedded systems we used were in no way optimised for this sort of workload. The network interfaces and platform design principles used for their construction may have influenced this outcome.

4.5 Conclusions

Benchmarking with the aim to compare total system energy consumption requires sound methodology. As we have shown, using compute-intensive workloads (which don't allow the CPU to idle) does not cover cases where C state transitions occur at a high frequency. We showed via artificial introduction of *slack-time* that there are costs associated with C state transitions, which, depending on the workload and the system, can influence energy efficiency. On the Core i7, memory performance is sufficient to hide many of the costs of these transitions, but on other systems, this will not necessarily be the case.

Furthermore, the basic methodology that has been used in the past to compare the energy consumption of compute-intensive workloads under DVFS is incomplete, as it does not account for static energy consumption. Padding improves the methodology, but it still does not account for frequent C state transitions.

We addressed these issues by introducing slack-time into SPEC workloads. We found that system-level energy consumption could be marginally reduced by lowering the CPU frequency with DVFS.

Next, after analysing two real-world workloads, we found several interesting characteristics. Firstly, the playback of MPEG video streams presented a very light workload for the Core i7 processor.

Again, we found that system-level energy consumption could be reduced marginally by lowering the CPU frequency. Similar results are achieved on the embedded platforms. However, being single-threaded, when the single active core is idled, the processor is able to enter a very low-power *package C* state (as all cores will be idle). This makes DVFS even less likely to achieve energy savings.

Secondly, after analysing the MPEG workload on the fitPC and Pandaboard, we found that the system-level energy consumption of these systems could be reduced when using the hardware acceleration provided by the Atom's PowerVR VXD MPEG decode unit and the OMAP4430's IVA-HA unit.

We found that the web-server workload differs from the MPEG workload in several ways. Firstly, Apache causes significantly higher numbers of C state transitions. Furthermore, the Apache workload has a high system-call rate and spends a significant amount of time in the Linux kernel performing network operations. Secondly, being multi-threaded and network oriented, the processor is unable to enter the very low-power *package C* state, which is achieved only when all cores enter the same C state.

As a result, we found that the system-level energy consumption of a lightly loaded web-server based on the Core i7 could be improved by approximately 15 % when the CPU frequency is reduced and the deep C6 idle state is used. This also improved energy efficiency in terms of Joules per transaction. We also found that the reduction in CPU frequency had no measurable impact on either the throughput or the response latency of the web-server.

We also analysed the Apache workload on the Pandaboard and fitPC and found that similar energy efficiency improvements could be achieved by reducing the frequency. However, the total throughput achieved on these systems was much lower than the Core i7, resulting in much greater energy consumption per transaction. The Pandaboard is a development platform with a USB network interface, which we believe was a factor in the low throughput that was achieved.

In conclusion, the decisions surrounding energy management are non-trivial. For workloads that result in high system utilisation, reducing the CPU frequency using DVFS does not result in significant energy savings. However, for lightly loaded systems even with high a C state transition frequency, it can reduce energy consumption and has the potential to improve energy efficiency significantly.

The trade-offs presented by power-management mechanisms have changed significantly over time. In the next chapter, we look closely at several trends which we believe are influencing the effectiveness of power-management mechanisms.

Chapter 5

The Laws of Diminishing Returns

Throughout this thesis, we have highlighted several trends which we believe have contributed to the reduced effectiveness of DVFS on modern processors. These include:

- scaling of silicon transistor feature sizes;
- increasing memory performance;
- improved sleep/idle modes;
- addition of asynchronously clocked caches and memory-controllers; and
- the increasing complexity of multi-core processors.

These trends are discussed in greater detail below.

5.1 Transistor Scaling

As transistors are scaled to smaller feature sizes, dynamic power consumption is considerably reduced, because all of the voltages (V_{DS} , V_{GS} and V_{th}) are scaled as well. This reduces short-circuit current and gate charge/discharge current. However, as discussed in Section 2.2.3, weak-inversion leakage current increases exponentially with reduced feature size. The effect of this is an increase in static power draw as a percentage of total processor power draw. As DVFS is more effective at reducing dynamic power draw, this can significantly undermine its effectiveness at reducing total power draw.

Furthermore, as feature size is reduced, the maximum voltage that can be applied to a transistor's gate is also reduced. However the minimum voltage remains constant. This is because the band-gap of silicon dictates that the lowest threshold voltage that will allow a transistor to switch on is

approximately 0.6 V. The combination of lower maximum voltage and a fixed minimum voltage has the effect of reducing the dynamic voltage range. As DVFS relies on this dynamic range to achieve reduced power draw, transistor scaling can severely hamper the operation of DVFS. This is compounded by the quadratic relationship between P and V , as shown in Equation 2.2.

As shown in Table 4.1, the maximum voltage from the first generation to the second generation of AMD Opteron processors was reduced from 1.5 V to 1.35 V. The voltage of the third generation Opteron is the same as the second generation, however it is clocked at a much higher frequency, resulting in a higher relative voltage. A fourth generation Opteron clocked at the same frequency as our second generation (2.4 GHz) runs from a supply of only 1.1875 V. Furthermore, an example of an *ultra-low voltage* (ULV) Opteron clocked at 1.7 GHz runs at only 0.9625 V [Advanced Micro Devices, 2010, Model 4162 EE]. As AMD and Intel move to the 32 nm process node and smaller, voltages will be reduced even further, therefore limiting the range of voltages that DVFS can work with.

Esmailzadeh et al. explore the effects of transistor scaling by looking at pairs of Intel processors, one of which has undergone a *die-shrink* [Esmailzadeh et al., 2011]. Performing a die-shrink is the act of manufacturing a processor of one generation on a newer CMOS process, resulting in an identical processor, but constructed from smaller transistors. This, due to the reasons highlighted above, results in a processor with the same performance (for a given frequency) that consumes less power. They also found that this process results in significant reductions in energy consumption.

5.2 Memory Performance

As discussed in Chapter 3, increasing memory performance means that processor cores stall for fewer cycles when executing memory-bound workloads. As memory-bound workloads offer the greatest opportunities to reduce the CPU's frequency without significantly impacting performance, DVFS is most effective at improving energy efficiency with these types of workloads.

As manufacturers add more cores to traditional CMPs, memory throughput must be increased to handle many cores accessing main memory over a shared bus. However, workloads that are single-threaded but memory-bound get a much larger performance boost from the increased memory throughput, and become less memory-bound.

In Chapter 3, we compared two systems based on Intel processors, a Pentium-M-based laptop and a Core i7-based desktop, which were release approximately 5 years apart in 2004 and 2009 respectively. Our analysis showed that workloads which were extremely memory-bound on the Pentium-M laptop are much less so on the Core i7 desktop.

5.3 Improved Idle Modes

As discussed in Chapter 4, recent processors are increasingly using multiple levels of idle states (C states). Previously, processors either had no idle states, or relied on the OS to slow the processor down (using DVFS) to reduce power consumption when idle. Intel's recent *Nehalem* and *Sandy Bridge* architectures have per-core power gating, allowing a single core of a CMP to be effectively *switched off*, reducing both dynamic *and* static power consumption by eliminating leakage power.

To keep C state entry/exit overhead small, an *inclusive* cache-hierarchy can be used, like on the Intel processors we analyse. Using this type of cache hierarchy means that lower levels of cache (e.g. L1) contain a subset of cache lines stored in the higher levels (e.g. L2). Unless data is modified, flushing the low-level caches on C state entry is cheap. In contrast, AMD's Opteron processors use an *exclusive* cache design at L1 and L2, and a selectively inclusive L3 (i.e. some cache lines are kept in L3 if multiple cores have copies in L1 or L2). When an exclusive cache is flushed, all cache lines must be written out to the next level, potentially resulting in evictions from the upper level, adding to latency. However, as discussed in Section 4.3.2, these processors only implement a single C state, C1, which does not require any cache flushes on entry. An exclusive cache hierarchy has the benefit that data is not replicated, hence more data can be stored in the caches.

The ability to use multiple C states has resulted in new trade-offs for power-management, as different C states have different costs to enter and exit. These costs were analysed in Section 4.4.1.1. Our analysis showed that although the costs are variable, in most cases they are small and have little impact on energy consumption. As a result, these C states are causing the *race-to-halt* (i.e. *sleep*) approach to become more appealing.

5.4 Asynchronously Clocked Caches

Figure 5.1 shows the results of an analysis of three AMD Opteron processor die photos. The left bar shows the breakdown in die area for the first generation **Sledgehammer**, a single core processor fabricated at 130 nm. The middle bar shows the breakdown for a second generation **Santa Rosa** dual-core Opteron fabricated at 90 nm. Finally, the right-hand bar shows the breakdown for a third generation quad-core **Shanghai** Opteron fabricated at 45 nm.

As shown in Figure 5.1, AMD added a large third-level cache (6 MiB) within the Shanghai die. This cache occupies a large proportion of total die area and is clocked asynchronously to the four cores. Its frequency cannot be scaled at all, regardless of the frequency of the cores. This results in the L3 cache consuming a large amount of energy due to leakage.

The percentage of die occupied by the cores is approximately equal over the generations, despite the increase in core-count from one to two to four cores. This is made possible by the reduction in transistor size from 130 nm to 45 nm.

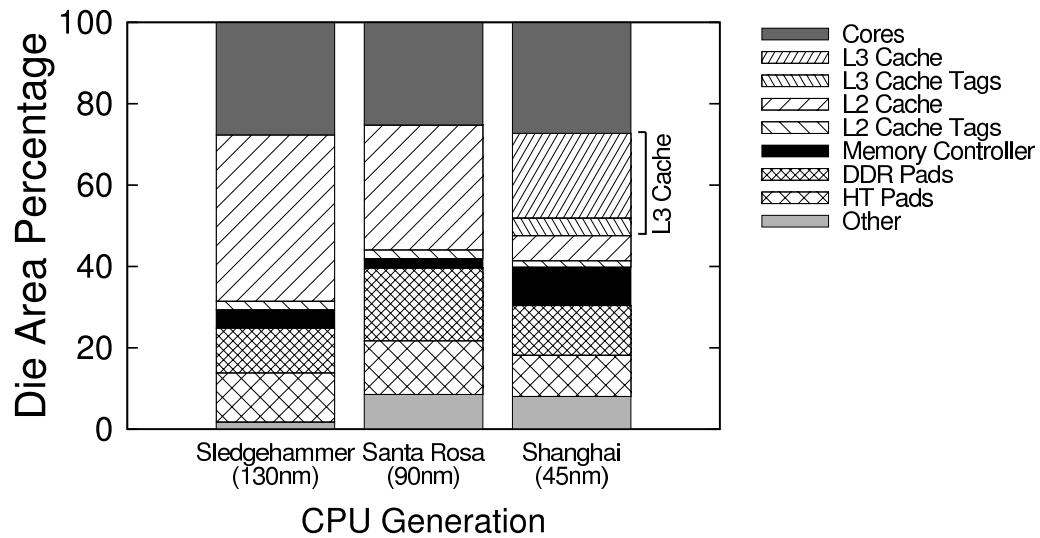


Figure 5.1: Die area breakdown of three generations of Opteron processors.

As discussed in Chapter 4, our analysis shows that the use of DVFS on the Shanghai Opteron-based system does not save energy and in fact, actually increases energy consumption for even the most memory-bound workloads.

5.5 Multi-core Processors

Recent CPUs are designed as chip-multi-processors (CMPs), meaning there are multiple cores on a single die. These cores usually share an L2 or L3 cache allowing fast communication and shared memory. However, this design topology gives rise to more complex power-management questions. Most significantly: should all cores be clocked at the same frequency and voltage (allowing *chip-wide* DVFS) or should the chip be designed to allow individual cores to be clocked and supplied separately (allowing *per-core* DVFS) ?

Isci et al. studied the effectiveness of chip-wide and per-core DVFS when constrained by a total chip-wide power budget [Isci et al., 2006]. They found that a global policy that was able to individually set each core's frequency performed better than chip-wide DVFS. However they did not take into account the cost of supplying each core with a separate voltage.

Existing CPUs are designed to be used with off-chip voltage-regulators (VRs) to supply power at a stable voltage over a wide range of currents. These consist of a controller (connected to the CPU) and some passive components such as inductors and capacitors. All of these components must be located close to the CPU to reduce parasitic inductance and capacitance [Ren et al., 2004].

In most cases, a CPU with chip-wide DVFS requires only one off-chip VR. However, in the case of

the AMD Shanghai Opteron which has an asynchronously clocked L3 cache and memory controller (see Section 5.4) a second VR may be required.

Off-chip voltage regulation becomes problematic when large numbers of cores require separate supplies, as would be the case with a CMP with individual voltage and frequency domains for each core.

An advantage of a single voltage domain is that current supplied by a single VR can, and must, be shared amongst all of the cores. When some cores consume less current or are powered down, current supply can be directed to other cores, as is done with Intel's TurboBoost feature. This is easy to achieve when a single VR is designed to handle the current supply requirements of all cores. However if a VR is designed to supply only one core, then it is limited by how much current it can supply (e.g. when the TurboBoost feature is enabled).

Furthermore, a CMP which is implemented with multiple frequency domains will require FIFO buffering for data transfer between each domain. This can create bottlenecks inhibiting performance.

Rotem et al. compare the effectiveness of chip-wide and per-core DVFS using an Intel Core 2 Duo based model and simulation of a 16-core CMP. They found that a single voltage and frequency domain performs up to 30% better than multiple domains for lightly-threaded workloads [Rotem et al., 2009]. They conclude that power delivery is a primary constraint for CMPs with a high core-count and found that the benefits of per-core DVFS are small when these constraints are taken into account. Furthermore, as described above, there are benefits to having only a single voltage domain where features such as TurboBoost require more power from the voltage regulator in order to boost the frequency of a single core.

Kim et al. describe a 3-level on-chip VR capable of performing voltage transitions in nano-seconds [Kim et al., 2011]. If implementation of these on-chip VR modules is commercially viable, it may be possible for manufacturers to provide per-core DVFS. However, for now, modern, commercially available CMPs implement chip-wide DVFS only.

5.6 Conclusions

This chapter has discussed several trends in processor design and manufacturing which have influenced the effectiveness of power-management mechanisms. Firstly, as Esmaeilzadeh et al. show, the *die-shrink*, achieved by constructing processors from smaller transistors, is one of the most effective ways manufacturers can reduce the power consumption of a processor [Esmaeilzadeh et al., 2011]. However, as we show in Chapter 4, this trend also has an effect on DVFS with reduced voltage range and higher static power.

Secondly, the effectiveness of DVFS has also been affected by the introduction of multi-core processors. In such processors, improved memory performance is required to accommodate the

many cores accessing a single shared block of main memory. This reduces pipeline stalling due to memory-intensive instruction streams and therefore, limits the usefulness of DVFS algorithms that take advantage of situations where a CPU core is wasting cycles waiting for operands from main-memory.

Thirdly, changes to/improvements in idle modes have also altered the effectiveness of DVFS with our analysis showing that low-power idle modes have small entry/exit costs which makes race-to-halt more effective.

Fourthly, asynchronously clocked CPU sub-components such as memory-controllers and L3 caches use large amounts of die area and transistor budget and therefore add a significant amount to static leakage power. This has the effect of diminishing the range of dynamic power consumption that DVFS can control.

Lastly, the hardware complexity required to provide individual cores with their own power supplies inhibits the implementation of per-core DVFS, hence most commercial CPUs have only chip-wide DVFS. This requires that all tasks running on a CPU be analysed together before making a decision to alter the CPU frequency.

These trends cause the energy-saving benefits of DVFS to diminish and will continue to into the foreseeable future.

Chapter 6

Summary and Conclusions

6.1 Summary

In this thesis, we aimed to determine which power-management mechanisms were the most effective at improving system energy efficiency. In 1994, Mark Weiser claimed that the Tortoise was better than the Hare, that it was better to slow the CPU down, rather than *race-to-halt*. Computer processors have progressed significantly since then, and the correct approach to take is no longer obvious.

Chapter 2 discussed the power-management mechanisms (including DVFS) that are available on most modern CPUs. As discussed, while many of these mechanisms have been the subject of prior studies, few have critically analysed the trends that dictate their effectiveness in real-world scenarios. Much of the prior work has tried to model the impact of DVFS on workload performance and processor power consumption. Several research frameworks were described, as well as the current power-management algorithms used in Linux.

Chapter 3 described the impact of increasing memory performance on the effectiveness of DVFS. We showed that certain architectural features, such as hardware prefetching, can significantly improve the performance of some workloads, but also make it difficult to characterise models that are accurate over a wide range of general workloads. We concluded that this is because of a lack of performance-counter events that give insight into the effectiveness of the hardware prefetcher. Furthermore, we showed that improved memory throughput reduces the memory-intensity of previously memory-bound workloads, resulting in fewer opportunities for the CPU to be slowed down using DVFS without significantly impacting performance.

In Chapter 4, we showed that idle modes are becoming much more effective at reducing power consumption when a processor is idle. We found that on recent systems, the costs associated with their use are small. Therefore, the deep sleep states should be used in most circumstances. We

analysed both compute-intensive (non-idling) workloads and those that create slack-time in the system, finding that even for workloads that have a high C state transition rate, the overhead in energy-consumption is still small. We found that DVFS could achieve only marginal improvements in energy efficiency of up to 5 %. In contrast, previous studies showed that savings of up to 30 % could be achieved in some cases.

We found that DVFS could be effective at reducing system-level energy consumption for systems running workloads with the following characteristics:

- low load (less than 30 % CPU utilisation),
- high interrupt and/or DMA rate (e.g. from a network interface)

An example of such a workload is that generated by the Apache web-server, for which we showed that DVFS could improve per-request energy efficiency by approximately 12 %. We found that even with a high rate of C state transitions, the overhead observed from these (i.e. increased latency) was small. Furthermore, we found that the reduction in CPU frequency had little effect on the throughput or response latency of the web-server. The positive benefits of DVFS in this workload scenario were found to be a result of the processor being unable to enter the very low power *package* C-state that is achieved only under certain circumstances, such as when all cores are idle at the same time and there is no outstanding DMA activity or incoming interrupt deliveries.

However, DVFS is not the only mechanism that manufacturers are using to reduce energy consumption. We showed that using hardware acceleration can reduce the energy consumption of two embedded systems (the fitPC and Pandaboard) for specific workloads like MPEG decode and playback.

Chapter 5 analysed the trends that are causing the effectiveness of DVFS to change. We discussed the effects of scaling transistors to smaller feature sizes, highlighting the exponential increase in static leakage power exponentially with smaller features. In contrast, dynamic power is reduced due to smaller gate capacitance and short-circuit current. This means that processors have a smaller range of dynamic power consumption, reducing the effects of DVFS on *total* power consumption. Scaling transistors down also reduces the range over which supply voltage can be varied which further limits the effectiveness of DVFS.

We further surmised that large, power-hungry, asynchronously clocked caches (like those found on AMD's Shanghai Opteron) are a key factor influencing the effectiveness of DVFS on platforms based on this particular processor. Large L3 caches require large processor die area and huge transistor budgets. These add significantly to static power consumption.

Finally, we discussed issues relating to the implementation of per-core DVFS and why manufacturers do not provide this level of control.

6.2 The Future of CPU Power Management

The trends which were analysed in Chapter 5 are not going away. Transistors will continue to shrink, with the ITRS predicting feature size will be as small as 9 nm within the next 12 years. In addition, the use of multi-core processors will continue and core-count will likely rise.

However, we may start to see commercially available asymmetric multi-processor (*AMP*) systems in the future which will raise different power-management questions. At the same time, memory performance will continue to improve in order to accommodate the growing number of cores, making single-threaded workloads much less memory-bound.

Multi-core processors and AMP systems create difficult decisions for OS task scheduling. Studies have already been undertaken in this area. Fedorova et al. implement a comprehensive scheduler for AMP systems capable predicting the benefits of running a task on the different types of cores on an AMP system [Fedorova et al., 2009].

Performance asymmetry using DVFS is a common way to emulate an AMP. Intel has already found that the TurboBoost mechanism is an effective way of improving single-threaded workload performance [Charles et al., 2009; Rotem et al., 2009]. However, whether this improves energy efficiency is again, critically workload and system dependent.

Researchers at Intel are also studying different processor architectures based on *message-passing*. Rather than using global cache-coherence protocols for shared memory, a core in a message-passing processor wishing to share data with another core must explicitly pass data to the other core in a message. This has resulted in designs such as the *single-chip cloud* (SCC), a 48-core CMP [Howard et al., 2010]. More than just a new design topology, this processor also implements DVFS *islands* where small groups of cores can operate at a frequency independent of other groups. The message-passing queues act as buffers between different frequency domains, making this relatively easy to implement when compared to a design based on global shared memory.

6.3 Future Work

6.3.1 DVFS on hardware acceleration units

The hardware acceleration units described in Section 4.4.3 themselves often implement DVFS. For example, the IVA-HD unit on the Texas Instruments (TI) OMAP4430 *system-on-chip* (SoC) is designed to decode many different types of MPEG streams, including those with variable bit-rates. The full processing requirements of this unit are often not required when decoding low bit-rate streams and, as a result, the frequency could be reduced to improve energy efficiency. Unfortunately, the firmware that runs on these hardware accelerators is often proprietary and not available to

analyse, as is the case with the TI IVA-HD. However, this may change in the future as we are currently discussing the possibility of a public release with TI.

6.3.2 Per-core DVFS

Very recently, work was published showing that on-chip voltage regulators are becoming feasible to implement, mitigating some of the problems arising from the use of off-chip voltage regulators to achieve per-core DVFS on multi-core processors, as discussed in Section 5.5. The main benefit of on-chip voltage regulation is the speed at which voltage transitions can be performed. Kim et al. show that with on-chip regulators, dynamic voltage scaling can be done at the nanosecond time scale, allowing fine-grained, per-core DVFS [Kim et al., 2011]. However, as Rotem et al. show, the benefits of per-core DVFS are not obvious and further analysis will be needed before processor manufacturers will implement per-core DVFS [Rotem et al., 2009]. Furthermore, Kim's work is still in very early stages, and full-blown implementations will take time to develop.

The Intel SCC also presents interesting power-management problems which require further research. However, the design of the SCC has not yet been proven as an effective way forward for high core-count CMPs.

6.3.3 Modelling with improved performance counters

Processor manufacturers are continually improving their performance monitoring units (PMUs) with more and more measurable events. The difficulty we had in modelling the effects of hardware prefetching, as described in Section 3.2, may improve in the future, allowing the modelling techniques described in Chapter 3 to be more successful.

6.3.4 OS scheduler optimisation for improved C state usage

As discussed in Section 4.4.4.1, for multi-threaded workloads such as Apache, the processor is unable to enter a *package* C state, due to at least one core being active or due to other activity in the system such as DMA or interrupt deliveries. The OS task scheduler could be optimised to allow all cores to enter the same idle state at once, potentially achieving greater energy efficiency. However, this will have an effect on response time and latency which would need to be addressed. This avenue of research is left to future work.

6.4 Conclusion

Managing energy consumption has become one of the most important factors influencing the design of modern computer processors. Manufacturers strive to provide the highest possible performance within strict power constraints driven by users asking for greater battery longevity for mobile devices and data-centre managers asking for lower power bills.

Many different power-management mechanisms are available on today's CPUs, but their effectiveness at improving energy efficiency is not immediately clear. Past studies have shown that DVFS can be effective in improving energy efficiency in single-core systems, however we show that this is not necessarily the case for more recent systems.

We analyse several recent systems and find trends that are influencing the effectiveness of DVFS including continuously shrinking feature sizes, improving memory performance, cheaper low-power C states and other architectural features. On this basis, we conclude that the effectiveness of DVFS in improving energy efficiency is likely to decrease in the future, creating a need for new power-management mechanisms to be developed.

Glossary

AMP asymmetric multi-processor. 22, 23, 76

CAS column access strobe. 40

ccNUMA cache-coherent non-uniform memory access. 18

CMOS complementary metal-oxide semiconductor. 6, 7

CMP chip multi-processor. 16, 21, 22, 32, 42–44, 70

CPU central processing-unit. 2, 3, 28

DCT DRAM controller. 38

DDR double-data-rate. 39, 45

DMA direct-memory-access. 59

DRAM dynamic random access memory. 28, 39

DVFS dynamic voltage and frequency scaling. i, xv, 2–5, 10, 14, 16, 21–24, 26, 27, 29–31, 39, 41–45, 57, 58, 62, 65, 66, 68–78

energy-delay product (EDP) a metric that allows for comparisons that account for both energy and execution time. 46, 47, 52

HPC high-performance computing. 42, 45

HT hyper-transport. 18

ISA instruction-set architecture. 22

ITRS international technology roadmap for semiconductors, a working group that publishes a set of documents describing the future for semiconductor manufacturing. 7, 13, 76

IVA-HD image and video accelerator - high definition. A hardware accelerator on the Texas Instruments OMAP 4430 SoC. 59, 76, 77

LLC last-level cache. 12, 30, 34

MOSFET metal-oxide semiconductor field-effect transistor. 6

OS Operating System. Software that controls the execution of computer programs, management of resources and may provide various services. i, 2, 6, 9, 11–14, 16, 22–25, 52, 54, 70, 76

PLL phase-locked loop. 10

PMU performance-monitoring unit. 14, 15, 18, 24, 32, 35, 53, 54

PSU power-supply unit. 17, 44

QoS quality-of-service. 10

SMP symmetric multi-processor. 22

SoC system-on-chip. 76

SuT system under test. 14, 60, 61

TDP thermal design power, the maximum expected power dissipation from a processor. Cooling solutions should be designed to dissipate this level of power.. 40, 43, 47

ULV ultra-low voltage. 69

voltage regulator a device designed to efficiently convert one voltage to another. 77

References

- Advanced Micro Devices. AMD Opteron processor reference. <http://products.amd.com/en-us/OpteronCPUResult.aspx>, 2010.
- ARK, Intel Corporation. Intel Core i7 840 processor specifications, retrieved December 2010. <http://ark.intel.com/Product.aspx?id=43125>, 2011.
- L. A. Barroso and U. Hözl. The case for energy-proportional computing. *IEEE Computer*, 40(12): 33–37, Dec. 2007.
- F. Bellosa. The benefits of event-driven energy accounting in power-sensitive systems. In *Proceedings of the 9th SIGOPS European Workshop*, Kolding, Denmark, Sept. 17–20 2000.
- W. L. Bircher and L. K. John. Complete system power estimation: A trickle-down approach based on performance events. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software*, San Jose, CA, USA, Apr. 25–27 2007.
- W. L. Bircher and L. K. John. Analysis of dynamic power management on multi-core processors. In *Proceedings of the 22nd International Conference on Supercomputing*, Island of Kos, Greece, June 2008.
- Brian Carlson and Bill Giolma. SmartReflex power and performance management technologies: reduced power consumption, optimized performance. http://focus.ti.com/pdfs/wtbu/smartreflex_whitepaper.pdf, 2008.
- D. Brooks, P. Bose, V. Srinivasan, M. Gschwind, P. Emma, and M. Rosenfield. New methodology for early-stage, microarchitecture-level power-performance analysis of microprocessors. *IBM Journal for Research and Development*, 47(5):653–670, 2003.
- T. Burd and R. Brodersen. Energy efficient cmos microprocessor design. volume 0, page 288, Los Alamitos, CA, USA, 1995. IEEE Computer Society. doi: <http://doi.ieeecomputersociety.org/10.1109/HICSS.1995.375385>.
- A. Carroll and G. Heiser. An analysis of power consumption in a smartphone. In *Proceedings of the 2010 USENIX Annual Technical Conference*, pages 1–12, Boston, MA, USA, June 2010.

- J. Charles, P. Jassi, N. Ananth, A. Sadat, and A. Fedorova. Evaluation of the Intel Core i7 Turbo Boost feature. In *IEEE International Symposium on Workload Characterization*, pages 188–197, Oct. 2009. doi: 10.1109/IISWC.2009.5306782.
- H. Esmaeilzadeh, T. Cao, X. Yang, S. M. Blackburn, and K. S. McKinley. Looking back on the language and hardware revolutions: Measured power; performance, and scaling. In *Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems*, Newport Beach, California, USA, Mar. 2011.
- Extech Instruments Corporation. Extech 380801 appliance tester / power analyzer product datasheet. <http://www.extech.com/instrument/products/310399/380801.html>, Aug. 2008.
- A. Fedorova, J. C. Saez, D. Shelepov, and M. Prieto. Maximizing power efficiency with asymmetric multicore systems. *Commun. ACM*, 52:48–57, December 2009. ISSN 0001-0782.
- J. 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.
- T. L. Floyd. *Electronic Devices*. Merrill, 1st edition, 1984.
- D. Grunwald, P. Levis, K. I. Farkas, C. B. Morrey III, and M. 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, Oct. 2000.
- J. Howard, S. Dighe, Y. Hoskote, S. Vangal, D. Finan, G. Ruhl, D. Jenkins, H. Wilson, N. Borkar, G. Schrom, F. Paillet, S. Jain, T. Jacob, S. Yada, S. Marella, P. Salihundam, V. Erraguntla, M. Konow, M. Riepen, G. Droege, J. Lindemann, M. Gries, T. Apel, K. Henriss, T. Lund-Larsen, S. Steibl, S. Borkar, V. De, R. Van Der Wijngaart, and T. Mattson. A 48-core ia-32 message-passing processor with dvfs in 45nm cmos. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International*, pages 108–109, 2010. doi: 10.1109/ISSCC.2010.5434077.
- Imgtec. VXD392 Video Decoder Factsheet. <http://www.imgtec.com/powervr/powervr-vxd.asp>, 2011.
- Intel Corporation. Intel XScale Microarchitecture for the PXA255 Processor. <http://int.xscale-freak.com/XSDoc/PXA255/27879601.pdf>, Mar. 2003.
- Intel Corporation. Intel Atom Z Series processor datashseet, retrieved March 2011. <http://www.intel.com/products/processor/atom/techdocs.htm>, 2011a.
- Intel Corporation. Intel Core i7 840 processor datashseet, retrieved March 2011. <http://www.intel.com/design/corei7/documentation.htm>, 2011b.
- C. Isci and M. Martonosi. Runtime power monitoring in high-end processors: methodology and empirical data. In *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture*, Dec. 2003.

- C. Isci, A. Buyuktosunoglu, C.-Y. Cher, P. Bose, and M. Martonosi. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In *Proceedings of the 39th ACM/IEEE International Symposium on Microarchitecture*, pages 347–358, Orlando, FL, USA, Dec. 2006.
- ITRS. System drivers and design tables. http://www.itrs.net/Links/2009ITRS/2009Chapters_2009Tables/2009Tables_FOCUS_F_ITRS.xls, 2009.
- R. Jotwani, S. Sundaram, S. Kosonocky, A. Schaefer, V. Andrade, G. Constant, A. Novak, and S. Naffziger. An x86-64 core implemented in 32nm SOI CMOS. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International*, pages 106–107, 2010. doi: 10.1109/ISSCC.2010.5434076.
- W. Kim, D. Brooks, and G.-Y. Wei. A fully-integrated 3-level DC/DC converter for nanosecond-scale DVS with fast shunt regulation. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2011 IEEE International*, Feb. 2011.
- R. Kumar, K. Farkas, N. Jouppi, P. Ranganathan, and D. Tullsen. Single-ISA heterogeneous multi-core architectures: the potential for processor power reduction. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture. MICRO-36.*, pages 81–92, Dec. 2003. doi: 10.1109/MICRO.2003.1253185.
- R. Kumar, D. M. Tullsen, P. Ranganathan, N. P. Jouppi, and K. I. Farkas. Single-ISA heterogeneous multi-core architectures for multithreaded workload performance. In *Proceedings of the 31st annual international symposium on Computer architecture, ISCA '04*, pages 64–, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2143-6.
- T. Li, D. Baumberger, D. A. Koufaty, and S. Hahn. Efficient operating system scheduling for performance-asymmetric multi-core architectures. In *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, SC '07, pages 53:1–53:11, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-764-3.
- J. R. Lorch and A. J. Smith. Improving dynamic voltage scaling algorithms with PACE. In *Proceedings of the 2001 ACM SIGMETRICS international conference on measurement and modeling of computer systems*, pages 50–61, New York, NY, USA, 2001. ACM. ISBN 1-58113-334-0.
- M. M. Mano and C. R. Kime. *Logic and Computer Design Fundamentals*. Prentice Hall, 1st edition, 1997.
- A. Merkel and F. Bellosa. Memory-aware scheduling for energy efficiency on multicore processors. In *Proceedings of the Workshop on Power Aware Computing and Systems (HotPower'08)*, San Diego, CA, USA, Dec. 2008.
- A. Miyoshi, C. Lefurgy, E. V. Hensbergen, R. Rajamony, and R. 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.

- OMAP 4430 Technical Reference Manual. Texas Instruments. <http://focus.ti.com/lit/ml/swpt034a/swpt034a.pdf>, 2011.
- K. K. Pusukuri, D. Vengerov, and A. Fedorova. A methodology for developing simple and robust power models using performance monitoring events. In *Workshop on the Interaction between Operating Systems and Computer Architecture*, Austin, Texas, June 2009.
- R. The R Project for Statistical Computing. <http://www.r-project.org/>, 2011.
- K. Rajamani, H. Hanson, J. C. Rubio, S. Ghiasi, and F. L. Rawson. Online power and performance estimation for dynamic power management. Technical report, IBM Research Division, Austin Research Laboratory, Austin, TX, USA, 2006.
- Y. Ren, K. Yao, M. Xu, and F. Lee. Analysis of the power delivery path from the 12-V VR to the microprocessor. In *IEEE Transactions on Power Electronics*, volume 19, pages 1507–1514, Nov. 2004. doi: 10.1109/TPEL.2004.836679.
- E. Rotem, A. Mendelson, R. Ginosar, and U. Weiser. Multiple clock and voltage domains for chip multi processors. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 42, pages 459–468, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-798-1.
- A. Roy, S. M. Rumble, R. Stutsman, P. Levis, D. Mazires, and N. Zeldovich. Energy management in mobile devices with the Cinder operating system. In *Proceedings of the 6th EuroSys Conference*, EuroSys '11, Salzburg, Austria, Apr. 2011.
- K. Roy, S. Mukhopadhyay, and H. Mahmoodi-Meimand. Leakage current mechanisms and leakage reduction techniques in deep-submicrometer CMOS circuits. *Proceedings of the IEEE*, 91(2): 305–327, Feb. 2003. ISSN 0018-9219. doi: 10.1109/JPROC.2002.808156.
- J. C. Saez, M. Prieto, A. Fedorova, and S. Blagodurov. A comprehensive scheduler for asymmetric multicore systems. In *Proceedings of the 5th EuroSys Conference*, EuroSys '10, pages 139–152, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-577-2.
- Siege. An HTTP load testing and benchmarking utility, version 2.70. <http://www.joedog.org/index/siege-home>, 2010.
- D. Snowdon and D. Johnson. PLEB2: The portable linux embedded box version 2. <http://www.ertos.nicta.com.au/hardware/pleb/projectdir/pleb2.pml>, Jan. 2003.
- D. C. Snowdon. *OS-Level Power Management*. PhD thesis, School of Computer Science and Engineering, University of NSW, Sydney 2052, Australia, Mar. 2010. Available from publications page at <http://www.ertos.nicta.com.au.au/>.
- D. C. Snowdon, S. M. Petters, and G. 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, Oct. 2007.

- D. C. Snowdon, E. Le Sueur, S. M. Petters, and G. Heiser. Koala: A platform for OS-level power management. In *Proceedings of the 4th EuroSys Conference*, Nuremberg, Germany, Apr. 2009.
- Standard Performance Evaluation Corporation. CPU2006. <http://www.spec.org/cpu2006/>, 2006.
- Texas Instruments. OMAP44xx Technical Reference Manual. http://focus.ti.com/pdfs/wtbu/OMAP4430_ES2.0_Public_TRM_vJ.pdf, 2011.
- W. Tschudi, T. Xu, D. Sartor, B. Nordman, J. Koomey, and O. Sezgen. Energy efficient data centers. Technical report, California Energy Commission, Mar. 2004. URL <http://www.osti.gov/bridge/servlets/purl/841561-a07Lg9/native/841561.pdf>.
- L. Wanner, C. Apte, R. Balani, PuneetGupta, and M. Srivastava. A case for opportunistic embedded sensing in presence of hardware variability. In *Proceedings of the 2010 Workshop on Power Aware Computing and Systems (HotPower'10)*, Vancouver, Canada, Oct. 2010.
- M. Weiser, B. Welch, A. J. Demers, and S. Shenker. Scheduling for reduced CPU energy. In *Proceedings of the 1st USENIX Symposium on Operating Systems Design and Implementation*, pages 13–23, Monterey, CA, USA, Nov. 1994.
- A. Weissel and F. 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, Oct. 8–11 2002.
- H. Zeng and C. S. E. A. R. Lebeck. Experiences in Managing Energy with ECOSystem. *IEEE Pervasive Computing*, 4(1):62–68, 2005.
- H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat. Ecosystem: managing energy as a first class operating system resource. *SIGPLAN Not.*, 37(10):123–132, 2002. ISSN 0362-1340. doi: <http://doi.acm.org/10.1145/605432.605411>.
- H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat. Currentcy: Unifying policies for resource management. In *Proceedings of the 2003 USENIX Annual Technical Conference*, San Antonio, Texas, June 2003.