Technical Report

UCAM-CL-TR-821
ISSN 1476-2986

Number 821

UNIVERSITY OF
CAMBRIDGE
Computer Laboratory

Modelling energy efficiency for computation

Charles Reams

October 2012

15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500

http://www.cl.cam.ac.uk/

© 2012 Charles Reams

This technical report is based on a dissertation submitted October 2012 by the author for the degree of Doctor of Philosophy to the University of Cambridge, Clare College.

Technical reports published by the University of Cambridge Computer Laboratory are freely available via the Internet:

http://www.cl.cam.ac.uk/techreports/

ISSN 1476-2986

Summary

In the last decade, efficient use of energy has become a topic of global significance, touching almost every area of modern life, including computing. From mobile to desktop to server, energy efficiency concerns are now ubiquitous. However, approaches to the energy problem are often piecemeal and focus on only one area for improvement.

I argue that the strands of the energy problem are inextricably entangled and cannot be solved in isolation. I offer a high-level view of the problem and, building from it, explore a selection of subproblems within the field. I approach these with various levels of formality, and demonstrate techniques to make improvements on all levels. The original contributions are as follows.

Chapter 3 frames the energy problem as one of *optimisation with constraints*, and explores the impact of this perspective for current commodity products. This includes considerations of the hardware, software and operating system. I summarise the current situation in these respects and propose directions in which they could be improved to better support energy management.

Chapter 4 presents mathematical techniques to compute energy-optimal schedules for long-running computations. This work reflects the server-domain concern with *energy cost*, producing schedules that exploit fluctuations in power cost over time to minimise expenditure rather than raw energy. This assumes certain idealised models of power, performance, cost, and workload, and draws precise formal conclusions from them.

Chapter 5 considers techniques to implement energy-efficient real-time streaming. Two classes of problem are considered: first, hard real-time streaming with fixed, predictable frame characteristics; second, soft real-time streaming with a quality-of-service guarantee and probabilistic descriptions of per-frame workload. Efficient algorithms are developed for scheduling frame execution in an energy-efficient way while still guaranteeing hard real-time deadlines. These schedules determine appropriate values for power-relevant parameters, such as dynamic voltage–frequency scaling.

A key challenge for future work will be unifying these diverse approaches into one “Theory of Energy” for computing. The progress towards this is summarised in Chapter 6. The thesis concludes by sketching future work towards this Theory of Energy.

Acknowledgements

First I would like to thank my supervisor Prof. Alan Mycroft, whose faith in my ability to complete this work always exceeded my own. I hope I have repaid the trust and freedom he has granted me over the last three and a half years.

I must also thank my family, who unknowingly put me on the road to this PhD twenty years ago when they would allow me to play on my Game Boy until (the agony!) the batteries ran out. I would also like to thank Emma Seaber, who tolerated the long and difficult labour that birthed this thesis with exceptional grace, and feigned interest in Pareto frontiers far beyond the call of duty. There are not enough Nutrageouses to repay you.

Many more people deserve a mention than this space will allow, but to select a few: Michael Wallace, Miloš Puzović, and Greg Chadwick for making it so much more than the work, and David Brown, who on many occasions showed me the light at the end of the tunnel. Thank you all.

Additional thanks belong to the examiners of the thesis on which this technical report is based, John Daugman and Simon McIntosh-Smith. Their comments have improved this final edition in every respect, from high concept to lowly punctuation. All remaining errors are, of course, mine.

My final thanks belong to Clare College, whose Domestic Research Scholarship supported me throughout my studies.

This thesis is dedicated to the memory of my late grandfather, Jim Burnett, who lived to see the first words of this thesis but not the last. As the more mathematical chapters demonstrate, I have inherited his enthusiasm for brackets.

Contents

1 Introduction 15

1.1 Central thesis . . . 16

1.2 Contributions . . . 16

1.3 Chapter plan . . . 16

2 Survey of previous work 19

2.1 The meaning of energy awareness . . . 19

2.2 Energy-aware hardware . . . 20

2.2.1 Historical trends in hardware energy efficiency . . . 21

2.2.2 Future tends in hardware energy efficiency . . . 23

2.2.3 Energy characteristics at sub-peak performance . . . 26

2.3 Energy-aware operating systems . . . 28

2.4 Energy-aware protocols . . . 28

2.4.1 Energy-aware data protocols . . . 29

2.4.2 Energy management protocols . . . 30

2.5 Energy-aware software . . . 31

2.6 Summary of related work . . . 32

3 Towards energy-efficient computing 35

3.1 Energy in the computing space . . . 36

3.2 Modern power management . . . 38

3.2.1 ACPI . . . 39

3.3 Energy efficiency in computing . . . 39

3.3.1 Maintenance of required performance . . . 40

3.3.2 Response to changes in demand . . . 41

3.3.3 Power capping . . . 42

3.3.4 Summary of constraints . . . 42

3.4 Directions toward a solution . . . 42

3.4.1 Power model . . . 42
3.4.2 Workload constraints and performance assessment . . . 43
3.4.3 Energy optimisation by the system . . . 44
3.5 Routes towards energy efficiency . . . 45
3.5.1 Considerations for the operating system . . . 45
3.5.2 The evolution of application software . . . 48
3.6 Conclusion . . . 49

4 Cost optimisation for power-aware computing 51
4.1 Motivation . . . 51
4.2 Related work . . . 52
4.2.1 Dynamic voltage–frequency scaling . . . 53
4.2.2 Other formal methods . . . 55
4.3 Definition of terms . . . 55
4.4 Problem statement . . . 55
4.5 Formal model . . . 56
4.5.1 Assumptions and justification . . . 57
4.6 Solutions for specific cost models . . . 59
4.6.1 Constant cost . . . 59
4.6.2 General discrete variable cost . . . 63
4.6.3 Real-world example . . . 65
4.6.4 Exponential cost . . . 67
4.6.5 Generalised time intervals . . . 67
4.6.6 Summary of cost model-specific solutions . . . 70
4.7 General methods . . . 70
4.7.1 Monotonic cost functions . . . 71
4.7.2 Constant spending . . . 72
4.8 Conclusion . . . 74

5 Energy-efficient real-time streaming 77
5.1 Introduction . . . 77
5.1.1 Definition of an operating point . . . 79
5.1.2 Assumptions regarding operating points . . . 80
5.2 Problem statement . . . 81
5.3 Related work . . . 83
5.4 Algorithm description . . . 85

5.4.1 Dominated points 87
5.4.2 Downscaling 89
5.5 Evaluation 90
5.5.1 Diminishing returns 90
5.5.2 Simulation 92
5.5.3 Heuristic improvements 94
5.5.4 Evaluation of domination 95
5.6 Probabilistic generalisation 95
5.6.1 Probabilistic model 95
5.6.2 Suitability for practical soft real-time problems 96
5.6.3 Assumption of normality 97
5.6.4 Probabilistic manifestation 98
5.6.5 Probabilistic algorithm 98
5.6.6 Generalised domination 100
5.6.7 Probabilistic evaluation 101
5.6.8 Probabilistic summary 104
5.7 Conclusion 105

6 Conclusions **107**
6.1 Concluding argument 107
6.2 Future directions 109
6.2.1 Future work on the theory of energy-efficient computing 109
6.2.2 Future work on cost-efficient computing 110
6.2.3 Future work on energy-efficient streaming 110

Appendices **113**

A Future work: online learning in hard real-time **113**
A.1 Introduction 113
A.2 Motivation 113
A.3 Problem outline 114
A.4 Proposed methodology 115
A.5 Challenges 115
A.6 Related work 116

Bibliography **119**

Glossary **133**

List of Figures

2.1 Graph of computations per unit energy through history, in support of Koomey's Law. Reproduced from [Koomey et al., 2011]. 20

2.2 Size comparison of a set of DEC Alpha cores from a heterogeneous multi-core chip; each core is labelled with the single-core chip in which it originally appeared. Reproduced from [Kumar et al., 2003]. 24

2.3 Plot of power consumption against nominal parallel efficiency for various numbers of cores. Reproduced from [Li and Martínez, 2005]. 25

2.4 Transistor behaviour in sub-, near-, and super-threshold operating voltage regions. Reproduced from [Dreslinski et al., 2010]. 27

3.1 Energy usage breakdown by equipment type in United States, from 2000 to 2006. 37

3.2 Time-throughput diagram showing latency to reach maximum performance. 41

4.1 Potential cost saving λ, as a percentage, against static fraction μ, for some realistic α values. 62

4.2 Cost incurred to perform a long-running computation, for various ratios of static to dynamic power. 65

4.3 Energy required to perform a long-running computation, for various ratios of static to dynamic power. 66

4.4 A continuous cost function and two different methods of discretising it into ten steps. 69

4.5 Schedules for two different quantised approximations to the same continuous cost function. Arbitrary units of rate. 70

4.6 Comparison of cost overheads incurred by regular and fitted quantisations for several values of α. Horizontal axis is on a logarithmic scale. 71

5.1 Comparison of the traditional Turing model and streaming computation. . 78

5.2 Example set of operating points. 79

5.3 An example set of operating points and a per-frame deadline shown in energy–time space. 81

5.4 Example set of operating points, showing dominated and undominated points, arbitrary units. 88

5.5 Probability that integer time values generated uniformly at random have a common factor, as a function of m, the number of values. 90

5.6 q_ν for increasing values of ν, averaged over one hundred random examples. Note that the vertical axis starts at 0.4. 91

5.7 Energy–time diagram for real BSOM operating points. 92

5.8 Comparison of the per-frame energy reduction achieved by my static algorithm and a dynamic greedy algorithm. 93

5.9 Comparison of the per-frame energy reduction achieved, as a fraction of the available maximum. 94

5.10 Diagram showing the variable deadline encountered by the dynamic frame scheduling algorithm. 102

5.11 Comparison of the energy reduction achieved by my static algorithm and a dynamic greedy algorithm for normally distributed execution times. . . . 103

5.12 Two different forms of operating point energy–time description, represented graphically. 105

List of Tables

3.1 Energy usage breakdown for computing equipment in the United States. Energy figures are in billions of kWh. 36

5.1 Average solution time. 94

5.2 Point counts for BSOM data, and comparison to heuristic estimates. 95

5.3 Power reductions measured for the BSOM benchmark. For meaning of column headings, see text. 104

List of Algorithms

1 Dynamic programming algorithm to find minimal energy which meets the throughput requirement. 86

2 Algorithm to find undominated operating points. 88

3 Brute-force algorithm to find frequencies which minimise expected energy and meet the probabilistic timing constraint. 99

Chapter 1

Introduction

Many fields of computing are currently experiencing a new or renewed interest in energy efficiency. In desktop computing, ecological concerns are forcing hardware designers to refine their traditional assumption that mains electricity is an unlimited source of power. In the server space, energy demands over the lifetime of the hardware are rapidly increasing and may come to dominate the cost of the hardware itself [Barroso and Hölzle, 2007]. In both contexts, reducing power demand from electronic components also reduces the power demand of the cooling subsystem, so savings are multiplied. And of course any form of mobile computing, a huge growth area in the last decade, must consider power efficiency; most obviously from the perspective of maximising battery life, and additionally from the complication of cooling in a mobile context.

The energy problem has diversified considerably in recent years. The range of use cases and attendant hardware and software, from servers to desktops to laptops to tablets to mobile telephones to embedded devices and so on, is now so vast that it is infeasible to attempt to approach all of them simultaneously. This is particularly true if one seeks analytic rather than heuristic solutions, as will generally be the case in this thesis. One might compare the situation to that of physics in the 19th Century; the great breakthrough of James Clerk Maxwell's unification of electricity and magnetism was not possible until the two phenomena were individually well-understood.

In recognition of these difficulties, I make a two-part argument in this thesis: on the one hand, we should not be too hurried to solve all the problems of energy-efficient computing, since such a solution is clearly intractable with our current understanding; on the other hand, we should remain aware of the larger picture as we advance the subproblems, so that opportunities to unify and generalise can be uncovered as they come within reach.

Furthermore, with the central argument in mind, I make headway on some of these particular problems: power cost management and real-time streaming. My intention is that these pieces will ultimately be fitted into a "grand unified theory" of energy-efficient computing.

In power cost management, I generalise traditional notions of energy efficiency to include energy cost efficiency, and argue that these concerns are connected but not identical and that cost optimisation presents an interface to economic reality that is not available if one simply "counts joules". I present formal models connecting power, performance, and cost, and show formal optimisation methods for them. I then investigate whether these optimisations present adequate efficiency savings to justify their deployment in practice.

In real-time streaming, I develop a framework in which tasks in a (potentially infinite) stream can be processed energy-efficiently while still meeting hard real-time deadlines. This framework includes an abstraction from the particular characteristics of the workload—hardware or software—and, on this platform, presents a generalisation of traditional energy-efficient scheduling techniques such as voltage dithering. I also present algorithms to find these schedules, and explore whether it is possible to compute them in a time- and space-efficient manner. Again, the results of the schedules are tested by simulation against existing approaches, and the energy differential measured.

I conclude with a discussion of what is still to be done towards a unified theory of energy-efficient computing.

1.1 Central thesis

My central thesis is that *the subproblems of the energy problem as it pertains to computing are deeply interrelated, and it is necessary to consider the whole before addressing the parts*. To support this argument, I survey and discuss the state of the field, and in later chapters, make headway on certain parts—in particular, power cost management and real-time streaming—and relate these back to the problem as a whole.

1.2 Contributions

My contributions to the field are as follows:

- An extensive description of the current situation "on the ground" with regards to energy efficiency, across the server, desktop and mobile sectors.
- The proposal of energy cost as the true metric of optimisation in energy-aware computing.
- Formal methods for solving energy cost problems under realistic cost and performance models.
- The proposal of the *operating point* as a clean abstraction of execution details (in both hardware and software), suitable for general-purpose energy-aware modelling.
- The description of precise algorithms for solving operating point problems in energy-efficient streaming, and their time and space characteristics.

1.3 Chapter plan

Following this chapter, the chapter structure of this thesis is as follows:

- Chapter 2 surveys various strands in the field of energy-aware computing.
- Chapter 3 presents a large-scale view of several pragmatic levels on which the energy problem might be approached, with a particular emphasis on the view from industry.

- Chapter 4 introduces the concept of energy cost minimisation, describes mathematical techniques by which such minimisation may be achieved, and measures the utility of these techniques.
- Chapter 5 defines the “operating point” and develops a framework for energy-efficient hard real-time streaming, based on the concept of operating point dithering.
- Chapter 6 synthesises the results of previous chapters into a single argument, making the final demonstration of the central thesis.

Chapter 2

Survey of previous work

Given the diversity of the topics considered in this document, this survey of prior work does not attempt to introduce all of the detailed context immediately; instead, much of the relevant material will be introduced directly in the appropriate chapter. Rather, this chapter serves as a general summary of the state of the art in energy-aware computing, and thereby puts the specific work of later chapters in broader context. Of course, it makes no pretension to cover every area, and is necessarily highly selective in those that are covered, but nevertheless its purpose is the provision of an adequate foundation for the details of what follows.

2.1 The meaning of energy awareness

The phrase "energy-aware computing" is carefully chosen, because there are many possible interactions between computation and the energy it requires. For example, in a battery-operated device, the usual objective is to minimise the total energy of a computation; that is, the integral of power with respect to time over the whole computation. In other devices, peak instantaneous power is the more pressing constraint. Some machines, such as spacecraft, may combine solar power with battery back-up, and so are subject to both constraints. Other constraints might include a thermal dissipation limit imposing a maximum average power over a given period of time. Additionally, there are various classes of problem depending on which performance, energy, power and so forth are to be optimised and which are constrained. To further complicate matters, batteries require quite specific usage profiles to give optimal performance, as described below. Moreover, the machine may not be intended for energy efficiency at all, but pure energy awareness, simply measuring its own energy usage and displaying this to the user or relaying it to an external reporting service. In practical systems, resolving these requires a different approach in each case, so this survey presents as broad coverage of these areas as possible.

To recapitulate, my central thesis is that the different parts of the energy problem are closely related and must be considered as one. Consequently, while this chapter catalogues the related work according to the particular area that it targets–hardware, operating system, protocols, and software–this taxonomy does not quite carve nature at the joints, and so the chapter concludes with a discussion of some interdisciplinary work.

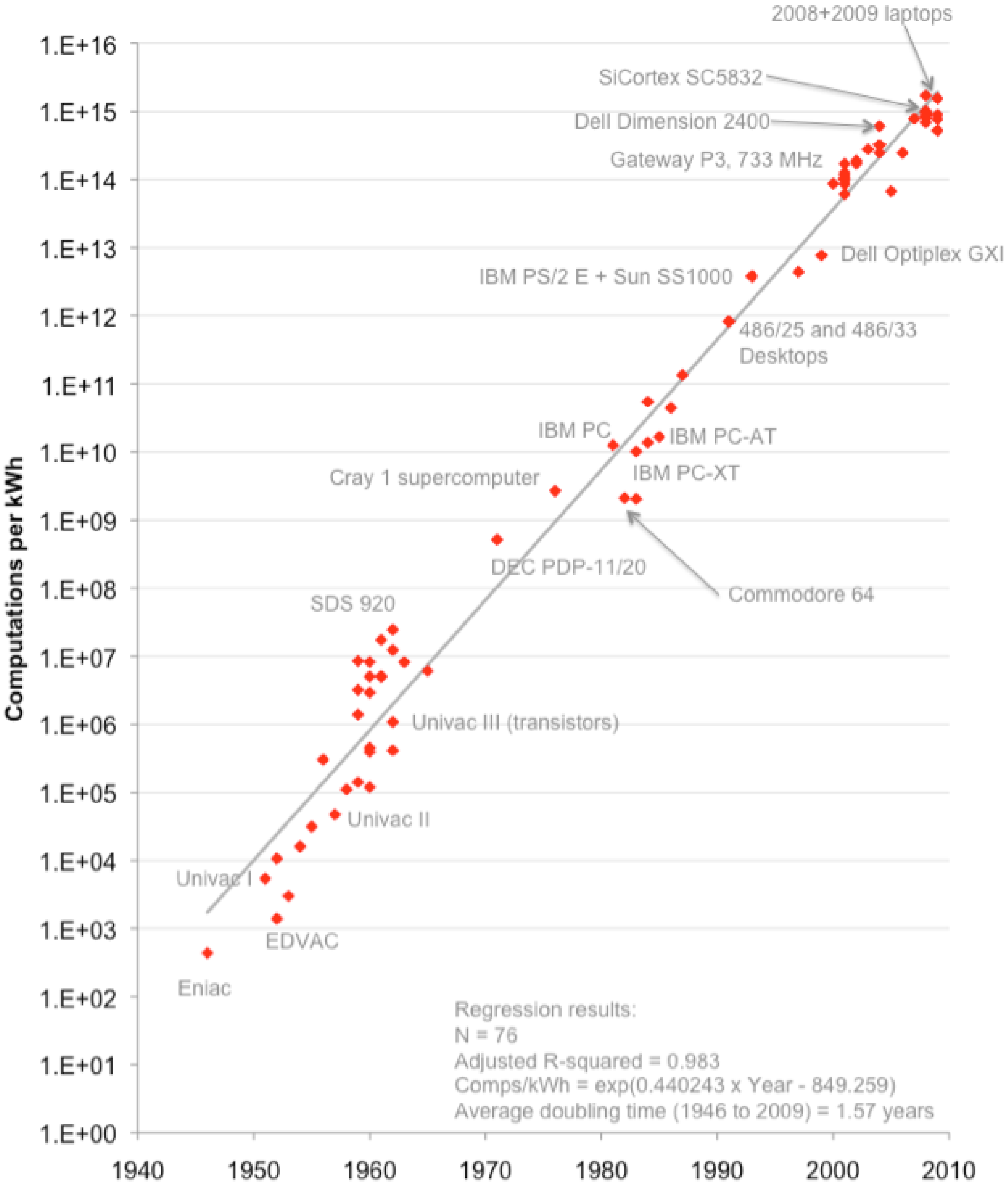

Figure 2.1: Graph of computations per unit energy through history, in support of Koomey's Law. Reproduced from [Koomey et al., 2011].

2.2 Energy-aware hardware

To call hardware energy-aware seems almost tautological, since hardware is definitionally composed of physical objects subject to the usual laws of thermodynamics. However there is still a distinction to be drawn between three areas of work: techniques by which the hardware itself is made more energy-efficient; mechanisms for monitoring and communicating power and energy values to higher levels; and hardware features that these higher levels may use to further improve the energy profile, closing the loop.

In fact, the historical trend of increasing performance closely tracks increases in energy efficiency, and this is encapsulated in an analogue of Moore's Law known as Koomey's Law [Koomey et al., 2011]. Moore stated that the number of components (latterly transistors) per chip doubled every two years; Koomey observed that computation per kilowatt-hour has followed a similarly exponential trend, doubling every 1.57 years. This trend is seen to hold throughout the history of computing, from the 1946 ENIAC to the most recent laptop and desktop machines, as shown in Figure 2.1. It has been said that "in 1978, a commercial flight between New York and Paris cost \$900 and took seven hours. If the principles of Moore's Law were applied to the airline industry, that flight would now cost about a penny and take less than one second" [Semiconductor Industry Association, 2005]. If Koomey's Law were applied analogously, the aircraft would now require less than 27 milligrams of fuel[1].

Koomey's Law is an empirical observation about machines operating at peak performance and consequently maximum power consumption. This suggests a neat partition of the existing work on hardware power-awareness into three parts: how such reductions have actually been achieved historically; to what extent this trend can be expected to continue into the future; and, how machines behave when operating at less-than-peak performance.

2.2.1 Historical trends in hardware energy efficiency

Koomey's central collection of evidence is presented in Figure 2.1. The trend is clear to the naked eye, and is borne out by the line of best fit, with an $\overline{R^2}$ value of 0.983.[2] Furthermore, since Koomey's observation is more recent, it is less vulnerable than Moore's Law to the criticism that it is a self-fulfilling prophecy, with the semiconductor industry adjusting its goals to meet the predictions. Therefore one can be reasonably confident of its robustness.

First, I address the historical explanation of Koomey's observation. There are several salient features of Figure 2.1. The leap seen in the early 1960s is explained by the transition from vacuum tube computing to the transistor, and progress through the 1970s can be attributed to the adoption and eventual dominance of CMOS, but since then progress has been steadier. The continued reductions can be understood by consideration of the three main ways in which energy is expended in a CMOS circuit: leakage, direct-path short-circuiting, and switching [Chandrakasan et al., 1995].

Leakage power is given by $I_{leakage}V_{dd}$, where $I_{leakage}$ is the leakage current and V_{dd} the supply voltage. Traditionally neglected, leakage current is determined primarily by the CMOS fabrication technology, and has increasingly become a limitation in modern chip design since its reduction has been rather slower than that of other factors [Roy et al., 2003]. Only in the last decade has it been a significant explanatory factor in Koomey's Law, and its future is discussed below.

---

[1] This assumes a Boeing 777 at its maximum 150 MW power output and BP Avgas 80 fuel producing 44.5 MJ/kg, considered independently from the reduction in flight time. Values taken from the BP Handbook of Products, 2000. `http://www.bp.com/liveassets/bp_internet/aviation/air_bp/STAGING/local_assets/downloads_pdfs/a/air_bp_products_handbook_04004_1.pdf`

[2] The traditional coefficient of determination R^2 can only increase as more variables are added, encouraging over-fitting. $\overline{R^2}$, or *adjusted* R^2, is a modification that penalises models for each additional variable they introduce. Koomey's model is bivariate so the adjustment is very slight.

Direct-path short circuiting occurs when the N- and P-transistors of a CMOS unit are both activated at once, briefly connecting the supply to ground, and is given by $P_{sc} = I_{sc}V_{dd}$; again I_{sc} is another hardware-determined constant, typically small in well-designed circuits.

Finally, the switching power is given by perhaps the most important equation in the field:

$$P_{switch} = \alpha C_L V_{dd}^2 f \tag{2.1}$$

where P_{switch} is the switching power expended, α the switching activity, C_L the load capacitance, and f the clock frequency. Historically, switching power has been the dominant contributor to total power in CMOS, and prior work has targeted all four of these factors.

The switching activity $0 \leq \alpha \leq 1$ is a constant denoting the fraction of clock cycles on which a transition occurs; CMOS consumes little power except during such a transition. Some techniques addressing α occur at the compiler or ISA design level, addressed below, but other work targets hardware components directly. For example, the *Bus-Invert* method reduces the switching factor on an I/O bus by sending data inverted if this would produce fewer state switches; this inversion must itself be signalled, which may require an additional switching event. This trade-off is in fact a profitable one, as empirical results show that Bus-Invert produces a mean reduction of 25% in average power and 50% in peak power [Stan and Burleson, 1995]. Address buses are another attractive target, since they follow naturally long paths within and across the chip boundary, and carry fairly predictable data; for example, adjacent instructions are usually accessed sequentially, and therefore the instruction addresses tend to be consecutive or nearby integers. This suggests that use of a Gray code for instruction addresses might substantially reduce the expected number of transitions on the bus; indeed, in simulations a 58% reduction in α was observed by use of this technique [Su et al., 1994b].

Load capacitance is a physical characteristic determined by the chip's wiring. Modern place-and-route tools seek (among many competing objectives) to minimise the total routed wire-length (RWL), and in particular to shorten long wires, which are the main culprits for high C_L values [Alpert et al., 2010]. One simple optimisation, typically performed as one of the last phases in place-and-route, is to insert additional repeaters into long wires off the critical path, transforming the path into multiple shorter wires [Alpert et al., 2010, p. 9]. Capacitance also scales up with increasing wire diameter, and of course wider wires are more difficult to route and (like repeaters) increase delay [Li et al., 2008]. More recently, the advent of three-dimensional integrated circuits has allowed substantial reductions in RWL. The extra dimension afforded by stacking multiple planes of transistors on top of one another, while presenting many new challenges in design and manufacturing, is known to afford remarkable improvements in overall routing quality; in a typical set-up, it was shown to reduce the RWL by 28–51% and the length of the longest wire by 31–56% [Das et al., 2003].

The supply voltage is the most obvious target in Equation 2.1 due to its quadratic proportion to the switching power. Miniaturisation of MOSFETs, and the consequent shrinkage of CMOS, naturally allows lower supply voltages. Modern chip designs often permit the supply voltage to be varied dynamically, at the cost of some performance, but this discussion is left for Section 2.2.3.

Clock frequency is primarily determined by the delay of the critical path, although clock distribution imposes additional limitations. Frequency has become something of a controversial issue in the last twenty years, as frequency became a quotable marketing point for competing processor manufacturers. Some retailers promoted the notion that a higher number simply denoted better performance: the so-called "megahertz myth".[3] While this is typically true for a given architecture, the comparison between architectures is a good deal more complex. While Intel's design focused on maximising clock speed, rival manufacturer AMD explored some of the techniques by which comparable performance can be achieved with a lower clock speed; in particular, superior superscalability and reduced memory latency [Matsui, 2006]. As discussed below, varying the clock frequency dynamically is also a well-explored field.

It is worth noting, finally, that the rate of improvement in battery technology has been very modest in comparison with the explosive trend of Koomey's Law, and that we therefore cannot rely on increasing the total energy available in order to lengthen time between charges. Also, modern batteries do not simply provide a fixed amount of energy; rather, their efficiency depends on the current drawn, which should ideally be low [Pedram and Wu, 1999]. In other words, doubling the current drawn by a device would reduce the battery life by substantially more than half. The variability of the current over time also has an impact, with more stable currents correlating with higher efficiency. This complication is rarely considered in the literature to date.

2.2.2 Future tends in hardware energy efficiency

Early transistor improvements were driven by so-called "Dennard scaling": ever-improving miniaturisation, with attendant improvements to supply voltage and clock speed [Dennard et al., 1974]. Pure Dennard scaling ended around the year 2000 with the 130 nm CMOS process; since then, other techniques have been required to keep pace with Moore's Law [Kuhn, 2009]. The International Technology Roadmap for Semiconductors[4] estimates that the current 22 nm CMOS production process will be reduced to 16 nm by 2013 and 11 nm by 2015. However, the majority of the future improvements will have to lie elsewhere.

On the general rate of energy reduction, Koomey notes cheerfully that "we fully expect those improvements to continue in coming years" [Koomey et al., 2009]. Indeed, work is underway in many areas that are expected to yield further increases in computations per unit energy irrespective of miniaturisation. It is now widely agreed that the future of processing lies in more cores per chip—the so-called "chip multi-processor" (CMP) era—for the following reasons. As mentioned above, leakage current is now a significant factor of total power, and this can be controlled effectively by disabling individual cores. Another motivation is Pollack's Rule, which states that [Borkar, 2007]

$$\text{performance} \propto \sqrt{\text{complexity}} \tag{2.2}$$

and this mitigates against very large-area cores; therefore, it is better to invest the die area in a larger number of small-area cores. Options more flexible than completely disabling cores are available. One approach is use of heterogeneous multi-core architectures, with

---

[3] The origin of the term "megahertz myth" is unknown, but it had been adopted by the mainstream press by the early 2000s, for example `http://www.guardian.co.uk/technology/2002/feb/28/onlinesupplement3`.

[4] International Technology Roadmap for Semiconductors, 2011 Edition. `http://www.itrs.net/Links/2011ITRS/Home2011.htm`

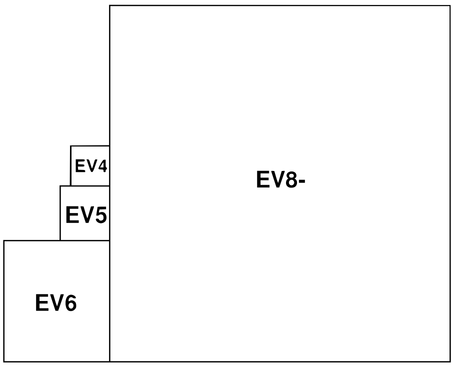

Figure 2.2: Size comparison of a set of DEC Alpha cores from a heterogeneous multicore chip; each core is labelled with the single-core chip in which it originally appeared. Reproduced from [Kumar et al., 2003].

each core (or set of cores) placed at a different point in the power-performance trade-off; workloads are then assigned to an appropriate core dynamically. Tests show that, for an example set of DEC Alpha cores running the SPEC benchmarks, one can reduce total energy by an average of 39% while increasing execution time by just 3% [Kumar et al., 2003]. This particular configuration, in which various generations of the same processor family are loaded onto a single die, has multiple benefits: first, the die size is kept manageable, since the most recent generation is typically at least as large as the previous generations combined, as shown in Figure 2.2; second, the design and verification is more straightforward since the components already exist and are known to work. So we can expect to see more such designs in the future.

For applications that exhibit a high degree of internal parallelism (as opposed to the external parallelism of multiple applications running independently), multicore offers another opportunity for energy saving. Define the *nominal parallel efficiency* $\epsilon_n(N)$ of such an application when run on N processors as

$$\epsilon_n(N) = \frac{C_1}{NC_N} \tag{2.3}$$

where C_i is the number of cycles the program will consume when executed on i cores. (Note that in general, $C_i > C_1$ for $i > 1$ because the program must perform extra work to communicate data and synchronise control flow between the threads.) Existing work shows that, for certain values of $\epsilon_n(N)$, a higher value of N may produce the same performance for less energy [Li and Martínez, 2005]. Tests were conducted on an example program running on a typical 65 nm processor; Figure 2.3 plots its power consumption against $\epsilon_n(N)$ for a given performance requirement; the plot shows the power for $N = 2, 4, 8, 16$ and 32, normalised by the power required for the same program on a single core. Evidently, for $\epsilon_n(N) \gtrsim 0.25$, energy can be saved by deploying more cores, although the optimal number of cores bears a complex relationship with the value of $\epsilon_n(N)$. Even for values close to one (denoting perfect linear speedup), it is not necessarily optimal to enable as many cores as possible. This demonstrates that savings can also be made on homogeneous systems.

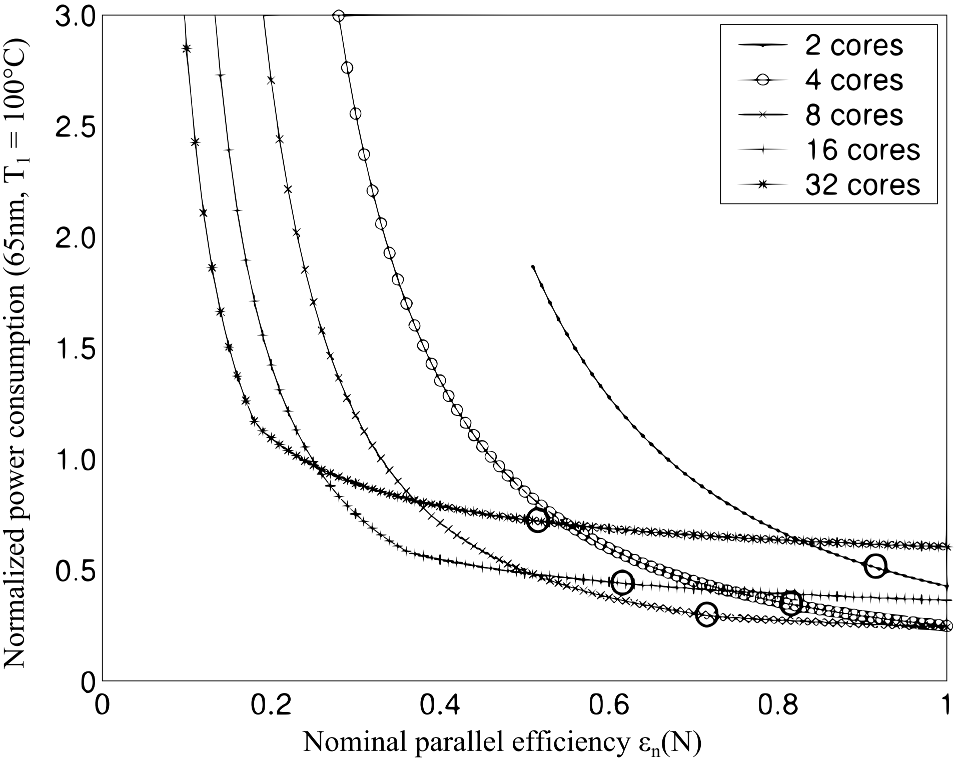

Figure 2.3: Plot of power consumption against nominal parallel efficiency for various numbers of cores. Reproduced from [Li and Martínez, 2005].

Theoretical limits to computational energy

One might reasonably ask how long Koomey's Law can continue to hold before some theoretical bound is reached. In fact, the question of computational energy bounds has been explored since well before the formulation of Koomey's Law, and wide-ranging arguments have been made on the subject for several decades [Bennett and Landauer, 1985]. In 1985, Richard Feynman estimated that the amount of electricity per unit computation might be reduced by a factor of 10^{11} [Koomey et al., 2011]. Improvements since then amount to around 10^5, and if Koomey's Law continues, such a limit would be reached sometime in the year 2041.

More fundamentally still, Landauer's Principle gives a thermodynamic lower bound on the amount of energy expended in changing a single bit of information in any physical representation [Landauer, 1961]. This Landauer limit is $kT \ln 2$, where k is the Boltzmann constant and T is the absolute temperature of the relevant physical object. Let us assume, very optimistically, that a computation required only a single bit change and occurred at the background temperature of the universe, the lowest temperature one can reach without expending further energy on cooling. In this case, $T = 2.725$ K and a computation could be performed in 7.2×10^{-33} kilowatt-hours. This limiting value is the efficiency predicted by Koomey's Law for the year 2098.

In fact, one can go further still, by application of *reversible computing*, a mode of computing in which information is not destroyed and therefore Landauer's entropic argument can be sidestepped [Bennett, 1973]. Although it is not possible to build a computer that preserves information perfectly in its state transitions, there is, in our current understanding of physics, no limit to how closely one might approach this limit. So, at least in theory, we may see Koomey's Law upheld for some time yet. Neatly enough, the promising field of quantum computing, a form of reversible computing, also originates with Feynman [Feynman, 1982].

2.2.3 Energy characteristics at sub-peak performance

Koomey's Law addresses the efficiency of a computer running at peak performance. However, machines often allow a much wider range of dynamic performance-power trade-offs. This is discussed in greater detail in Chapter 3, but some points are of general relevance. Many devices now provide low-power idle states, in which they are not usable but must instead be transitioned back to the active state. (Of course the transition itself also requires some time and energy.) Modern CPUs are among the most dynamic of these. For much of their history, CPUs simply executed no-ops when they had no useful work to perform. This was the basis for many of the early "crowdsourcing" computational projects, such as the Great Internet Mersenne Prime Search[5], SETI@home[6], and distributed.net[7], since these cycles would otherwise have been wasted. More recent designs allow CPUs to transition between various idle states, among other power-saving measures, such as dynamic voltage–frequency switching.

Dynamic voltage–frequency scaling

Equation 2.1 demonstrates the relationship between supply voltage and switching power, the latter of which has until recently accounted for the majority of the processor's power consumption. Many modern designs allow *dynamic voltage scaling* (DVS), in which the operating voltage of a processor can be adjusted on-the-fly so as to trade performance for power. However, the circuit delay also varies with the operational voltage, according to the equation

$$t \propto \frac{V_{DD}}{(V_{DD} - V_T)^{\alpha}} \tag{2.4}$$

where V_{DD} is the operational voltage, V_T the threshold voltage, and α the so-called velocity saturation index [Taur and Ning, 1998, pp. 269–271]. (The latter two are parameters of the CMOS technology.) Since the maximum clock frequency is in turn limited by the circuit delay, one typically scales frequency and voltage in concert: *dynamic voltage–frequency scaling* (DVFS). Since the power is reduced quadratically while the performance degrades only linearly, there is a net linear saving in total energy. Therefore, at least in the region for which the switching power is the dominant contributor to total power, we can expect approximately linear energy savings as the time allowed for computation increases. Of course, DVFS is not without its drawbacks. Changing the supply voltage often requires a processor stall, incurring a time delay of 10–100 μs [Von Kaenel et al., 1990]. The energy savings will therefore be obliterated by the overhead of changing voltages unless the system can make accurate predictions about the future performance requirements on the order of milliseconds. Recent work has explored techniques for reducing this delay to the nanosecond range, but there are many complications and this is not yet widely deployed

---

[5] `http://www.mersenne.org`

[6] `http://setiathome.berkeley.edu`

[7] `http://www.distributed.net`

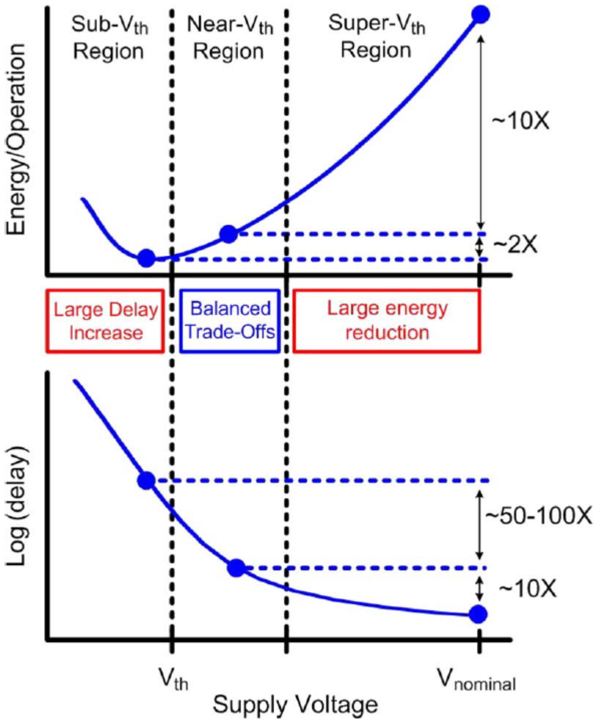

Figure 2.4: Transistor behaviour in sub-, near-, and super-threshold operating voltage regions. Reproduced from [Dreslinski et al., 2010].

in practice [Kim et al., 2008]. A further disadvantage is that the thermal variation induced by varying the voltage may cause microscopic damage to the device and ultimately reduce its lifespan [Lee, 2000]. Notwithstanding these limitations, later chapters discuss the many and varied ways in which DVFS can be leveraged to save energy.

Near-threshold computing

An exciting development at the extremes of the power-performance trade-off is near-threshold computing (NTC). The model embodied by Equation 2.4 does not apply to operating voltages below the threshold voltage, in which region the behaviour of the transistor is slower, more complex, and vastly more erratic; see Figure 2.4. Sub-threshold computing, although certainly low-power, has therefore never found widespread applicability, while traditional computing has reserved itself to operating voltages safely in excess of the threshold. However, NTC operates in the region at which $V_{DD} \approx V_T$, which offers significant energy benefits and only a subset of the difficulties faced in the sub-threshold region: in comparison to normal super-threshold operation, the circuit experiences an order of magnitude reduction in performance; five orders of magnitude increase in memory failures; a fivefold increase in inter-device variability with regards to performance; and doubled increase in sensitivity to operating temperature [Dreslinski et al., 2010]. Recent work has made significant developments in managing the impact of these limitations; in particular, the introduction of highly parallel architectures to ameliorate the loss of performance and device stability, and the use of transistors and SRAM blocks designed for sub-threshold reliability. Further experiments have justified other enhancements, such as the use of dual operating voltages, both in the near-threshold range, to offset a significant fraction of the performance penalty [Kakoee et al., 2010]. On the theoretical side, recent circuit models describe near-threshold behaviour much more precisely than was possible with earlier approximations [Harris et al., 2010]. Superior understanding of the underlying behavioural properties will inevitably lead to improvements in the practicality of NTC, and practical NTC-purposed cores have now been constructed; for example, the Phoenix processor, which operates in the near-threshold region (among many other optimisations), requires only 2.8 pJ per cycle when executing at full speed [Seok et al., 2008]. Including its on-die battery, the Phoenix occupies less than one cubic millimetre and, in a sensor network (its intended environment) can run on this battery for over a year. All of this suggests that NTC may soon find real-world deployment in particularly energy-sensitive devices.

2.3 Energy-aware operating systems

The operating system makes multiple important contributions to the overall picture of an energy-aware system. First, while the hardware can address some power concerns directly, it can also provide facilities for higher layers to exploit; for example, DVS simply allows performance to be traded for power, and it is the task of the operating system to set the proportions of this trade-off in order to achieve actual savings. Making these decisions is the motivating problem for the area of energy-aware scheduling, and a full description of this work is left for Chapter 4 where it is more pertinent. Such decisions may theoretically be made in hardware, but in general there are typically some features that are better handled by a higher level.

A second factor in the operating system's contribution to energy-awareness is that the kernel is itself a process that must be executed and therefore requires energy. In the future we may see operating systems optimised for energy as any other piece of software might be. However, the operating system presents some unique opportunities. One particular example is the recent development of a "tickless kernel", detailed in Section 3.5.1, which dispenses with the traditional polling-based implementation of thread preemption. Similar re-examination of fundamental operating system constructs has been considered in other operating systems, such as OpenSolaris' Tesla Project.[8]

2.4 Energy-aware protocols

Computing is a world of protocols, but in reference to energy-awareness, one usually means networking protocols; there has been little research on energy usage for internal protocols such as APIs at the operating-system level. Network protocol design is an important field: 1997 figures suggest that around 18% of a laptop's power is expended in the wireless card, and, while there is a dearth of comparable measurements for modern hardware, the current figure is perhaps higher, since the hardware improvements of Section 2.2 do not substantially impact the cost of radio broadcast itself [Stemm and Katz, 1997]. Within networking protocols, one can distinguish two independent strands of work: on the one hand, energy concerns in communication protocols, which typically means modifying the existing protocol stack to improve energy efficiency; on the other hand, creation of new protocols intended for energy management *per se*. This section describes each in turn.

---

[8] The Tesla Project is no longer active, although it did make some lasting contributions such as the PowerTOP power monitoring tool, `http://hub.opensolaris.org/bin/view/Community+Group+pm/powertop`.

2.4.1 Energy-aware data protocols

Wireless networks are typically taxonomised into *infrastructure* networks, in which wireless devices communicate with a wired base station in a single hop, and *ad hoc* networks, in which multiple mobile devices communicate amongst each other, and may be completely isolated from any non-mobile power source. Clearly these pose quite different challenges from an energy perspective. When a fixed device transmits to a mobile device, it is worth expending a substantial amount of power in the transmitter to save power in the receiver, and similarly when the mobile device is to transmit, *mutatis mutandis*. On the other hand, when communication is mobile-to-mobile, interacting concerns of efficiency and fairness arise. This summary will mainly consider infrastructure networks since this is the underlying assumption of later chapters. The Open Systems Interconnection (OSI) model is the standard characterisation of the layers of the network stack, so this summary categorises the relevant work according to these layers [Zimmermann, 1980].

The challenges of the physical layer are well-known, such as hidden and exposed terminals and frequency-band collisions. No more will be said about these directly, although the tight layer integration requires close consideration of the physical layer in what follows.

At the data link layer, the focus is on reducing the overhead of retransmissions. Two techniques dominate: forward error correction (FEC) and automatic repeat request (ARQ). FEC extends packets with redundant information such that, up to some maximum number of bit errors, the original information can be recovered even if some of the data is damaged in transit. Clearly this redundancy is a data overhead in itself, and the optimal trade-off depends on the expected fidelity of the connection. Modern "turbo codes" can very closely approach the Shannon limit, the theoretical maximum for data transmission over a noisy channel [Berrou et al., 1993]. Resultant work also exists on decoding these turbo codes in an energy-efficient manner [Leung et al., 1999]. In ARQ, intermediate nodes in the network detect dropped packets without explicit notification from the end-point and request retransmission immediately, thereby reducing latency and eliminating the need for some metadata packets from the receiver. ARQ may hook directly into the physical layer, suggesting the optimal transmission power, which must balance the obvious trade-off between energy required per packet and the probability of a retransmission being required; existing work suggests that this may well be a profitable approach, although empirical measurements are hard to obtain [Arulselvan and Berry, 2002]. There is a similar trade-off with packet size versus retransmission probability, which has been analysed in an energy-aware context [Modiano, 1999]. The optimal packet size is relatively straightforward to derive given an accurate model of the channel, and surprisingly an adequate model can be learnt relatively quickly, in perhaps 10^4 bits of transmission. These results illustrate the general principle that close integration of layers in the network stack is often necessary for optimisation, and this is no less true for energy than for other performance metrics.

Further illustrating the principle, system designers frequently coalesce the network and transport layers in energy-aware devices, following observations on the substantial energy benefits this can bring [Raisinghani and Iyer, 2004]. In almost all cases this means

"TCP/IP"; the Transmission Control Protocol[9] at the transport layer, above the Internet Protocol[10] at the network layer. Early results indicated that none of the basic variants of TCP performed particularly satisfactorily from an energy standpoint, with TCP Tahoe being generally the least bad among them [Tsaoussidis et al., 2000]. This motivated the development of more suitable TCP variants. Energy-efficient TCP (E^2TCP) was proposed soon afterwards, with several refinements [Donckers et al., 2002]. To reduce data overhead, E^2TCP uses header compression, and supports selective acknowledgements to minimise retransmitted data. Furthermore, while traditional TCP attributes all dropped packets to congestion and consequently backs off rapidly, E^2TCP recognises the distinct problem of burst errors (from, say, wireless interference) as essentially transient, and handles this situation differently. In tests, E^2TCP was shown to have slightly superior data efficiency but vastly superior time efficiency, allowing the wireless card to be placed into a low-power state earlier and to remain so for longer. In tests, the two protocols were used to support real-time streaming of the kind explored in Chapter 5. The energy overhead of each protocol was then calculated by reference to the theoretical minimum energy for transmission. When the channel entered a "bad" state (indicating high probability of bit errors) for two seconds for every twenty seconds of clean transmission, E^2TCP incurred 2% energy overhead to Tahoe's 5%; with more frequent cycling, such as 0.1 seconds of bad transmission for every second of clear transmission, E^2TCP required 6% overhead and Tahoe 10%. This demonstrates not only the amount of energy improvement available for traditional TCP variants but also the headroom for further improvement on E^2TCP.

Higher layers in the OSI model (session, presentation, and application) are addressed in Section 2.5.

2.4.2 Energy management protocols

A newer area is the development of protocols to manage and coordinate power directly. This area emerged in ad hoc wireless networks, where it is a fundamental concern, but now has applications more relevant to this document, such as web server management. For example, consider the following: a modern web service typically adopts a multi-tiered architecture, such as a Web interface to business logic with a database back-end. Each tier is run on a separate machine or set of machines. Therefore, depending on the demands of each tier, the performance of each one may be traded off differently for energy, subject to (for example) a latency constraint on the whole stack. Previous work has developed a protocol based on the "Weighted Feedback DVS" algorithm, by which the tiers may communicate to coordinate the performance-power points selected [Horvath et al., 2007]. In tests, a Linux implementation of the protocol was shown to reduce energy consumption by up to 30% compared to the native power-saving mode. Further developments extend the control loop to include virtualised servers, in which it is not so straightforward to power down or scale back hardware components [Wang et al., 2008]. Empirical evidence in this case suggests that such an approach can reduce average power from 240 W to around 205 W for an example system, while still maintaining adequate average response time. Future work may explore the coordination of other hardware and software configuration choices.

---

[9] See RFC 793, *Transmission Control Protocol: Darpa Internet Program: Protocol Specification*, `http://www.ietf.org/rfc/rfc793.txt`.

[10] See RFC 791, *Internal Protocol: Darpa Internet Program: Protocol Specification*, `http://tools.ietf.org/html/rfc791`.

2.5 Energy-aware software

Energy efficiency at the higher layers of the OSI stack—the session, presentation, and application layers—is naturally a highly domain-specific problem and therefore the focus has been on providing developers with tools to profile their own implementations and improve performance directly. An early contribution to this area was PowerScope [Flinn and Satyanarayanan, 1999]. PowerScope is analogous to a traditional profiler, but measures the energy rather than the time and space demanded by each point in the call stack; then, as with CPU or memory profiling, the programmer can direct his or her efforts towards improving the most energy-intensive parts of the application. In an example given by the authors, such analysis and improvement allowed a 46% reduction in the energy consumption of a certain video decoding program. A measure of the success of this work is that most mobile device manufacturers now produce their own tools to measure the power usage of various hardware components, such as Nokia's Energy Profiler[11].

Another general approach targeting the software layer is energy-aware compilation. Typical compiler techniques are manipulations intended to improve the energy efficiency of the program with little or no loss in performance, and therefore they form part of the general literature of optimising compilers, albeit with a novel metric of optimisation. One approach is based on profile-driven compilation, in which measurements of the program's behaviour are taken on representative input data during compilation and used to optimise various code features for that data. For example, one example system looks for regions that are mostly memory-bound and, using DVFS, slows the CPU in these regions [Hsu and Kremer, 2003]. This system is carefully designed for practical use, allowing for non-CPU static power, transition time and energy penalties, battery discharge characteristics, and other details. Consequently, when measured on a real laptop, it produces energy reductions for various benchmarks averaging 11%, reaching 28% in the most favourable cases although with essentially no benefit in the hardest cases; performance is only reduced by a few per cent. An alternative approach, requiring more programmer intervention, is to insert "checkpoints" at the beginning of every basic block, and require annotations describing the maximum time that execution may take to make all possible transitions between these checkpoints; in other words, to ascribe maximum time values to every arc in the control flow graph [Azevedo et al., 2002]. A run-time system, again using DVFS, adjusts the performance within each block to meet these values in a best-effort fashion; it is shown empirically that a good heuristic for estimating the frequencies is the *minimum* time between the checkpoints. This is shown to reduce the energy of the processor by up to 82%, although whole-system values are not provided.

Register allocation has been another popular mechanism to address. Traditional metrics, such as minimising register spillage, certainly correlate positively with energy efficiency, but more can be done. For example, successive reads of a register allocated to the same variable are not relevant to traditional liveness analysis, which defines the lifetime of a variable as spanning from its first write to its final read; however, each of these reads incurs an energy cost, which must be factored in to the total analysis. At the register allocation phase, one can also employ a detailed model of register and main-memory energy consumption, including the benefits of locality of reference in main memory. One implementation of this technique was able to reduce the energy consumption of the memory hierarchy by 87–90%, under the assumption that a memory access requires about thirty times as much power as a register access [Zhang et al., 2002]. However, this was only evaluated against small example programs, although the compilation technique is efficient enough computationally to attack larger examples.

---

[11] `http://www.developer.nokia.com/Resources/Tools_and_downloads/Other/Nokia_Energy_Profiler/`

There are numerous other strands of work in this area. For example, one might aim to reduce register spills (even with a large L1 cache) or to reduce the activity factor [Tiwari et al., 1994, Su et al., 1994a]. As seen in Equation 2.1, the leakage energy is another relevant factor, and an increasingly significant one in modern architectures. Leakage can be reduced by disabling unused functional units within the CPU. A compiler implementation has been presented that inserts explicit enable/disable instructions into basic blocks according to which units are required, subject to the competing consideration of the units' start-up latencies, and was able to reduce average leakage by 45.4% over various examples from the MediaBench and Spec benchmark suites [Zhang et al., 2003]. This is particularly beneficial on complex architectures, which provide various subunits such as a floating-point co-processor, which, while vital for some applications, are largely irrelevant for others, and can be disabled completely.

This document will not consider compilation techniques much further, but rather assumes that the software is provided as-is in binary form, and primarily targets techniques that could be provided by the operating system rather than the application programmer. This is motivated by the argument that software engineers are already overwhelmed with competing considerations, and would be unlikely to extend their development cycles in pursuit of reducing an energy bill incurred, for the most part, by someone else (their customers). This is discussed further in Chapter 3.

2.6 Summary of related work

Evidently, a vast range of work has been undertaken on the energy problem, and there is much more to come. At the hardware level, Koomey's Law continues to drive up the number of computations per unit energy, and to provide new features for exploitation by higher layers. In the operating system, the multicore era creates highly complex scheduling problems which must still be solved quickly if they are to remain essentially transparent to the user. Energy-aware protocols impact more and more devices as Internet connectivity becomes expected on ever-smaller and more portable devices. And software engineers, already challenged by the demands of highly parallel programming, now face a second front from the demands for energy-efficient operation at every level of the execution stack.

Furthermore, beyond the four-piece taxonomy used above lie many interactions between these areas; for example, the shift to a large number of simple cores may ultimately reduce the necessity of kernel quiescence, since waking one core among thousands is a much less significant event than waking a single monolithic processor. To give another example, the low voltage of near-threshold computing offsets some of the difficulties of power dissipation in three-dimensional circuit integration. And there are further issues external to the taxonomy given here; for example, power generation, while certainly outside the scope of computer science, impacts the way in which power can be used within a computing context. However, these details are rather specific and are left for the relevant chapters.

This document does not, of course, attempt to tackle all of these problems at once, but rather argues that to improve on certain details it is valuable to understand the full picture. In my own contributions, I generally take the hardware as a given, representing an immutable background above which other optimisations take place. The protocol stack is significant firstly because it suggests the sort of devices that will be expected to be networked in the future (which is to say: all of them) and secondly because it gives an idea of the performance and characteristics of networking on those devices; the latter is of particular significance in Chapter 5. I also generally consider the software layer to be out of reach, because the diversity of software energy challenges is too great to describe completely general methods for them; also, as discussed, the pressures on modern software engineers are already substantial, and therefore techniques making further demands on them (such as annotation) are unlikely to gain traction in industry. Instead, my focus lies primarily in the middle: in deciding how the hardware's features may be best used to improve the energy of the system, which decisions the operating system can make transparently, and, contrariwise, which features the system might expose directly to the software layer. This is the topic of the succeeding chapters.

Chapter 3

Towards energy-efficient computing

The majority opinion in both corporate and academic circles is that the "energy problem" is now real and pressing; humanity's primary sources of energy are running out while the demand for energy in commercial and domestic environments is increasing, and the side-effects of energy use have important environmental considerations on a global scale. The emission of greenhouse gases such as CO_2, now seen by most climatologists to be linked to global warming, is only one issue.

World leaders and pre-eminent scientists are perhaps most focused on a strategic solution: the need to develop new sources of clean and renewable energy if humanity is to ultimately overcome its energy problem. Lord Rees, president of the Royal Society, emphasised its urgency in an annual address delivered in 2008, saying[1]

"At this year's G8 summit, in Japan, the member nations formally espoused the goal of reducing global CO_2 emissions, by 2050, to half the 1990 level. …Realistically, there is no chance of reaching this target, nor of achieving real energy security, without new technologies."

However, the realisation of new sources of sustainable energy is expected to be at least three decades away. Steve Chu, director of the Lawrence Berkeley National Laboratory prior to his appointment as United States' Secretary of Energy, placed this in context, saying[2]

"A dual strategy is needed to solve the energy problem:

1. Maximise energy efficiency and decrease energy use.

2. Develop new sources of clean energy.

Number 1 will remain the lowest-hanging fruit for the next few decades."

---

[1] Quoted from Lord Martin Rees of Ludlow's Anniversary Address to the Royal Society, 2008, `http://royalsociety.org/uploadedFiles/Royal_Society_Content/about-us/history/Anniversary_Address_2008.pdf`.

[2] Quoted from *The energy problem and Lawrence Berkeley National Laboratory*, talk given to the California Air Resources Board in February 2009.

Component	Energy (2000)	% Total	Energy (2006)	% Total	CAGR
Site infrastructure	14.1	50%	30.7	50%	14%
Network equipment	1.4	5%	3.0	5%	14%
Storage	1.1	4%	3.2	5%	20%
High-end servers	1.1	4%	3.2	5%	20%
Mid-range servers	2.5	9%	2.2	4%	-2%
Volume servers	8.0	29%	20.9	34%	17%
Total	28.2	–	61.4	–	14%

Table 3.1: Energy usage breakdown for computing equipment in the United States. Energy figures are in billions of kWh.

3.1 Energy in the computing space

Energy is an inescapable problem in computing. Even if future improvements in power efficiency adhere to Koomey's Law and increase the available energy for computation beyond our current imagination, as discussed in Section 2.2.2, there are fundamental limits to what can be computed with a given amount of energy. More currently, power distribution and cooling is already a significant challenge in the data centre, and future hardware refinements are only likely to exacerbate this [Fan et al., 2007]. Put simply, the conclusion of thermodynamics is that energy is the ultimate limited resource in the universe.

In August 2007, the Environmental Protection Agency (EPA) issued a report to the U.S. Congress on the energy efficiency of servers and data centres [U.S. Environmental Protection Agency, 2010]. Some key findings from the report include:

- Servers and data centres consumed 61 billion kilowatt-hours of energy in 2006.
- This was 1.5% of total U.S. electricity consumption that year, amounting to $4.5 billion in electricity costs, which is equivalent to 5.8 million average U.S. households.
- Electricity use in this sector doubled between 2000 and 2006, a trend that is expected to continue.
- Infrastructure systems necessary to support the operation of IT equipment (such as power delivery and cooling systems) also consumed a significant amount of energy, comprising 50% of annual IT electricity use.

Excerpts from the EPA report are shown in Figure 3.1 and Table 3.1. There are two particularly notable points in the data. The first is that as much energy is being consumed by site infrastructure as by the computing equipment itself. This infrastructure primarily represents HVAC (heating, ventilation, and air-conditioning) equipment, as well as that used to convert and transmit power and to maintain its continuity; this includes transformers and in-building power-switching and transmission equipment, as well as power-conditioning and sustaining equipment such as uninterruptible power supplies. This factor is of great consequence because it suggests that energy savings at the computing level would have an attendant impact on HVAC as well.

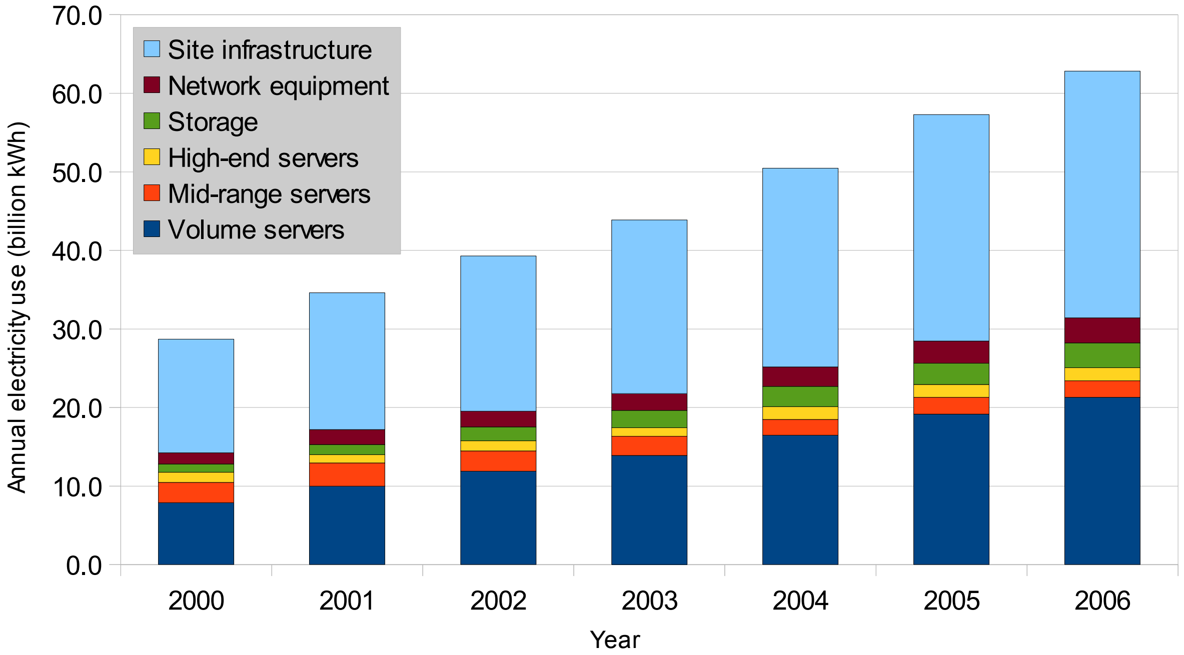

Figure 3.1: Energy usage breakdown by equipment type in United States, from 2000 to 2006.

Within the computing equipment itself, however, is the second point of interest. Of the five types of IT equipment studied, volume servers alone were responsible for the majority (68%) of the electricity used. Assuming that the 17% combined annual growth rate of volume servers continues, this suggests that they are the prime targets for energy reduction in the server space. The 20% growth rate of storage devices shown here—a rate that more recent data suggests is accelerating—indicates another significant trend.

If the exponential growth of data-centre computing equipment revealed by this study continues, the demand for electricity in data centres seen in 2006 is expected to have roughly doubled by the time of writing. This poses challenges beyond the obvious economic ones. For example, peak instantaneous demand is expected to have risen from 7 gigawatts in 2006 to 12 gigawatts in 2011, and ten new base-level power plants would be needed to meet such a demand.

Physical limitations on power availability are already a constraint for data centres in some areas; a managing director of IT for Morgan Stanley observed in 2009 that the company is now physically unable to source the power needed for a new data centre in Manhattan. The seriousness of the situation is demonstrated by the zeal with which corporations such as eBay, Google, Amazon, Microsoft, and Yahoo have pursued suitable locations in which the data centres required to run their contemporary Web applications and services can be constructed [Katz, 2009]. A number of these companies have already negotiated with certain states in the U.S., as well as internationally, to construct these facilities along with the power plants necessary to supply them. In 2006, Google triggered a “multibillion-dollar face-off” after situating a new facility along the Columbia River in Washington [Markoff and Hansell, 2006]. The combined benefits of lower land cost, lower external ambient temperature, and the availability of running water for cooling and hydroelectric power generation are intended to provide relief for Google’s acute energy-availability and energy-cost problems.

The U.S. Energy Information Administration showed in their report to Congress[3] that in 2001, PCs and printers in American households consumed 23.1 terawatt-hours of energy; the figures were similar in 2006 [Roth and McKenney, 2007]. This suggests that the amount of energy consumed by mobile and desktop computing equipment is of roughly the same magnitude as that used by servers in data centres, although there is no correspondingly comprehensive and authoritative current study to refer to. The EPA data presented here provides some detailed perspective on where the energy goes in the important and growing server segment of the computing landscape. Also, some groundwork has already been laid in the mobile and desktop computing space as a result of the earlier focus of the EPA's EnergyStar program on consumer electronics, which includes computer systems.

3.2 Modern power management

Perhaps the key factor to consider with today's computer systems is that the amount of power they consume does not adjust gracefully according to the amount of work the system is doing. The principal design objective for most general-purpose computer systems to date has been to maximise performance (or perhaps performance at a given price point) with very little consideration given to energy use. This is changing rapidly as we near the point where the capital cost to acquire computing equipment will be exceeded by the cost of energy to operate it, even over its relatively short (three- to five-year) amortisation period, unless we pay some attention to energy-conscious system design.

The case has been made for energy-proportional computing [Barroso and Hölzle, 2007]. This is the design ideal in which the amount of power a system (or component) requires corresponds directly to its degree of utilisation. However, this is far from the current situation. Many components of computer systems today exhibit particularly poor efficiencies at low levels of utilisation, and most systems spend a great proportion of their time operating at relatively low usage levels. Power supplies have been notorious for their inefficiency, especially when at low load, and fans can also waste a significant amount of energy when operated carelessly. In just four years, however, the efficiency of power supplies has improved markedly[4], and algorithms are emerging that continuously adjust fan speeds in response to thermal need, rather than using just a few discrete speed points. The majority of hardware components in today's computer systems must still be managed explicitly, however, and the current widely deployed conceptions and facilities for power management in computer systems remain rudimentary.

There are two basic modalities for power management: a running versus suspended (not-running) aspect in which a component (or whole system) can be powered off when not in use and turned on again when needed; and a performance-adjustment aspect (while running) in which the performance level of a component can be lowered or raised in reaction to either the observed or predicted level of utilisation or other needs of the workload.

The running versus not-running choices are often called the component's (or system's) power states. Typically a single state represents running, while different levels of suspension may be distinguished into multiple states. The latter allows power to be progressively removed from more of the hardware associated with the component (or system) if there is some important power-relevant structure to its implementation. CPUs, for example, may have their execution suspended simply by stopping the issuance of instructions or by turning off their clock circuitry. “Deeper” power states, however, might successively remove power from the processor’s caches, translation lookaside buffers, memory controllers, and so on. While more energy is saved as more of a component’s hardware has its power removed, there is then either a greater latency to recommence its operation, or extra energy is required to save and restore the hardware’s contents and restart it, or both.

---

[3] Energy Information Administration Residential Energy Consumption Surveys 2001, `http://www.eia.doe.gov/emeu/recs/recs2001/enduse2001/enduse2001.html`.

[4] Recent standards for power supply efficiency; `http://www.80plus.org`

The performance-adjustment choices while running are most naturally called the component’s *performance states.* A widely applied technique for adjusting performance is to change the component’s operating frequency. When clock speed is slowed, operating voltage levels can also be reduced, and these two factors together—the dynamic voltage–frequency scaling of Section 2.2.3—result in a compound power saving, a fact exploited in later chapters. Performance states were first introduced for CPUs, since processors are among the most consequential consumers of power on the hardware platform (something in the range of 35 to 165 watts is typical of a contemporary multicore CPU). Performance states might also be used to control the active cache size, the number and operating rates of memory and I/O interconnects, and so on.

3.2.1 ACPI

For reference in later discussion, it is worth presenting the most widely implemented architecture for power management in use today: the Advanced Configuration and Power Interface (ACPI). ACPI has evolved together with the Intel architecture, the hardware platforms based on the most widely available commodity CPUs and related components. Although there are many detailed aspects to the specification, ACPI principally offers the controls needed to implement the two power-management modalities just described. It defines power states: seven at the whole-system level, called S-states (S0–S6); and four at the per-device level, called C-states (C0–C3) in the case of CPUs and D-states (D0–D3) for other devices. The semantics of each non-running power state are specific to the device (or device class) in question. The zero-numbered state—S0 for the system, or D0 for each device—indicates the running (or active) state, while the higher-numbered states are non-running (inactive) states with successively lower power and correspondingly decreasing levels of availability (run-readiness). ACPI also defines performance states, called P-states (P0–P15, allowing a maximum of sixteen per device), that affect the component’s operational performance while running. Both power states and performance states affect power consumption.

3.3 Energy efficiency in computing

Although ACPI is an important *de facto* standard with reasonably broad support from manufacturers, it provides nothing more than a mechanism by which aspects of the system can be controlled to affect their power consumption. This enables but does not explicitly provide energy efficiency. Higher-level aspects of the overall system architecture are needed to exploit this or any similar mechanism.

One might ask how energy-efficient computing differs from power management, and how one would know that the energy-efficiency problem had been “solved” for a given computer system. A simple criterion might be: the system consumes the minimum amount of energy required to perform any task. In this formulation, energy efficiency is simply an optimisation problem. Such a system must adjust the system’s hardware resources dynamically, so that only what is needed to perform those tasks (whether to complete them on time, or analogously, to provide the throughput and latency required to maintain a stated service level) is made available, and that the total energy used is minimised as a result. Section 2.1 demonstrated that this formulation is inadequate, but it is instructive to consider the complications of even this simplest of objectives.

Traditionally, systems have been designed to achieve maximum performance for the workload. In an energy-efficient system, maximum performance for some tasks (or the whole workload) will still be desired in some cases, but even within that constraint the system must also minimise energy use. Performance and energy efficiency are not mutually exclusive. For example, even when achieving maximum performance, any resources that can be deactivated, or whose individual performance can be reduced without affecting the workload’s best possible completion time or throughput, contribute to energy optimisation. Indeed, for a typical desktop or server machine, it is almost impossible to simultaneously demand the full performance of every component; in fact, this has security consequences seen in the relatively young field of study surrounding “thermal viruses”, malicious code designed to place specific parts of the system under maximum energy load, thereby damaging them or overloading the cooling system [Dadvar and Skadron, 2005, Hasan et al., 2005]. Systems that strive to achieve maximum performance at all times are notoriously over-provisioned and correspondingly under-utilised; however, dynamic capacity planning and provisioning is a difficult problem and is far from solved.

Energy optimisation is obviously subject to certain constraints. Some examples follow.

3.3.1 Maintenance of required performance

The system must generally endeavour to meet deadlines, where they exist. In the general case, a deadline is specified for a task or the workload. When any deadline is specified that is less than or equal to the optimum that the system can achieve with any or all of its hardware resources, this can be taken to maximum performance; this is effectively the degenerate case. Maximum performance for a task or the workload provides an implicit stipulation of the optimal deadline t_o, or “as soon as possible”. All values of the deadline D less than the shortest achievable deadline t_o is equivalent to setting $D = t_o$. We can therefore denote maximum performance by $D = 0$. In this case, energy optimisation is restricted to those resources that can be deactivated, or whose individual performance can be reduced, without affecting the workload’s best possible completion time; alternatively, there may be several resource configurations that produce the same latency and throughput, in which case optimisation is free to select among them according to lowest energy, instantaneous power, or other thermal concerns.[5]

If a deadline later than the best achievable deadline is specified, the computation may take any length of time up to this deadline, and the system can seek a global energy minimum for the task (or workload). Deadlines might be considered “hard”, in which

---

[5] This suggests a higher-dimensional version of the Pareto frontier that will be introduced in Chapter 5.

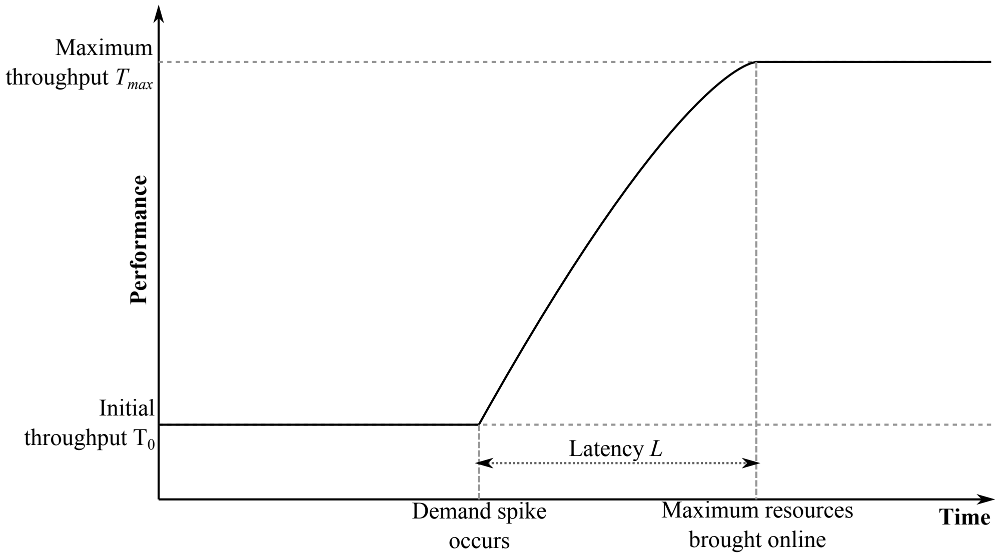

Figure 3.2: Time-throughput diagram showing latency to reach maximum performance.

case the system's energy-optimising resource allocator must somehow guarantee to meet them (raising difficult implementation issues), or "soft", in which case a best effort can be tolerated. In the latter case, some sort of "quality of service" metric must be provided, describing how, when, and by how much deadlines may be missed.

Services must operate at their required throughput. For online services, the notion of throughput may be more appropriate than that of a completion deadline. Since services, in their implementation, can ultimately be decomposed into individual tasks that do complete, we expect there to be a technical analogue, although the most suitable means of specifying its performance constraint might be different.

3.3.2 Response to changes in demand

Real workloads are typically not static: the amount of work provided and the resources required to achieve a given performance level will vary over the course of execution, and from instance to instance. Dynamic response is an important practical consideration related to service level. Figure 3.2 shows the relationship diagrammatically; stated formally, the system must be able to reach its maximum throughput T_{max} from any throughput T_0 within latency L. Therefore, whenever a spike in the demand for the service occurs, the system can react to it, re-enabling or scaling up the performance of the relevant components, within the given time limit. Specification of the maximum latency within which reserved hardware capacity can be activated or its performance level increased seems a clear requirement, but this must also be related to the performance needs of the task or workload in question.

The meaning of "throughput" is dependent on the nature of the task. A metric such as transactions-per-second (TPS) might be relevant for database system operation, triangles-per-second for the rendering component of an image-generation subsystem, or corresponding measures for a filing service, I/O interconnect, or network interface. Interactive use imposes real-time responsiveness criteria, as does media delivery: computational, storage, and I/O capacity required to meet prescribed audio and video delivery rates. A means by which such diverse throughput requirements might be handled in practice is suggested below.

3.3.3 Power capping

Instantaneous power must never exceed a specified power limit P. A maximum power limit may be specified to respect practical limits on power availability, whether to an individual system or to some aggregated structure, such as a data centre. In some cases, exceeding this limit briefly may be permissible, although this requires a more detailed system description to manage the thermal consequences.

3.3.4 Summary of constraints

Other constraints are also possible. Combinations of such constraints mean that over-constraint must be expected in some circumstances, and therefore a policy for constraint relaxation will also be required. A strict precedence of the constraints might be chosen or a more complex trade-off made between them. This area is not currently well-understood.

3.4 Directions toward a solution

Given this concept for energy-efficient computing, a natural question is how such a system might be constructed, and in general, how one would expect an energy-efficient system to operate. A system has three principal aspects that could solve this problem, each of which represents a significant body of existing work.

1. The system must be able to construct a power model of how and where power is consumed, and how it can manipulate that power, since this is the basis for enacting any form of power management.
2. The system must have a means for determining the performance requirements of tasks or the workload, whether by observation or by some more explicit means of communication. This is the constraints-determination and performance-assessment component.
3. Finally, the system must implement an “energy optimiser”: a means of deciding on an energy-efficient configuration of the hardware at all times while operating. The optimisation may be relative (heuristically decided) or absolute (based on analytical techniques). This is the capacity-planning and dynamic-provisioning component.

Each of these are discussed in detail below.

3.4.1 Power model

In order to manage the system’s hardware for energy efficiency, the system must know the specific power details of the physical devices under its control. The “system” here most naturally suggests the operating system, although it is clear that this must include the hypervisor for virtualised systems; one can reasonably expect that this concept will need to be broadened to include some aspects of the firmware and even hardware components (on the low end) and important runtimes, such as the Java Virtual Machine, which have responsibility for, or particular knowledge of, resource allocation. Power-manageable components must expose the controls that they offer, such as their power and performance states (D-states and P-states, respectively, in the ACPI architectural model). To allow modelling of power relative to performance and availability—in other words, relative to its activation responsiveness—the component interface must also describe at least the following:

- The per-state power consumption (for each inactive state) or power range (for each active state).
- State-transition latency (time required to make each state transition).
- State-transition energy (energy expended to change state).

Once the system has such a power model, consisting of all its power-manageable hardware, it has the basic foundation for operating to optimise energy. Importantly, it has the knowledge of those components that consume the most power and those that have the most highly responsive controls that can be used to affect power use and effect power reduction.

3.4.2 Workload constraints and performance assessment

In order to impose appropriate constraints on the optimisation of active hardware and energy consumption, a system must also be able to measure the throughput of its applications to ensure that they still meet the relevant service levels and deadlines. The assessment of throughput is subject to the task or application in question. The operating system can observe the degree to which its various resources have been and are currently being used, and it might use these observations as its best basis for prediction of future resource needs, thus shrinking or enlarging what is available. This is a relatively weak basis to determine what the workload will need, especially to anticipate its dynamic responsiveness sensitivities. As a result, the system will inevitably be much more conservative in its reduction of available resources or their performance levels. It seems clear that the best result will be realised if applications assess their own throughput relative to their service-level requirements or completion deadlines, and can convey that information to the operating system through an interface. This is counterbalanced by the desire to avoid overloading the application programmer with additional programming language complexity. However this is resolved, the system can then use this information to make potentially much more aggressive resource adjustments and realise an improved overall energy-optimisation solution accordingly.

Perhaps the suitable division of labour is to make the system responsible for solving the energy-optimisation problem subject to the resources it allocates, while the application is responsible for monitoring its own performance level and informing the system so that appropriate resources can be provided to meet them. Thereby the application needs only a domain-specific metric of performance to feed back to the operating system; it does not need to be concerned with the details of energy *per se*.

3.4.3 Energy optimisation by the system

Once provided with the hardware's power characteristics, and possibly descriptive information from application-level software about its constraints, the operating system must begin the dynamic process of adjusting the hardware's performance and availability levels to control power consumption and improve system-wide energy use. The following techniques might form part of this decision.

Heuristic methods

Provisioning for maximum throughput may, in some cases, optimise energy. This is the conjecture that "[maximum] performance is green," reflected in the ideas of race-to-idle or race-to-sleep [Garrett, 2007]. Although there is some evidence that this approach has merit in client-side computing when the system becomes idle—especially for embedded and mobile systems where 95% of the energy may be saved if the entire system can be put into a suspended state—it is not clear how applicable this is to server-side computing. A super-linear increase in the power required to get linear speed-up (or throughput) exists in some cases (Intel's Turbo mode on contemporary CPUs being one example) and hence, the energy optimum will not be found at a provisioning and performance point commensurate with maximum throughput in all cases.

A widely used heuristic for energy improvement on active systems is to adjust the hardware's performance level dynamically, based on its current utilisation: downward with low utilisation or upward with high utilisation (with some hysteresis). This can be an effective technique but is restricted to situations in which both the latency and the energy to make the state change are so low as to be inconsequential.

Constrained optimisation

In some cases, it may be possible to idealise the problem into precise mathematical models, and from these derive a complete analytical solution. For example, if we consider only a single task on a single CPU with a well-understood power–performance trade-off, it is relatively straightforward to completely specify a schedule in which the task will meet its deadline with the minimum total energy; more general formal results are also possible, as demonstrated in Chapter 4. This relies, however, on a number of assumptions, such as good estimates of the total work required by a process, which are often uncomputable or hard to obtain in practice. Weaker assumptions can also lead to formal work, as demonstrated in Chapter 5, although, in existing systems, optimisation is typically performed by online heuristic algorithms. There is some existing work in this area but not yet enough to underpin a general-purpose operating system [Yao et al., 1995].

For an optimisation-based approach to be generally applicable, a range of techniques will be necessary. In the simplest cases, autonomous device-level operation is possible; for example, at the hardware level, a graphics co-processor (GPU) can power down unused hardware pipelines aggressively, based solely on instantaneous assessment of their utilisation levels, because the latency to bring those pipelines back up as they become necessary is inconsequential. Similar practices appear to be applicable in the use of CPU P-states, since both the state-transition energy and latency are very low.

Hardware state changes that affect power require a different treatment if they exhibit a much greater transition latency or energy. An obvious example is spinning down a hard disk, considering the long latency to bring it back to full performance, but reactivation latency is not the only concern. Semiconductor memory systems in which part of the total physical memory could be powered off if not required, and where power-on latency may be near zero, will still have a consequential transition energy, since a great many in-memory transactions may be required to gather the working set into those physical pages that will remain active. One might consider whether traditional heuristics might have analogues in energy optimisation, such as the Five-Minute Rule, proposed in 1986 as a memory hierarchy heuristic and renewed several times since [Gray and Putzolu, 1987, Gray and Graefe, 1997, Graefe, 2007]. Resources of this class require greater knowledge of the task or workload behaviour, as well as an anticipatory treatment of the required hardware resources, to ensure that the activation latency can be tolerated or managed and that the state-change energy will be exceeded by the energy that will be saved while in that state.

Some common optimisation techniques may be based on state-change latency, their energy demands, and so on, and a taxonomy of such techniques might arise from this, some formal or analytical, some based on more numerical or heuristic methods.

Although we expect the specific techniques for energy optimisation appropriate to different hardware resources or subsystems to be somewhat different, subject to the properties of the hardware resources in question, the hope is that the composition of energy-efficiency optimisers for all such resources will accumulate to form an efficiency scheme for the whole system. On the other hand, such reductionism may be overly optimistic if there are interactions between the resources allocated by different subsystems, and a more holistic approach may then be necessary in systems for which “every joule is precious” [Vahdat et al., 2000].

3.5 Routes towards energy efficiency

The vision of system-wide concessions to energy efficiency cannot be accomplished in a single swift step. Today’s systems software is not equipped in the ways described, nor are applications written in a way that could exploit that capability. This section discusses how this outcome might be achieved in practical terms, and what steps are already underway.

3.5.1 Considerations for the operating system

As a first consideration, systems need to be revised to pay attention to their use of energy; even the operating system itself, while always running, has not yet been optimised in its own use of energy, as discussed in Section 2.3. To date, almost all software, including systems software, has been optimised (quite understandably) for performance, robustness, and scalability with no consideration of energy. An initial step, therefore, is the redesign and implementation of the operating system so that its operation is energy efficient. This is a significant undertaking, and its full implications are not yet well understood. It is not clear whether modifying existing operating systems to consider energy as a first-class constraint is feasible, although this would certainly be preferable. Experience with system security, multi-tasking, multimedia and so forth suggests that introducing such fundamental considerations after the fact is fraught with complications [Loscocco and Smalley, 2001, Leslie et al., 1996]. We can certainly anticipate fundamental new structures within systems software, and perhaps even that new operating systems will emerge as a result of the energy-efficiency pressure.

At the very least, resource-management facilities within the operating system must be adapted for energy awareness, and then for energy optimisation. This section surveys the components common to most server systems and the energy issues which surround them.

Processor efficiency

A significant fraction of power on contemporary computing platforms can be attributed to CPUs (and the early introduction of power-management features on them as a result), and much progress has already been made with operating-system schedulers and thread dispatchers. Reactivation of idle hardware components when there is no useful work to be done is a common culprit; polling within the operating system (or within applications) is an obvious example, and the use of a high-frequency clock-tick interrupt as the basis for timer events, time-keeping, and thread-scheduling can be equally problematic. If the OS supports preemptive scheduling, this is typically implemented by scheduling a regular interrupt, known as a *tick*, that passes control to the OS and allows it to select which thread is to be executed. Both kernel space and user space permit the scheduling of further regular timers to handle periodic events in a non-blocking fashion; this may be used as the basis for a polling system, for example. From a design and performance standpoint, this is preferable to a busy-wait methodology when implementing regular events, but does present an unfortunate interaction with CPU sleep modes. If a timer is scheduled for, say, every 10 ms, an otherwise-idle CPU is woken by an interrupt at least this often, and is therefore unable to enter a deep sleep state or to remain in a shallow sleep state for any substantial period of time. As the number of timers increases, each with an independent offset and period, the length of time until the next interrupt dwindles rapidly, and this has a significant impact on power efficiency, especially for devices for which the CPU is the pre-eminent power consumer. One proposed solution to this is the "tickless kernel", for which a working implementation is now available in Linux [Siddha et al., 2007]. This introduces several techniques to reduce the number of interrupts generated. One is the concept of a *jiffy*, which is an approximate unit of time used to schedule events for which the precise periodicity is not important; the kernel then coalesces all events to jiffy boundaries and executes them at once. For example, instead of scheduling twenty events with 8 ms of work to do each second, the kernel schedules a single interrupt every second which then triggers 160 ms of contiguous work, allowing the CPU to enter a sleep state for the rest of the second. In tests, the number of interrupts on an idle system was reduced by this technique from 2 002 per second to 118, and the average time spent in an idle state increased from 651 μs to 10 161 μs. The tickless kernel also presents a new *deferrable timer* API; a deferrable timer will trigger normally when the CPU is busy anyway, but is deferred when the CPU is idle; many device drivers naturally tolerate the consequent variability in latency. This further doubled the average time between state transitions. This work has motivated sweeping changes to other parts of the OS, middleware, and software, all in pursuit of increasing interrupt quiescence and further decreasing the number of interrupts on an apparently idle system[6].

---

[6] A bug filed against the Fedora kernel, reporting excessive wake-ups caused primarily by driver polling, has at the time of writing aggregated seventy-three dependent fixes! `https://bugzilla.redhat.com/`

The confluence of features on modern processors—CMT (chip multithreading), CMP (chip multiprocessor), and NUMA (non-uniform memory access) for multiprocessor systems with multiple sockets—invites a great deal of new work to implement optimal-placement thread schedulers [Fedorova, 2006]. The ability to alter performance levels and the expected introduction of heterogeneous multicore CPUs[7] will only introduce further opportunities for successive improvement [Shelepov et al., 2009, Fedorova et al., 2009].

Storage efficiency

Compared with CPUs, the power consumed by a disk drive does not seem especially large. A typical 3.5-inch, 7200-RPM commodity disk consumes about 7 to 8 watts, only about 10% of a typical multicore CPU's consumption. Although higher-performance 10 000-RPM spindles consume about 14 watts, and 15 000-RPM drives perhaps use around 20 watts, this is still a small consideration. The alarming relative rate of growth in storage, mentioned earlier, could quickly change the percentage of total power accounted for by storage devices. Performance and reliability factors have already resulted in the common application of multiple spindles to implement a simple RAID solution, even on commodity desktop systems. In the data centre, storage solutions are scaling up faster still. Low-end volume server boxes now routinely house a dozen or more drives; an example 4U rack-mount storage array product from Sun accommodates 46 3.5-inch drives. A single instance of the latter unit, if it used 10 000- or 15 000-RPM industrial drives, might therefore account for 1.1 to 1.6 kilowatts, rather a more significant energy-use picture.

Storage subsystems are now obviously on the radar of the energy-attentive. There are at least two immediate steps that can be taken to help improve energy consumption by storage devices. The first is direct attention to energy use in traditional disk-based storage. Some of this work has been started by the disk hardware vendors, who are beginning to introduce disk-drive power states, and some has been started by operating-system developers working on contemporary file systems (such as ZFS) and storage resource management. The second, particularly derived from the recent introduction of large inexpensive flash memory devices, is a more holistic look at the memory hierarchy [Graefe, 2007]. Flash memory fills an important performance/capacity gap between main memory devices and disks but also has tremendous energy-efficiency advantages over rotating mechanical media [Leventhal, 2008, Mogul et al., 2009].

Memory efficiency

Main memory, because of its relatively low power requirement (on the order of 2 watts per DIMM), seems at first glance to be of even less concern than permanent storage. Its average size on contemporary hardware platforms, however, may be poised to grow rapidly. With hardware system manufacturers' focus primarily on performance levels (to keep up with the corresponding performance demands of multicore CPUs), maintaining full CPU-to-memory bandwidth is critical. The consequence has been an evolution from

---

`bugzilla/showdependencytree.cgi?id=204948`

[7] A heterogeneous CPU means, in this instance, a multicore CPU in which cores of different performance levels and different microarchitectures are included in the same multicore package; the power consumption characteristics are consequently very different to those of traditional single-core or homogeneous multicore processors.

single- to dual-channel and now to triple-channel DIMMs along with the corresponding DDR, DDR2, and DDR3 SDRAM technologies. Although reductions in the process feature size (DDR3 is now on 50-nanometer technology) have enabled clock frequency to go up and power per DIMM to somewhat decline, the desire for even greater performance via an increase in DIMMs per memory channel is still increasing the total power consumed by the memory system.

For example, a current four-socket server system (based on the eight-core Sun Niagara2 CPU) with 16 DIMMs per socket using DDR2 dual-channel memory technology has 64 DIMMs total. This would increase to 24 DIMMs per socket (96 total) if its faster successor used DDR3 triple-channel memory instead. A representative DDR2 DIMM consumes 1.65 watts (or 3.3 watts per pair), whereas the lowest-power edition of the current DDR3 DIMMs consumes 1.3 watts (or 3.9 watts per triplet). The result appears to be an increase in power consumption of only 20%, increasing from about 100 to 120 watts total in the example given.

Since the next-generation CPU will also have twice as many cores per socket, however, a possible scenario is also to desire twice the number of memory sets per socket (for a possible 192 total DIMMs) to balance overall memory system performance. The result, therefore, could be an increase from 100 watts to 240 watts: a 140% increase in power consumption for the whole memory system! This trend is even being observed on desktop-class machines, admittedly at a much smaller scale, as systems containing quad-core hyperthreaded CPUs (such as Intel's Nehalem) have appeared.

If available physical memory is to be enabled and disabled, and perhaps correspondingly reconfigured as a system's processing capacity is dynamically adjusted, some new functionality will be required of the operating system's memory-management subsystem. The design of a future-looking virtual memory system that is energy-aware and able to adjust physical memory resources while running is an open problem.

I/O efficiency

Energy aspects of the I/O system on hardware platforms will likely become more important as well. As a simple example, present-day local-area networking interconnect and subsystems have evolved in two important respects: link-aggregation is increasingly used to bolster network bandwidth and reliability; and individual interconnect speed has advanced from 1 Gb to 10 Gb, with 40 Gb on the horizon. A transceiver for a 10-Gb network interface card may now require as much as 14 watts when operating at full speed, with a consequential power reduction when its link speed is reduced to 1 Gb or lower (about 3 watts at 1 Gb, 1 watt at 100 Mb). Other high-speed interconnects such as InfiniBand can be expected to have similar energy implications for the overall system.

3.5.2 The evolution of application software

The most strategic aspect of energy-efficient computing will be the evolution of application software to facilitate system-wide energy efficiency. Although we can certainly expect new application interfaces to the system software, supporting the development of new energy-efficient applications, the transition of historical and present-day applications represents a long-term evolution. There is a significant question as to how the problem of greater energy efficiency for the remainder of the installed base might be addressed in the interim. Obviously, it will not be brought about as the result of a unique epoch in the implementation of all existing applications.

One possibility for addressing the energy agnosticism of existing applications is to perform extrinsic analysis of their runtime behaviour. Empirical data can be gathered about the degree to which application performance is sensitive to varying levels and types of resource provisioning. For example, one can observe the degree to which performance is increased by the addition of CPU resources, or the allocation of a CPU with higher-performance microarchitecture, and so on [Shelepov et al., 2009]. The application might then be labelled, in its binary form, with its measured degree of sensitivity, without requiring the alteration of its existing implementation. The operating system could use this metadata to assign resources that pursue a certain specified performance level or to locate an appropriate power-performance trade-off.

By analogy with memory management, it seems likely that a combination of techniques will be needed: explicit, in which the application itself informs the system of its throughput and resource provisioning needs; and implicit, in which static and dynamic analysis is used to model resource needs relative to performance and energy consumption.

3.6 Conclusion

We are still at the debut of energy-conscious computing, with a great deal of the industry's attention directed at the introduction and use of power-management mechanisms and controls in individual hardware components rather than the broader problem of energy efficiency: the minimisation of total energy required to run computational workloads on a system. This chapter suggests an overall approach to energy efficiency in computing systems. It proposes the implementation of energy-optimisation mechanisms within systems software, equipped with a power model for the system's hardware and informed by applications that suggest resource-provisioning adjustments so that they can achieve their required throughput levels or completion deadlines.

In the near term, a number of heuristic techniques designed to reduce the most obvious energy waste associated with the highest-power components, such as CPUs, are likely to remain practical. In the longer term, and for more effective total energy optimisation, increasing importance must be attached to techniques able to model performance relative to the system's hardware configuration (and hence its energy consumption), alongside improved understanding of workload prediction. In Chapter 4, I explore one respect, the processor speed, in which the operating system can schedule tasks for greater efficiency.

Chapter 4

Cost optimisation for power-aware computing

4.1 Motivation

In this chapter, I turn to a particular class of problem in the field of energy-efficient computing, and illustrate the value of developing a solution with understanding of the broader context. The problems are those surrounding *energy cost for long-running computations with deadlines.* The existence of a deadline implies that the work is time-sensitive, but that there is no particular benefit to completing the work quickly provided the deadline is respected. Such problems occur most commonly in the server environment; perhaps, for example, a large scientific calculation that must finish analysing the daily dataset before the next day's dataset arrives. This subspace in the vast space of energy optimisation problems is large enough to be of some useful generality, since I do not make any specific assumptions about the type of work that is being undertaken, and only weak assumptions about the nature of the hardware on which the task is run; on the other hand, the subspace is small enough that strong results can be derived in a precise mathematical framework.

In recognition of a consideration common in the server environment, I will be considering a more general problem than total energy consumption. For many institutions, the financial incentive to reduce energy consumption is stronger than the ecological one; in other words, they are more concerned with spending less on energy than using less energy *per se.* Of course, these two considerations are connected and until now have generally been considered identical, and, accordingly, previous work has been most concerned with minimising total energy rather than total cost of energy. But there are good reasons to separate the two quantities. As shown in Chapter 3, American data centres account for 61 billion kWh of electricity every year at a cost of \$4.5 billion, so even a small reduction in this cost would have significant impact on the bottom line [U.S. Environmental Protection Agency, 2010]. The distinction is relevant when power cost varies over time, such as with the daily off-peak period, by season, or with some stochastic element from the energy market. In this chapter I will describe the cost using a cost function parametrised by time. The cost is typically constant over short timespans, but here I am concerned with long-running computations; the simpler case can be recovered by supplying a constant cost function.

The term “cost function” naturally suggests monetary cost, but this need not be the case. For example, cost might incorporate rising user dissatisfaction as a server request takes longer to resolve. By constructing a mathematical model that includes this dissatisfaction alongside real energy usage, cost minimisation could be used to balance user satisfaction with monetary cost, although construction of such detailed models is not attempted here. Nevertheless, this problem and many others could be described and in some cases solved within the framework established here.

My approach is to describe algorithms or, when possible, closed-form equations to decide the rate at which computation should proceed in order to minimise cost while still respecting the deadline. I generally assume that this rate change is effected by dynamic voltage–frequency scaling although other implementations are amenable to a similar analysis. For reasons discussed in Chapter 3, I do not consider source or binary manipulation, but only the “how” and “when” of execution. Of course, it is well known that incorporating power considerations into the scheduling algorithm, alongside traditional time and resource allocation, can be an effective approach to energy efficiency [Hong et al., 1999]. The originality of my contribution derives from the particular class of problem considered, the precise analytical nature of the results, and the generalisation to power cost.

4.2 Related work

Long-running workloads are common across divers areas of computation. Some are purely numerical, such as the Lucas-Lehmer test used by Great Internet Mersenne Prime Search[1], which manipulates huge integers and takes several weeks to determine if large values of a certain form are prime. Others are the largely recreational, such as the two-decade effort to completely solve the game of draughts [Schaeffer et al., 2007]. And still others have significant implications for the security of some of the world’s most important cryptographic systems, such as the contests to break the once-popular DES encryption system by brute-force search of the key space[2]. In these and many other cases, a level of parallelism exists in the problem, but the granularity of efficiently exploitable parallelism is such that even the subproblems are long-running non-parallel workloads in themselves. For example, the Great Internet Mersenne Prime Search allocates candidate Mersenne numbers to individual users; each user then evaluates the candidate for primality, and, weeks or months later, returns the result to a central server. At the time of writing, the project has access to around 30 000 machines in a typical month, with a throughput of approximately 80 teraflops.[3] Evidently the candidates are processed on a hugely parallel scale; however, each individual test requires a large number of sequential operations which cannot be parallelised, since each step of the Lucas-Lehmer test is contingent on the result of the previous step. In some cases, such as the Mersenne numbers, the problem size is logically unbounded[4] and therefore the granules of parallelism expand naturally until they become long-running on any given hardware. Likewise there are problems which are provably intractable and difficult to parallelise, which are likely to favour single-processor execution for the foreseeable future. For example, algorithms for analysis of abstract strategy games such as generalised chess, Go and Reversi are PSPACE-complete with respect to the size of the board [Fraenkel and Lichtenstein, 1981, Lichtenstein and Sipser, 1980, Iwata and Kasai, 1994]; they also have complex control flow and intricate data dependencies, making efficient parallelisation very challenging as 64- or 128-core machines become available [Soejima et al., 2010, Brockington, 1996]. Given human nature, we must assume that these problems are likely to remain of interest as our computational power grows. Therefore we must recognise that long-running unparallelised workloads are a permanent feature of the computational landscape no matter what conceptual or technical improvements might arise in the future.

---

[1] Great Internet Mersenne Prime Search: The Math, `http://www.mersenne.org/various/math.php`.

[2] In the late 1990s, RSA Security Inc. proposed three challenges, with large cash prizes, to anyone who could recover the plaintext of a series of DES-encrypted messages. Although DES encryption was not broken *per se*, its 56-bit key size was shown to be inadequate, as the first message was recovered ninety-six days after release, while the final challenge lasted just twenty-two hours. An archive of these contests is available online at `http://www.rsa.com/rsalabs/node.asp?id=2092`.

[3] Figures drawn from GIMPS PrimeNet, urlhttp://www.mersenne.org/primenet/.

[4] It is not known whether the number of Mersenne primes is infinite, although there are many conjectures that imply it [Gillies, 1964]. In either case the number of Mersenne candidates is obviously infinite,

In this chapter I address only a specific subset of large-duration energy problems and a certain type of solution to them, and I cannot make an argument for the logical permanence of either, but I have tried to relax the assumptions made as far as possible to make the contribution useful in practice, notwithstanding its mathematical abstraction. I particularly address the problem of energy-efficient scheduling; that is, given a particular task to be run and hardware on which to run it, how that hardware should be used in order to complete the task and optimise the energy usage according to some metric, under various constraints. Formal methods have often been applied to problems of energy-efficient scheduling. The simplest model takes the machine as a set of components which have an active state and some number of sleep states, each trading idle power for latency of transition to activity. An energy-efficient strategy is then a path through the set of states, which can be seen as a Markov decision process and solved exactly [Benini et al., 1998].

4.2.1 Dynamic voltage–frequency scaling

This chapter asks what part computer equipment plays in the demand for energy, and where we must focus to reduce consumption and improve energy efficiency in the future. Recent work has explored the continuous and relatively low-latency power scaling afforded by DVFS. DVFS exploits the observation that reducing the operating voltage of a chip reduces its operating power approximately quadratically, while necessitating only a linear decrease in operating frequency; therefore, the execution time also increases linearly, and a net gain in total execution energy can be made [Burd and Brodersen, 1995]. The voltage can be varied quite freely during execution and can be controlled by the processor itself, closing the loop; the machine thereby controls a crucial component of its own energy footprint. Typically a chip has some minimum and maximum voltage, and may support only a limited number of voltages between these extremes, but the relatively low time and energy penalties for transitioning between voltages makes for much greater flexibility in scheduling than could be achieved by discrete sleep states. Consequently there already exists a vast body of work on the subject, and even summary papers are quite numerous [Chen and Kuo, 2007, Tiwari et al., 1994]; there is only space here to discuss some particularly pertinent strands.

---

since they are simply $2^p - 1$ where p is prime. Other classes of efficiently-testable prime candidates, such as Proth numbers, would provide comparably difficult problems if the Mersenne primes were to be exhausted.

Some approaches require a probability density function (pdf) over the number of cycles required to complete the task; this pdf must be zero beyond some maximum value, known as the worst-case execution cycles (WCEC), or more precisely, the worst-case execution cycle count. (In this chapter, I do not draw a distinction between instructions and cycles; a multi-cycle instruction is simply considered as separate instructions.) One particularly powerful technique, requiring a concrete power model and an execution time pdf, can derive precise optimal operating voltages for various situations: energy minimisation for a given deadline, time minimisation under an energy constraint, and minimisation of a generalised penalty model proportional to energy and time together [Barnett, 2005]. However, while this work is mathematically powerful and quite general in its arguments—supporting, for example, any execution time pdf with finite support—its relevance is limited by the lack of empirical data. The schedules, while formally optimal, are not tested with concrete parameters, so it is unclear how much improvement they can provide over a simpler approach; this is a limitation I attempt to address in my work. In other work, a similar approach is used to schedule a set of sequential tasks to run in an energy-efficient way, assuming (as in this chapter) a particularly idealised processor [Xu et al., 2005]. Again this work considers DVFS exclusively as the method of energy efficiency. Each task is scheduled to execute at a certain speed (and therefore voltage); speeds are chosen such that all tasks are completed before the specified deadline, while the total energy is minimised. It is shown that such a schedule can be found quite efficiently; although formal bounds are not derived, the algorithm requires minimisation of N convex functions, where N is the number of tasks, and of course highly efficient algorithms for convex minimisation are well known [Boyd and Vandenberghe, 2004, pp. 457–513].

Other approaches to static scheduling are possible. For example, Qu and Potkonjak also establish a theoretic framework similar to that used here, and go on to describe multitask scheduling as a "utility maximization problem"; they show this problem to be NP-complete, and then describe powerful approximations to efficiently compute schedules arbitrarily close to the optimal [Qu and Potkonjak, 2000]. On the other hand, Irani et al. describe the Dynamic Speed Scaling with Sleep algorithm (DSS-S), which assumes a convex power function, availability of DVFS, and zero-energy transitions in and out of sleep; all assumptions in common with this chapter [Irani et al., 2007]. In contrast to my approach, DSS-S is not a scheduler *per se*, but rather a method for improving an established schedule by reducing its energy usage without affecting (soft) deadline misses or increasing deadline overshoot times. A static scheduling approach more similar to mine is given by Hong et al., who propose the treatment of voltage as "an optimization degree of freedom for . . . applications with real-time constraints" [Hong et al., 1999]. They then propose an algorithm to perform this constrained optimisation, which simulation suggests is a successful approach. However, they do not attempt to bound or restrict the computational complexity of the algorithm, so its practicality is not clear even as an offline solution. Work on soft deadlines can also support energy minimisation, although the characterisation of the problem as one of reward maximisation is not transferrable to the case where all deadlines must be met [Melhem et al., 2002].

Online (dynamic) scheduling methods are generally heuristic in their approach, but work also exists on more formal techniques in this space. For example, by applying techniques from differential calculus to the execution-cycle pdf, one can derive an optimal execution speed which reacts to the progress of the task so as to minimise the expected total energy of execution [Gruian, 2001]. Clearly this reactive approach can only improve on the static case, but does incur a computation penalty in practice. The techniques in this chapter also employ calculus but do not demand an execution time pdf, which is often very difficult to obtain. An alternative approach generalises the Markov state model to a continuous-time decision process, and can be shown empirically to outperform heuristic policies [Qiu and Pedram, 1999]. However, the computational overhead of solving such a process is considerable and again this consideration is not factored into the total energy expenditure.

4.2.2 Other formal methods

To illustrate the full gamut of formal methods for energy, it is worth observing that there are other benefits to energy efficiency, and other objectives than the minimisation of total energy.

For example, all hardware has a maximum operating temperature, and we might wish to schedule computation so as to minimise the peak temperature of the cores during computation. Assuming Fourier Law cooling and making some further (quite reasonable) assumptions about hardware characteristics, Bansal et al. give an efficient algorithm to calculate such a schedule, using the same polynomial power model assumed here [Bansal et al., 2004].

Early energy-efficiency work considered the reliability impact of the rapid temperature changes potentially caused by voltage scaling [Lee, 2000], but this area has itself been scaled back to a low intensity in recent years. Energy efficiency also has an impact on reliability, since low-power computing is more susceptible to transient faults from radiation events and so forth; modelling and managing this trade-off has also been the subject of some formal work [Zhu et al., 2004, Zhu, 2006].

4.3 Definition of terms

The following terms will have these specific meanings in this chapter:

- The *workload* (or WCEC) is the maximum number of instructions to be executed in order to complete the task.
- The *deadline* is a point in time by which all work must be completed. This is a hard deadline; in other words, any admissible schedule must guarantee that the deadline is met.
- The *cost function* describes the price per unit energy for all points in time until the deadline.
- The *power model* characterises the interaction of energy consumption with performance.

4.4 Problem statement

Having outlined the general class of problem that this chapter addresses, I state the problem precisely as follows: given the workload, deadline, and power and cost functions, determine the rate of computation for all points in time (from “time zero” until the deadline) which minimise the total cost of the energy expended, while still completing all work before the deadline.

My approach is as follows.

- In Section 4.5, I justify a simple and approximate mathematical model of computational power consumption in terms of utilisation.
- In Section 4.6, I demonstrate optimal strategies for this model in some plausible cost models, and show that these strategies could produce significant cost savings in reasonable cases.
- In Section 4.7, I outline mathematical tools for the solution of more general cost models.

4.5 Formal model

Some simplifying assumptions are always required in order to construct a precise mathematical model. For simplicity, this section presents the model directly; the assumptions are defended at length below.

Let W be the number of instructions to execute (the *workload*) and let T be the time available to complete the task (the *deadline*). By convention t denotes some time value in the range $[0, T]$.

Power consumption is assumed here to follow the polynomial dynamic model [Cho and Melhem, 2008], which is

$$P(t) = (\sigma + R^{\alpha}(t))I(t) \tag{4.1}$$

(Throughout this chapter $f^n(x)$ stands for $[f(x)]^n$ rather than repeated application of f, although f^{-1} denotes the inverse of f.)

In this model, σ is the static power, $R : [0, T] \to \mathbb{R}^+$ is a function indicating the work rate (in cycles per unit time) of the processor at time t, and α is a constant that controls the relationship between power and work rate, which is determined by the hardware configuration. Typical values would be $\alpha \approx 2$ or $\alpha \approx 3$ [Cho and Melhem, 2008]. $I : [0, T] \to \{0, 1\}$ is a simple indicator function denoting whether the system is powered on. The ordered pair (I, R) is referred to as a “strategy”, since it determines the behaviour of the system, and consequently how much power it requires at any given time.

The cost function $U : [0, T] \to \mathbb{R}^+$ denotes the unit cost of energy at time t, so that the total cost of the computation, C, is given by

$$C = \int_0^T P(t)U(t)\, dt \tag{4.2}$$

The formal problem statement is

Given U, α, σ, T and W, find the strategy (I, R) which minimises C.

The following constraints apply:

- Work cannot be undone and energy has some value; this is encapsulated in the restriction of the codomains of R and U to $\mathbb{R}^+$.
- The power function is strictly increasing. In the polynomial model, this simply implies that $\alpha > 1$.
- The power function is convex. This has a simple consequence for the polynomial model, since a function is convex over a region if its second derivative is non-negative throughout that region [Rudin, 1987, p. 61]. This implies that

  $$\frac{\partial^2 P}{\partial r^2} = \alpha(\alpha - 1)r^{\alpha-2} \geq 0 \tag{4.3}$$

  for $r \geq 0$, and therefore simply that $|\alpha| \geq 1$. This is therefore subsumed by the previous requirement that the function be strictly increasing, since this implies the stronger constraint that $\alpha > 1$.
- All the work is done by the deadline. I refer to this as the "Completion Constraint". Formally,

  $$\int_0^T R(t)I(t)\,dt = W \tag{4.4}$$

The cost function and the strategy together determine the total energy of the computation.

4.5.1 Assumptions and justification

The power model describes a system with the following characteristics: when turned on, the system requires zero or more units of "static power" and some additional "dynamic power" which increases monotonically with performance; when turned off, it requires no power. The assumption that the power function is strictly increasing ($\partial P/\partial r > 0$) is fundamental and it is hard to imagine any system in which running faster would reduce the power requirement; certainly such situations are esoteric enough that I do not consider it necessary to optimise for them in the general case. The assumption of convexity ($\partial P^2/\partial^2 r \geq 0$) is more disputable; in concrete terms, I assume that each additional cycle requested from the processor (per unit time) requires more energy (per unit time) than the last. Nevertheless, the assumption is likely to hold, since systems generally scale super-linearly with utilisation [Barroso and Hölzle, 2007], and it is widely adopted [Lehoczky et al., 1989, Okuma et al., 1999, Gutnik and Chandrakasan, 1997, Lorch and Smith, 2001]. Generally this chapter assumes a polynomial model of dynamic power [Cho and Melhem, 2008]. With suitable parameters, this model is a reasonable approximation to most real hardware and crucially, it is sufficiently simple that it permits the derivation of strong theoretical results. Additionally, the convexity constraint follows from the monotonicity constraint, rendering the former assumption less debatable.

I consider only two contributors to the total system power: a dynamic term varying with utilisation (measured in cycles per unit time), and a constant term. The dynamic term models the power of the CPU, and potentially any further components which scale in line with the CPU; these may include, for example, main memory or the network interface card. The constant term encompasses the power of the remaining components, and the static power of the CPU itself. The dynamic behaviour which is not proportional to CPU utilisation, such as the power consumed by the network card in peak-traffic periods, is assumed to be small compared to the dynamic range of the CPU.

I assume an idealised CPU, in the following sense: the voltage and clock frequency can be changed instantaneously and at no energy cost, and can take on any value from zero to the maximum frequency supported by the CPU. In practice, ignoring switching energy makes little difference to the choice of frequencies, although of course it underestimates the exact energy totals [Swaminathan and Chakrabarty, 2001]. As for the energy penalty, energy cost changes are relatively infrequent and, by a result demonstrated explicitly below, computation speed changes are similarly infrequent, so it is reasonable to discard these for a computation of the assumed length. Clearly the assumption that frequency can be changed instantaneously does not hold, since in practice there is some overhead associated with adjusting the clock frequency; however, simulation has shown that the energy differential for more involved models is only about 7% [Hong et al., 1998]. The assumption that voltage and clock frequency may be varied continuously within the finite bounds is essentially true for some processors. On the other hand, for processors which support only a finite number of pre-determined frequencies, the optimal solution is simply to round to the nearest available frequency [Ishihara and Yasuura, 1998]. Empirical evidence suggests that the loss of flexibility in such cases has a small impact on overall efficiency [Lee and Sakurai, 2000].

I assume that the workload is known in advance. In reality the workload may be difficult to provide and is of course undecidable in general. In cases where exact workload predictions cannot be made, it may be possible to find an upper bound. The resultant schedule is likely to be over-eager and thus somewhat energy-suboptimal, but still no worse than the naïve approach. However, in cases where the upper bound is loose, a dynamic run-time approach would probably perform better. Requiring this input is not as unreasonable as it might once have been: there is now a large body of work on proving termination for programs as large as 100 000 lines of code, despite the impossibility of the general case [Cook et al., 2006]. The problem of calculating execution time bounds has also been extensively studied, with considerable practical success [Bedin França et al., 2011, Souyris et al., 2005, Heckmann and Ferdinand, 2004]. If no upper bound can be determined, it is impossible for any schedule to guarantee a hard deadline and so this case is not considered. On the other hand, there are some situations in which it is reasonable to expect an exact workload. For example, in decoding an encrypted or compressed stream, it is possible to determine at encode-time exactly what must be done to decode the resultant stream, and this information could then be embedded into the stream itself. Predictable workloads have already been shown to have power-efficiency benefits in the realm of digital signal processing [Chandrakasan et al., 1996]. In any case it is clear that an algorithm for this problem requires an upper bound on the workload, and that for offline algorithms a tighter bound will produce a superior schedule.

I further assume that the number of instructions executed is independent of the rate at which they are executed. In practice this is not completely accurate, due to complicating effects such as caching, pipelining, bus speed mismatches and so on [Seth et al., 2003]. However, empirical evidence suggests that, for a reasonable cache size, the effect is very small: typically less than 2% [Melhem et al., 2004, p. 4]. Therefore a task is simply considered to take a certain number of instructions to resolve, and every instruction requires a fixed amount of time. Therefore the worst-case execution time (WCET) is simply the workload divided by the speed of the processor, in cycles per unit time.

4.6 Solutions for specific cost models

In this section, I propose some forms of cost model that I believe are likely to correspond to realistic cases. As ever, a trade-off exists between the strength of assumptions and the viability of analytic solutions. This section provides exact analytic solutions, and therefore I have endeavoured to make the models as general as possible within that constraint; variables are introduced wherever possible to provide flexibility.

4.6.1 Constant cost

The simplest cost function is $U(t) = u$ for some constant u. This is the degenerate case in which minimising energy and energy cost are identical objectives. This result has been derived in previous work, under the name "energy-efficient frequency", so I verify that it is recreated within my framework [Zhu et al., 2004, p. 3].

Since the cost function is constant, I assume without loss of generality that the system is powered on for one contiguous block of time. Consequently, any strategy can be reduced to one in which the system begins powered off, remains off for some period of time S (the "slack time"), is then powered on, and remains on for the remaining $T - S$ time. (This also minimises the number of state changes, which helps preserve the assumption that these are not significant contributors to execution time and energy.) The slack time, which embodies the discovery that it may be economical to do nothing for a certain length of time in a large computation, is named in homage to Gottbrath's introduction of another form of slack time [Gottbrath et al., 1999]. In that formulation, the computational power of new hardware increases so rapidly that a long-running computation may finish earlier if we delay its start until better hardware becomes available. That result, of course, is predicated upon Moore's Law, whereas here I only make the less startling assumption that static power is a non-trivial component of total power. Nevertheless, the ideas enjoy numerous similarities.

Due to the convexity of the power function (the assumption that $\alpha > 1$), a strategy in which computation occurs at a constant rate for the entire work phase is at least as good as any other. This can be seen intuitively by the following argument: consider any strategy which deviated from a constant rate; now reduce the work rate at its highest point and increase it at its lowest point by the same amount, such that the total number of instructions executed is the same. Now the reduction in power at the high point is clearly greater than the increase in power at the low point, since the power function is convex with respect to work rate. Therefore the overall energy is reduced. Consequently, only a strategy which has no such "high points" can be optimal, which is exactly the constant functions. A more precise argument can be given; here, I formalise the argument sketched by Gutnik and generalise it to the continuous case [Gutnik and Chandrakasan, 1997]. This argument applies to any convex power function, including of course the dynamic polynomial model I generally assume. Consider Jensen's Inequality [Gradshteyn et al., 1980, pp. 1132–1133], which states the following.

Theorem 4.6.1. (Jensen's Inequality) *Consider an interval* $[a, b]$*. Let* p *be any function on the interval such that* $p(x) \geq 0$ *and* $p \not\equiv 0$*; let* f *be any function and define constants* α *and* β*, chosen such that* $\forall x \in [\alpha, \beta] . \alpha \leq f(x) \leq \beta$*; let* ϕ *be any convex function. Then*[5]

$$\phi\left(\frac{\int_a^b f(x)p(x)dx}{\int_a^b p(x)dx}\right) \leq \frac{\int_a^b \phi(f(x))p(x)dx}{\int_a^b p(x)dx} \tag{4.5}$$

Given Equation 4.5, the result follows quite straightforwardly.

Proof. Set $a = 0$, $b = 1$, $p(x) = 1$ in Jensen's Inequality, and substitute $x \mapsto t$, $f \mapsto R$ and $\phi \mapsto P$. By assumption, P is convex, as required; furthermore, since it is convex everywhere that it is defined (which is to say, $[0, T]$), we can take $\alpha = 0$ and $\beta = \infty$, which certainly bound any $R(t)$ we might see. Then Jensen's Inequality reduces to the following compact form:

$$P\left(\int_0^1 R(t)dt\right) \leq \int_0^1 P(R(t))dt \tag{4.6}$$

Assume without loss of generality that the window of computation is $0 \leq t \leq 1$. Then one can recognise the term inside the left-hand brackets as the workload, W, in Equation 4.4. Likewise the right-hand side is the total energy consumed, E. Hence we have the concise result that $E \geq P(W)$. Now consider a constant rate of work, which by the Completion Constraint requires that $R(t) = W$. In this case,

$$E = \int_0^1 P(W)dt = P(W) \tag{4.7}$$

This is the lower bound of the inequality above. Hence, as required, the lower bound for E is achieved when R is a constant function. □

Note that we have not proved the converse and therefore do not exclude the possibility that an alternative strategy might be equally efficient, but it cannot be superior. In fact, this result is not specific to power; any resource which is a convex function of utilisation is minimised by constant consumption, and similar results have been established in other work [Melhem et al., 2002, p. 133].

---

[5] There are multiple formulations of Jensen's Inequality, with measure theory's presentation being perhaps the most convenient; however, to avoid the dependency on the fairly specialised notation of that field, I give the inequality in the slightly more cumbersome form used for real analysis and then eliminate the unnecessary terms directly.

Given the assumption of a contiguous slack time and a constant rate of computation during the non-slack time, the formal strategy (I, R) must take the following form:

$$I(t) = \begin{cases} 0 & \text{if } t \leq S \\ 1 & \text{if } t > S \end{cases} \tag{4.8}$$

and $R(t) = r$ for some constant r. By the Completion Constraint,

$$\int_0^T R(t)I(t)\,dt = r(T - S) = W \tag{4.9}$$

so $r = W/(T - S)$, and the total cost C is

$$\begin{aligned} C &= \int_0^T P(t)U(t)\,dt && (4.10) \\ &= u(T-S)\left(\sigma + \left(\frac{W}{T-S}\right)^{\alpha}\right) && (4.11) \end{aligned}$$

To find the optimal slack time, set $\partial C/\partial S = 0$, so

$$\sigma + \left(\frac{W}{T-S}\right)^{\alpha} = (T-S)(\alpha W^{\alpha}(T-S)^{-\alpha-1}) \tag{4.12}$$

which has the unique solution

$$S = T - W\sqrt[\alpha]{\frac{\alpha-1}{\sigma}} \tag{4.13}$$

As expected, the solution is independent of u, which is just a scaling factor of the cost. It may transpire that this value of S is outside the valid range, in other words that $S < 0$, in which case simply take $S = 0$. Clearly it cannot be that $S > T$.

Substituting this optimal S gives the optimal rate of calculation:

$$r_{opt} = \sqrt[\alpha]{\frac{\sigma}{\alpha-1}} \tag{4.14}$$

This agrees with the established result for energy-efficient frequencies.

The minimal cost can also be given explicitly:

$$C_{opt} = \frac{\alpha\sigma W u}{\alpha-1}\sqrt[\alpha]{\frac{\alpha-1}{\sigma}} \tag{4.15}$$

Next, I determine how much this optimal solution stands to gain over the naïve solution: that is, computing at maximum speed until the work is complete. To answer this, define R_{max} to be the maximum work rate of the processor; this is both the speed at which the naïve strategy will compute (until the task is complete), and an upper bound on the admissible values of R from any alternative strategy. Clearly the naïve strategy incurs cost C_{max}, where

$$C_{max} = (\sigma + R_{max}^{\alpha})\frac{Wu}{R_{max}} \tag{4.16}$$

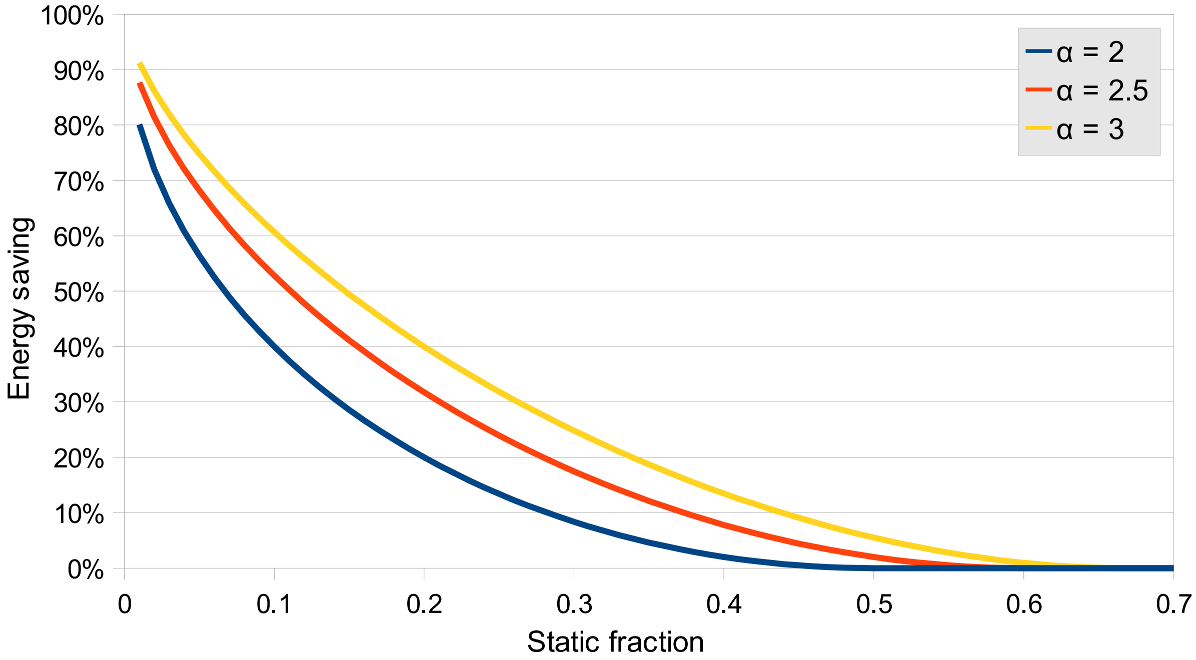

Figure 4.1: Potential cost saving λ, as a percentage, against static fraction μ, for some realistic α values.

Let λ be the fraction of the cost required by the naïve strategy that could be saved by the optimised strategy, so that

$$\lambda = \frac{C_{max} - C_{opt}}{C_{max}} = 1 - \frac{\alpha\sigma R_{max}}{(\alpha - 1)(\sigma + R_{max}^{\alpha})} \sqrt[\alpha]{\frac{\alpha - 1}{\sigma}} \tag{4.17}$$

Of course, the exact value of the cost saving depends on the values of the other variables in the power model and so forth, but usefully, this definition is independent of W, u and T. This reduces the dimensionality of the space over which one most compare the two strategies. One dimension that must be considered is the ratio of static to dynamic power, so define μ to be the fraction of total power consumed by static power at peak performance:

$$\mu = \frac{\sigma}{\sigma + R_{max}^{\alpha}} \tag{4.18}$$

This can then be substituted into Equation 4.17 to give

$$\lambda = 1 - \alpha \sqrt[\alpha]{1 - \mu} \left(\frac{\mu}{\alpha - 1} \right)^{1 - 1/\alpha} \tag{4.19}$$

and this eliminates R_{max} so that λ remains dependent on only two variables. Therefore one can reasonably sketch out the behaviour for some likely parameter values.

Note that this definition is only correct if the optimised strategy respects the maximum rate of computation, in other words $\forall t. R(t) \leq R_{max}$. This implies that we must have $W/(T - S) \leq R_{max}$, which reduces to the simple constraint that $\mu \leq 1 - 1/\alpha$. For $\mu > 1 - 1/\alpha$, the optimised strategy converges to the naïve strategy, so $\lambda = 0$.

Figure 4.1 shows the aforementioned sketch of λ for varying values of μ, with some plausible values of α. Several interesting conclusions can be drawn from this sketch that are not immediately obvious from Equation 4.19. First and most obviously, the optimised solution is always able to save some energy; and, when μ is small and the dynamic energy dominates, the optimised strategy is able to achieve much higher savings. Whether these solutions are actually possible also depends on the value of T, which is not considered here; the specific case $\mu = 0$ would favour computing infinitely slowly and so is not shown. The sensitivity to α is less obvious from Equation 4.19, but clear from the figure; for example, when the maximum dynamic power is fixed at twice the static power ($\mu = 1/3$), λ is 6%, 14% and 21% for $\alpha = 2$, 2.5, and 3 respectively, which is a significant discrepancy. The effect is predictable, since the steeper the power curve the more wasteful maximum-speed computation becomes; however, the magnitude of it is perhaps surprising. Furthermore, for larger α values, the static fraction has to be considerably larger before optimisation becomes useless; $\mu =$ 46%, 56% and 62% respectively. This suggests that the optimisation is markedly more useful when deployed in systems with α values towards the higher end of typical estimates; again this is qualitatively obvious but the magnitude of the effect is somewhat unexpected.

4.6.2 General discrete variable cost

Consider now a more general case, where U varies in any number of discrete regular time steps; in other words, for some integer m:

$$U(t) = \begin{cases} u_0 & \text{if } 0 \leq t < T/m \\ \vdots & \\ u_{m-1} & \text{if } (m-1)T/m \leq t < T \end{cases} \tag{4.20}$$

where $u_0, \ldots, u_{m-1}$ are positive real constants. Assume that $u_i \leq u_{i+1}$; this assumption can be made without loss of generality since one can always sort the time steps into cost order.

Since the cost is increasing, any optimal strategy must involve working from the start for some period of time and then switching off for the rest of the time. (In the general case, this corresponds to working in only the cheapest time slots.) This strategy is parametrised by the length of this working period, which is assumed to be some multiple of s; in other words, computation spans a whole number of time steps. This method can be extended to remove this assumption but this simpler case illustrates the technique more clearly; alternatively one can slice the steps into small substeps to provide an arbitrarily good approximation.

Under these conclusions, "Strategy n" can be defined as (I_n, R_n) with

$$I_n(t) = \begin{cases} 1 & \text{if } t \leq Tn/m \\ 0 & \text{if } t > Tn/m \end{cases} \tag{4.21}$$

and

$$R_n(t) = \begin{cases} r_0 & \text{if } 0 \leq t < T/m \\ \vdots & \vdots \\ r_{n-1} & \text{if } T(n-1)/m \leq t < Tn/m \\ 0 & \text{otherwise} \end{cases} \tag{4.22}$$

for constants $r_0, \ldots, r_{n-1}$, and $0 < n \leq m$. The Completion Constraint requires that

$$\sum_{i=0}^{n-1} \frac{r_i T}{m} = W \tag{4.23}$$

or, extracting r_0, that

$$r_0 = \frac{Wm}{T} - \sum_{i=1}^{n-1} r_i \tag{4.24}$$

The total energy cost is

$$C_n = \frac{T}{m} \sum_{i=0}^{n-1} (\sigma + r_i^\alpha) u_i \tag{4.25}$$

So to minimise C_n (for fixed n), the following system of equations must be solved:

$$\frac{\partial C_n}{\partial r_0} = \cdots = \frac{\partial C_n}{\partial r_{n-1}} = 0 \tag{4.26}$$

The variables $r_1, \ldots, r_n$ can be considered mutually independent, with r_0 constrained by Equation (4.24); therefore

$$\frac{\partial r_i}{\partial r_j} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } 0 \neq i \neq j \neq 0 \\ -1 & \text{otherwise} \end{cases} \tag{4.27}$$

So, for any $j \neq 0$,

$$\frac{\partial C_n}{\partial r_j} = \frac{T}{m} (\alpha r_j^{\alpha-1} u_j - \alpha r_0^{\alpha-1} u_0) = 0 \tag{4.28}$$

which is solved by $r_j = r_0 \zeta_j$ where

$$\zeta_j = \sqrt[\alpha-1]{\frac{u_0}{u_j}} \tag{4.29}$$

This gives each r_j in terms of r_0, so to complete the solution, return to the Completion Constraint, which is now

$$\sum_{i=0}^{n-1} \frac{T r_i}{m} = \frac{T r_0}{m} \sum_{i=0}^{n-1} \zeta_i = W \tag{4.30}$$

and so, by normalisation,

$$r_{0[opt]} = \frac{Wm}{T \sum_{i=0}^{n-1} \zeta_i} \tag{4.31}$$

and the solution is complete. This describes the strategy for a given n; the minimal cost is therefore

$$C_{n[opt]} = \frac{T}{m} \left(\left(r_{0[opt]} \sqrt[\alpha-1]{u_0}\right)^\alpha \sum_{i=0}^{n-1} \sqrt[1-\alpha]{u_i} + \sigma \sum_{i=0}^{n-1} u_i \right) \tag{4.32}$$

The optimal value of n can be found efficiently by any univariate maximisation technique. However, for larger values of m, a more direct mechanism might well be desirable; in Section 4.6.4, I demonstrate that for specific cost functions, more efficient computational techniques for finding n can be applied.

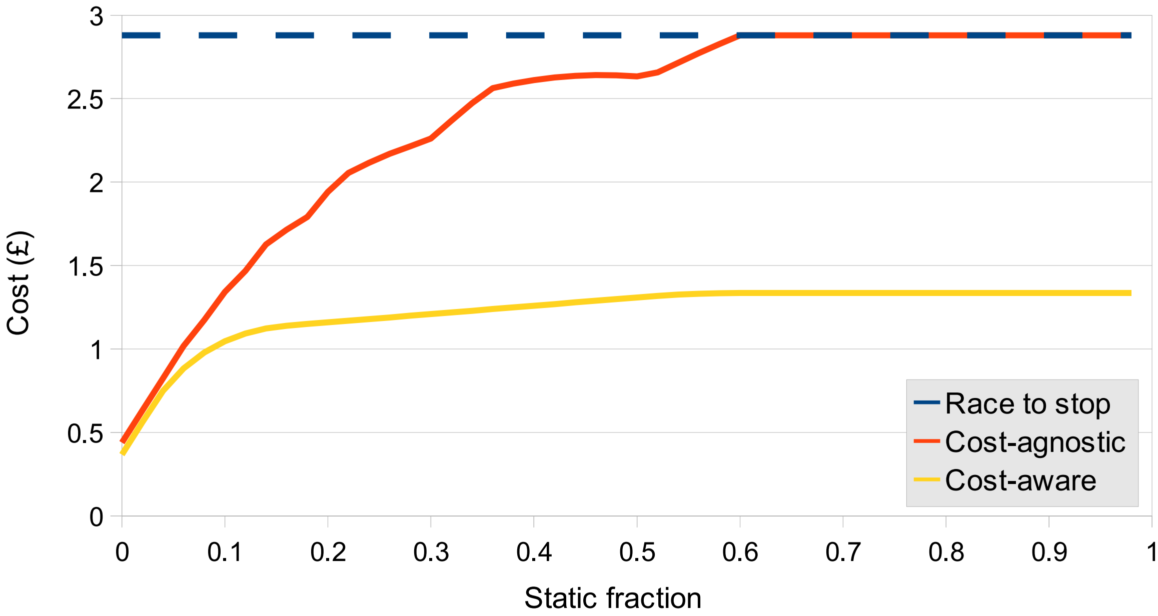

Figure 4.2: Cost incurred to perform a long-running computation, for various ratios of static to dynamic power.

4.6.3 Real-world example

To illustrate the advantages of cost optimisation, this section describes a plausible problem case and shows the savings that cost optimisation can achieve. I consider the following situation:

- The system has one week to complete some computational task, starting at midnight on Monday.
- The workload is 48 hours' work at the maximum rate.
- Electricity costs are based on the standard plan offered by power provider E.ON Energy[6] in the author's postcode as of November 2011. Therefore off-peak electricity is available from 11.30pm to 7.30am at 6.1845p per kilowatt-hour (kWh); on-peak electricity is available at all other times, priced 16.905p per kWh.
- When computing at the maximum rate, the system requires 450 W of power. This is divided between the static and dynamic components according to the *static fraction* parameter.
- The machine follows the dynamic polynomial power model, with $\alpha = 2.5$.

Note that the value of R_{max} follows from the static power, maximum dynamic power, and value of α, so is not given explicitly.

---

[6] `http://www.eonenergy.com`

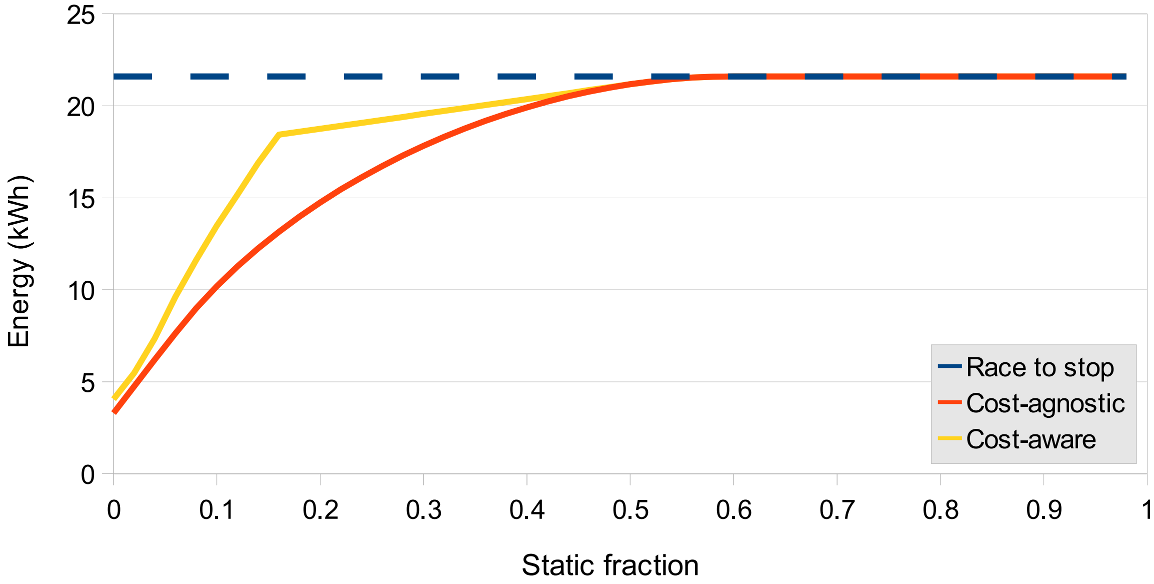

Figure 4.3: Energy required to perform a long-running computation, for various ratios of static to dynamic power.

Figure 4.2 shows the cost of completing this computation for varying values of the static fraction. The simple race-to-stop strategy, which simply runs at maximum speed until completion, always consumes 450 W and is therefore independent of the static fraction; it requires 21.6 kWh at a cost of £2.88. The figure also shows the behaviour of the cost-agnostic algorithm, which optimises for energy but not energy cost, and the full algorithm which minimises energy cost. (In both cases, the R_{max} limit is imposed, so the comparison is a fair one.) At low static powers, the two algorithms behave similarly because they both compute throughout the week, although the cost-aware algorithm is still slightly more efficient since it moves more cycles into off-peak pricing periods. As the static component increases, the cost-agnostic schedule runs early time segments at R_{max} and then turns the system off, converging on the race-to-stop schedule. When the static power accounts for more than about 60% of the total, all three schedulers level off, although the cost-aware model continues to make better use of off-peak energy, converging to £1.34.

Figure 4.3 shows the raw energy usage for the three schedulers. As expected, the cost-aware scheduler consumes more total energy than the cost-agnostic scheduler; the superior price is achieved by performing more total work but shifting it to off-peak times. This might seem undesirable but, since the grid has almost no storage capacity and produces a reasonably constant output of electricity throughout the day, this would actually be a net win for both the provider and the consumer; this is, after all, exactly the behaviour that the off-peak period is designed to incentivise. Both figures display a “phase change” in cost-aware behaviour at around 16% static power; this is the first point at which it becomes cost-efficient to perform any computation at on-peak prices.

Of course this example includes many arbitrary constants, but clearly demonstrates that there are significant cost savings to be made in a practical case; for a realistic system with static power of perhaps 40%, cost-aware optimisation is 56% cheaper than the naïve race to stop, and 52% cheaper than even an energy-optimal schedule. While some of this margin would undoubtedly be lost in the translation to all the complexities and overheads of a real system, I believe it is large enough that a significant reduction in the real cost would be achievable.

4.6.4 Exponential cost

In the long term, one might expect the cost of energy to behave exponentially, either increasing or decreasing depending on one's optimism about the progress of technology. In this section I specifically consider the case where cost increases exponentially at each time step, in order to preserve the assumption that $u_i \leq u_{i+1}$; the solution can easily be adapted for the case of exponential decrease. Assuming arbitrary units of cost, define

$$u_i = e^{Ti/m} = z^i \tag{4.33}$$

where $z = e^{T/m}$. Hence $\zeta_i = k^i$ where $k = \sqrt[1-\alpha]{z}$. Then, substituting into Equation 4.31 gives a simple geometric sum so

$$r_0 = \frac{Wm(1-k)}{T(1-k^n)} \tag{4.34}$$

Likewise Equation 4.32 collapses into a compact form:

$$C_n = \frac{T}{m}\left(\frac{1-z^n}{1-z}\sigma + \left(\frac{Wm}{T}\right)^{\alpha}\left(\frac{1-k}{1-k^n}\right)^{\alpha-1}\right) \tag{4.35}$$

Expanding terms dependent on n, this can be written in explicit and particularly concise form:

$$C_n = A(1-z^n) + B(1-k^n)^{1-\alpha} \tag{4.36}$$

where the constants are independent of n, *viz.*

$$A = \frac{T\sigma}{m(1-z)} \tag{4.37}$$

and

$$B = W^{\alpha}\left(\frac{m(1-k)}{T}\right)^{\alpha-1} \tag{4.38}$$

This equation makes it highly efficient to compute the optimal n for even large m by any standard optimisation technique.

4.6.5 Generalised time intervals

A simple approach to optimising for the fully general continuous cost function would be to replace it with an approximate discretised version of the same function. In this case, the obligation to separate time into fixed quanta of length T/m might be overly restrictive, and it would be preferable to sample the function more finely in regions with larger derivative. Therefore I also present the solution for arbitrary-length time segments. Define $0 = t_0 < t_1 < \ldots < t_m = T$, and then take

$$U(t) = \begin{cases} u_0 & \text{if } t_0 \le t < t_1 \\ \vdots & \\ u_{m-1} & \text{if } t_{m-1} \le t < t_m \end{cases} \tag{4.39}$$

The completion constraint is

$$\sum_{i=0}^{n-1} r_i (t_{i+1} - t_i) = W \tag{4.40}$$

and the cost

$$C_n = \sum_{i=0}^{n-1} (\sigma + r_i^\alpha)(t_{i+1} - t_i) u_i \tag{4.41}$$

so

$$\frac{\partial r_i}{\partial r_j} = \begin{cases} 1 & \text{if } i = j = 0 \\ -(t_{i+1} - t_i)/t_1 & \text{if } i = 0, j \neq 0 \\ 0 & \text{if } i \neq 0 \end{cases} \tag{4.42}$$

and, by the earlier method, we have $r_i = r_0 \zeta_i$ since the interval terms cancel. Normalising completes the solution, giving

$$r_0 = W \left(\sum_{i=0}^{n-1} (t_{i+1} - t_i) \zeta_i \right)^{-1} \tag{4.43}$$

The minimum cost is, therefore

$$C_n = \sigma \sum_{i=0}^{n-1} u_i (t_{i+1} - t_i) + (r_0 \sqrt[\alpha-1]{u_0})^\alpha \sum_{i=0}^{n-1} (t_{i+1} - t_i) \sqrt[1-\alpha]{u_i} \tag{4.44}$$

It can be readily verified that these solutions coincide with the fixed case when $t_i = Ti/m$.

To demonstrate the utility of this adaptive quantisation, consider the continuous cost function

$$U(t) = k + (1 - k) \sin\left(\frac{\pi t}{2}\right) \tag{4.45}$$

where k is a small constant to prevent zero-cost energy at $t = 0$ (I take $k = 0.01$). Take $T = 1$, so that $U(t)$ is monotonically increasing over the relevant range, $0 \le t \le 1$. Rather than solve for this function directly, let us take a discrete approximation. I discretise in two different ways to demonstrate the contrast. One method, regular quantisation, simply divides the function into n regular steps of width $1/s$. The other method, fitted quantisation, divides the function into s variable-width steps such that the value of $U(t)$ increases by the same amount over each; in this case, the ith step runs from

$$\frac{2}{\pi} \arcsin\left(\frac{i}{s}\right) \le t < \frac{2}{\pi} \arcsin\left(\frac{i+1}{s}\right) \tag{4.46}$$

Figure 4.4 compares the two methods of quantisation against the continuous function for $s = 10$; note how the fitted quantisation uses smaller steps in the steeper part of the sine

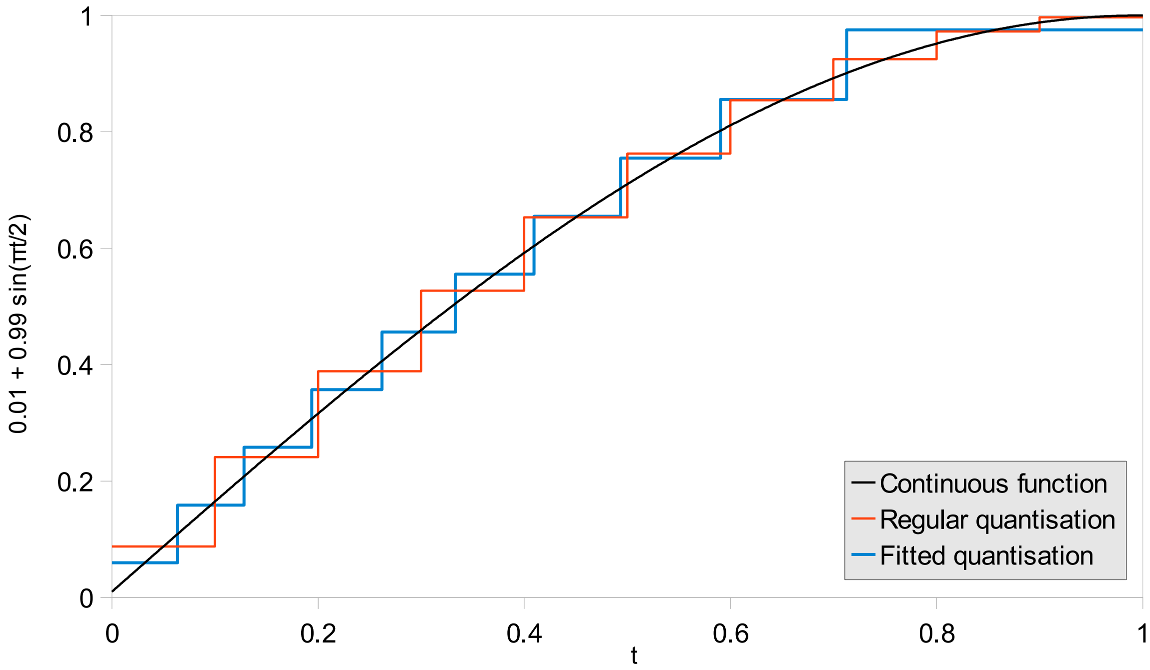

Figure 4.4: A continuous cost function and two different methods of discretising it into ten steps.

curve. In both methods the effective value of U is sampled at the midpoint of the interval, so the continuous curve intersects each step of the discrete functions at the midpoint of each step. Note that, although the fitted quantisation is intuitively a better fit, there is no reason to believe that it is the optimal approximation for our purposes; this is intended only as a demonstration of the advantages of variable width quantisation for this purpose. To demonstrate this advantage, I optimise for the two quantisations and measure the cost incurred, assuming the cost function of Equation 4.45.

For further illustration, Figure 4.5 shows the proposed schedules for both quantisations in the case where $s = 25$, with $\alpha = 2.5$. The figure also shows the optimal solution. (This was in fact computed numerically by taking $s = 10\,000$.) The system is assumed to be powered down wherever the rate is zero, and in particular the rate is zero for the unshown portion of the graph ($t > 0.3$). Note that the adaptive solution computes for ten steps to the regular solution's seven, but still finishes earlier since the steps are smaller. During the active period, it is not immediately obvious that the fitted quantisation is a better approximation to the optimal solution, but the total costs bear it out: regular quantisation produces a solution 6.3% more expensive than the optimal, while the fitted quantisation incurs only a 3.6% cost overhead.

Finally, Figure 4.6 shows the overhead of the two approaches, compared to the optimal solution, over a range of s values and for $\alpha = 2$, 2.5, and 3. Evidently the fitted quantisations are measurably better in every case, excepting a few small anomalies for the coarsest quantisations (which illustrates the point that this quantisation is not necessarily optimal for this purpose). For s values of 100 or more, the overhead becomes negligible; for $s \geq 180$ both methods are less than 1% inefficient. This demonstrates that, as an offline optimisation technique, there is no great benefit to the added complexity of the adaptive method, justifying the decision to present the simpler case explicitly. Nevertheless, this

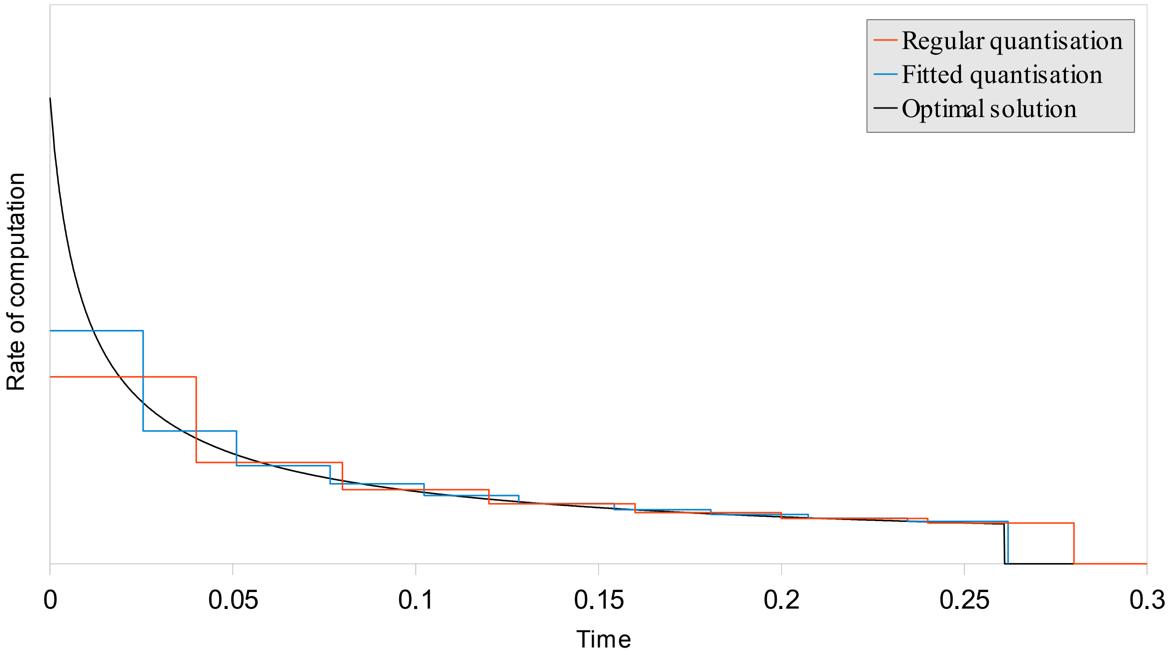

Figure 4.5: Schedules for two different quantised approximations to the same continuous cost function. Arbitrary units of rate.

also demonstrates that flexible quantisation is justified if the overhead of computing the schedule must be kept low; for example, as illustrated in later chapters, it might sometimes be desirable to recompute a schedule online in the light of new information.

4.6.6 Summary of cost model-specific solutions

In this section, I have presented precise algebraic solutions to the fixed cost model and to any model described by a series of discrete steps, and demonstrated that the latter can be an effective approximation to more complex continuous functions. I have also shown that the algebra can be pushed further in particularly amenable cases, such as $u_i = z^i$, to make the parameter search more efficient. However, while it can be approximated, the more general problem of an arbitrary continuous cost function remains unsolved. This I address in the next section.

4.7 General methods

In this section I present some techniques for optimising for continuous cost functions in the general case. While it is not possible to complete the solution in the general case, the methods developed here can be applied to particular continous cases and thereby simplify the process of finding a solution. I also describe a method for finding solutions subject to an additional constraint: constant spending.

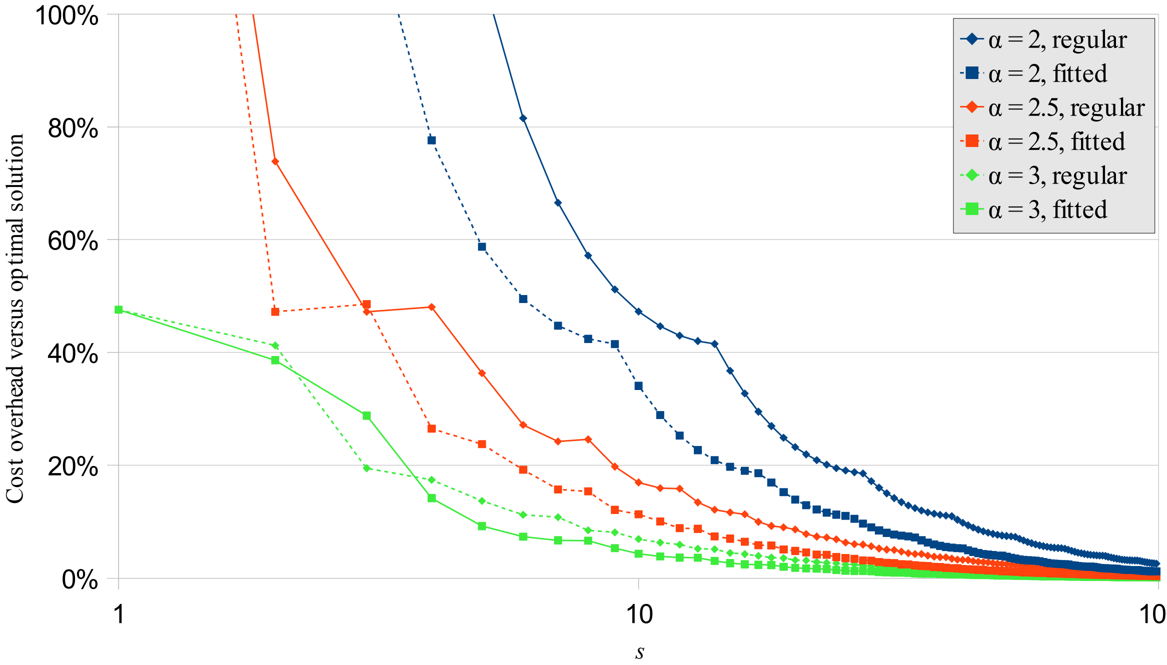

Figure 4.6: Comparison of cost overheads incurred by regular and fitted quantisations for several values of α. Horizontal axis is on a logarithmic scale.

4.7.1 Monotonic cost functions

It is not possible to provide a complete analytical solution for an arbitrary cost function, but in this section I develop some techniques that may be useful in minimising cost for general monotonic cost functions. By analogy with the earlier assumption that $u_i \leq u_{i+1}$, assume that U is a monotonically increasing continuous function. A monotonically *decreasing* function V can be converted into the appropriate form by taking $U(t) = T - V(t)$, effectively reversing the arrow of time; since the algorithm is entirely offline, there is no issue with causality. For cost functions that are not monotonic, one can apply the approximation technique of Section 4.6.5, or more powerful mathematics are required, as discussed in Section 4.8.

By the earlier argument, the following indicator is as good as any other for suitable S:

$$I(t) = \begin{cases} 0 & \text{if } t \leq S \\ 1 & \text{if } t > S \end{cases} \tag{4.47}$$

so that the Completion Constraint becomes simply

$$\int_S^T R(t)\,dt = W \tag{4.48}$$

Now the power function P is parametric in S; the longer the CPU idles at the beginning, the faster it must run once it starts. For convenience, define $f(S,t) = U(t)P_S(t)$, so one can simply state

$$C = \int_S^T f(S,t)\,dt \tag{4.49}$$

To find the optimal S, set $\partial C/\partial S = 0$ as usual. Under the reasonable assumption that f is continuous over $[S, T]$, one may differentiate under the integral to give

$$\begin{aligned}
\frac{\partial C}{\partial S} &= \frac{\partial}{\partial S}\int_S^T f(S,t)\,dt \\
&= \int_S^T \frac{\partial}{\partial S} f(S,t)\,dt - f(S,S) \\
&= 0
\end{aligned}$$

using the standard result [Gradshteyn et al., 1980, p. 23].

Rearranging, and expanding f, gives

$$P_S(S)U(S) = \int_S^T U(t)\frac{\partial P_S}{\partial S}\,dt \tag{4.50}$$

since U is independent of S.

Since only the dynamic power depends on the slack time, this can be further rewritten in terms of R:

$$[\sigma + R^\alpha(S)]\,U(S) = \alpha\int_S^T U(t)R^{\alpha-1}(t)\frac{\partial R}{\partial S}\,dt \tag{4.51}$$

The latter seems more readily soluble, since the Completion Constraint also involves R. This gives an approach to finding R for any general U, if one can solve the integral in Equation 4.51.

To illustrate the technique, let us briefly return to the case of the constant cost function, $U(t) = u$. As in Section 4.6.1, this implies $R(t) = W/(T - S)$, and consequently the derivative

$$\frac{\partial R}{\partial S} = \frac{W}{(T-S)^2} \tag{4.52}$$

One can now apply Equation 4.51, to give

$$\left(\sigma + \left(\frac{W}{T-S}\right)^\alpha\right)u = \alpha\int_S^T u\left(\frac{W}{T-S}\right)^{\alpha-1}\frac{W}{(T-S)^2}dt \tag{4.53}$$

Despite its apparent inelegance, this equation quickly cancels down to give

$$\sigma + \left(\frac{W}{T-S}\right)^\alpha = \alpha\left(\frac{W}{T-S}\right)^\alpha \tag{4.54}$$

and a simple rearrangement produces the now-familiar form of Equation 4.13, completing the solution. Even in this simplest of examples, it is apparent that the volume of algebra and risk of error can be reduced by applying the general method embodied in Equation 4.51.

4.7.2 Constant spending

A particular class of solution arises if the cost per unit time is required to be constant over the non-slack region of computation. Here let P be any power function, so that

$P(r)$ is the power consumption of computing at rate r. As before, P must be strictly monotonically increasing and hence invertible; let P^{-1} denote this inverse. Let V be the rate of spending, so that $\forall t. U(t)P(t) = V$ with V independent of t. To find V, apply the Completion Constraint:

$$\int_S^T P^{-1}\left(\frac{V}{U(t)}\right) dt = W \tag{4.55}$$

Solving this integral for the relevant U gives an equation for V in terms of S. The total cost is then

$$C = \int_S^T P(t)U(t)\,dt = (T-S)V \tag{4.56}$$

and so

$$\frac{\partial C}{\partial S} = (T-S)\frac{\partial V}{\partial S} - V \tag{4.57}$$

So setting $\partial C/\partial S = 0$, the optimal slack time can be derived by the neat equation

$$V = (T-S)\frac{\partial V}{\partial S} \tag{4.58}$$

Solving this may be hard in practice, depending on the tractability of Equation 4.55. Of course numeric approximations are possible, although in the general case the energy cost of the computation might exceed the energy saved by prudent scheduling.

As an example, assume that $P(r) = \sigma + r^\alpha$ as before, so that $P^{-1}(p) = \sqrt[\alpha]{p-\sigma}$. First set $U(t) = u$ in order to check that the simple case of Section 4.6.1 is recovered. The solution to Equation 4.55 is then

$$V = u\left(\left(\frac{W}{T-S}\right)^\alpha + \sigma\right) \tag{4.59}$$

Hence,

$$\frac{\partial V}{\partial S} = \alpha u W^\alpha (T-S)^{\alpha-1} \tag{4.60}$$

And, substituting these into Equation 4.58 leads quickly to the solution

$$S = T - W\sqrt[\alpha]{\frac{\alpha-1}{\sigma}} \tag{4.61}$$

which of course is Equation 4.13.

However, to illustrate the difficulties of this method, consider instead the case of exponential cost, so that $U(t) = e^t$. The solution to the integral in Equation 4.55 is then very difficult to obtain; an analytic algebra package[7] gives

$$-(\alpha)\ {}_2F_1\left(-\frac{1}{\alpha}, -\frac{1}{\alpha}; 1-\frac{1}{\alpha}; \frac{\sigma e^t}{V}\right)\sqrt[\alpha]{\frac{Ve^{-t}-\sigma}{1-\sigma e^t/V}} = W \tag{4.62}$$

where ${}_2F_1$ is Gauss's hypergeometric function. This is about as far as one can proceed with this method; evidently, numerical approximation is necessary to solve the integrals for even quite elementary cost functions.

---

[7] Wolfram's Online Integrator, `http://integrals.wolfram.com`

4.8 Conclusion

In this chapter, I have argued that minimising energy cost rather than energy itself is a realistic approach for long-running computations, and that it would be quite reasonable for real-world application. In fact, paradoxically, using more energy may be not only more cost-efficient but actually more energy-efficient, due to the structure of the electricity supply system. If such optimisations became a reality, and computing continues to increase its share of total energy usage, then one would expect to see power providers offering a more finely-grained incentive scheme; the framework I have described is capable of refining its schedules for exactly this scenario. The result would be an overall improvement in the efficiency of the world's power generation infrastructure, in terms of both cost and energy, and would therefore form some small part of the solution to the global problems of power provisioning as world population accelerates towards its peak of perhaps ten billion people [Lutz et al., 2001]. Of course there are significant caveats to the efficacy of this solution, but since all expectations are that computing will become only more widespread and diverse, it is clearly a goal worth pursuing.

Having established the legitimacy of the problem, I have also presented a framework in which such problems can be described, and argued for its fidelity as an approximation to the myriad complexities of real systems. As far as possible the results in this chapter apply to almost any plausible power function, since it is hard to imagine any hardware for which this would not be monotonically increasing and convex with respect to processor speed. However, I have also provided more detailed results for one particular class of power function, the dynamic polynomial model, which is believed to be widely applicable in practice. Although these results are necessarily somewhat specific, I believe the techniques used are of broader applicability for other power functions, and it seems likely that similar techniques could produce analogous results for other characterisations of power. Perhaps most importantly, I have demonstrated concretely that these techniques would have a non-negligible impact on the execution strategy used by real workloads, and on the energy cost incurred by those strategies. The margins obtainable in theory are large enough that, although some fraction would inevitably be lost in the translation to the complications of reality, the remainder would still be enough to justify the outlay in technical and conceptual complexity.

In general, of course, the greater the dynamic range of power supported by the hardware, the greater the opportunities for optimisation; for machines in which static power dominates, the naïve race-to-stop approach cannot be beaten for energy efficiency. Indeed, it seems that future hardware trends are in this direction; certainly for larger devices, it is expected that future chip design will focus on larger numbers of simpler computing elements, which individually may not support much power scaling. However, cost optimisation introduces a new front in this struggle between performance and efficiency; as amply demonstrated by Section 4.6.3, there are considerable savings to be made even when there is no margin for energy reduction.

The approach of this chapter is not without its limitations. Section 4.7.2 illustrates that relatively simple formulations can quickly lead to algebraically intractable problems, if one requires precise solutions; although, given the dependence on the notoriously recalcitrant subject of integrals, this should not be particularly surprising. On the other hand, for more challenging cost functions, such as the fully continuous non-monotonic case, a more powerful technique might be needed; in this case, it seems that the calculus of variations is perhaps the right tool. Given the difficulty of even the elementary problems presented above, I have not attempted these cases. Furthermore it is not clear that they would be of much practical utility.

One foreseeable objection to the work of this chapter is that the static, offline analysis of energy costs has an implicit assumption of determinism in the cost model. The real energy market is a complex stochastic system. A simple defence would be that price changes are generally long-term, and are quite predictable at these timescales. However, an important and widely predicted development in future power provisioning could weaken this defence substantially: the proliferation of renewable energy. Renewable energy, like traditional fossil fuel or nuclear power generation, provides little storage capacity and therefore tends to be over-provisioned in times of low utilisation. (Although, in some cases, the drop in supply conveniently mirrors the drop in demand, such as the low production of solar energy at night.) However, many renewable sources also have the radically different characteristic that their production varies over time in a way that cannot be controlled and can barely be predicted; for example, a wind turbine can produce any amount of power from zero to its maximum output, with variations in time, local geography, season and so on, and only broad trends in this value can be identified [Renewable Energy Research Laboratory, 2009]. This would create significant problems for an offline approach if the goal is a power billing infrastructure that more accurately captures the true cost of production. If consumer cost tracked production cost then it too would vary over time according to unpredictable environmental factors, even when aggregated across whole wind farms, solar farms and suchlike. However, there are several considerations that mitigate this problem. First, current power producers are eager to maintain the simplicity of relatively constant power availability, and research is underway into how this might be done as the fraction of power provided by renewables increases [Cavallo, 1995]. Likely candidates include large-scale energy storage using compressed air, efficient long-distance transmission and load balancing to increase the scale of aggregation, and backup from more traditional sources to smooth out demand spikes. Second, computing presents an unusual use-case for the power grid since it barely matters where the computation actually takes place. Rather than moving power to the task, it would be far simpler to move the task to where the cheapest power is. As discussed in Chapter 3, multi-nationals can perform this sort of allocation in siting new data centres; but, in the future, far more rapid relocations of individual tasks might be profitable, and much of the software infrastructure to achieve this already exists [Clark et al., 2005, Zamfir et al., 2007]. If a more dynamic energy market does come to pass by one means or another, it would most likely be better suited to a dynamic power scheduler, but there is considerable pressure to maintain the present straightforwardness for other reasons, and redistribution of computing could actually form part of the smoothing that makes this possible.

In summary, then, this chapter has demonstrated that generalisation to energy cost is economically useful in a broader context, analytically tractable in many cases, and can produce markedly better solutions than are possible with simpler methods.

Chapter 5

Energy-efficient real-time streaming

5.1 Introduction

This chapter investigates another aspect of the energy efficiency problem: energy-efficient streaming. Streaming is itself a multifaceted concept, and the term has related but subtly different meanings in various fields. In this chapter, I define a streaming computation by the following characteristics:

- The computation happens continuously, and need not necessarily terminate.
- New data and computational requests may arrive during the computation; unlike the traditional Turing model, the input is not a static set provided on entry to the program.
- The computation produces intermediate results for particular subproblems. In the non-terminating case, this is of course the only method by which the computation can produce results; again, this differs from the Turing model in which the computation is essentially useless if it does not terminate.
- Suitable measures of efficiency are subproblem throughput, latency, or some combination of the two. This contrasts with the traditional efficiency metric of whole-program execution time, which is not useful for non-terminating programs.

Figure 5.1 presents the comparison with Turing computation diagrammatically. The traditional model comprises three discrete parts: input, computation, and output. Streaming, on the other hand, is a more continuous model, processing a stream (infinite in this case) of new inputs and returning intermediate outputs. This should not be understood to imply that streaming is fundamentally "outside" the Turing model—certainly the same constraints of computability apply, and the Church-Turing thesis is not threatened—but rather that streaming presents an alternative perspective that is more appropriate for certain computational needs.

This is a broad definition of streaming, and encompasses several other concepts that are sometimes called "streaming", all of which are now common parts of the computing landscape. For example, streaming audio and video over a network or from permanent storage is now a common task for desktops and portable devices; this can be seen as consuming a

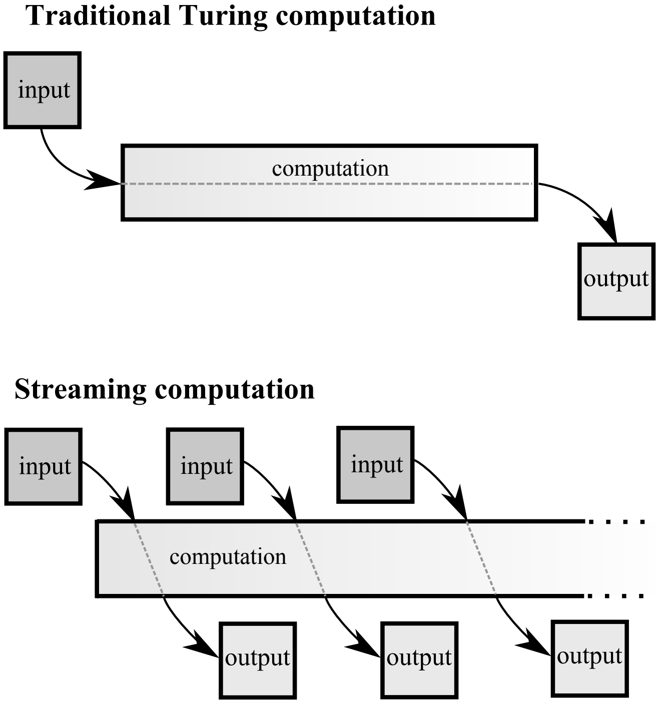

Figure 5.1: Comparison of the traditional Turing model and streaming computation.

series of encoded frames and producing the decoded media stream for the appropriate device driver, with latency and throughput requirement imposed by the sampling frequency and bit-rate of the media object. Examples may be finite (a movie) or essentially infinite (online radio). In another sense of the term, streaming is fundamental to modern computer graphics, where it occurs as a refinement of the Single-Instruction Multiple-Data (SIMD) paradigm. Generating modern 3D graphics is essentially a stream of matrix operations, implemented as a series of identical "kernel" functions applied across different geometrical data; at regular intervals, this stream produces a complete display's worth of graphical output, which is then presented to the user. This has acquired particular practical significance in the last few years with the ubiquity of GPUs able to perform SIMD calculations in a highly parallel manner; at the time of writing, commodity GPUs may provide 1024 cores and performance rated at almost 2.5 teraflops.[1] More recently, as the devices have become more sophisticated, the industry has seen the ascent of general-purpose processing on the GPU (GPGPU) [Wu and Liu, 2008]. GPGPU is intended to leverage the SIMD architecture for a wider class of computational problems, such as simulations of fluid dynamics [Müller et al., 2003]. Now GPGPU has spawned its own languages, which have acquired significant diversity and sophistication in themselves [Hwu et al., 2009, Stone et al., 2010]. In fact, streaming-centric programming languages have a long history of their own, in the form of dataflow programming, which is the ultimate origin of modern GPU streaming [Wadge and Ashcroft, 1985, Smolka, 1995]. Evidently, stream processing in various forms is now a standard requirement of many devices, whether portable or otherwise energy-constrained, and the latter category now includes almost any device anywhere.

---

[1] For example, the Nvidia GeForce GTX 590, `http://www.geforce.com/Hardware/GPUs/geforce-gtx-590/specifications`

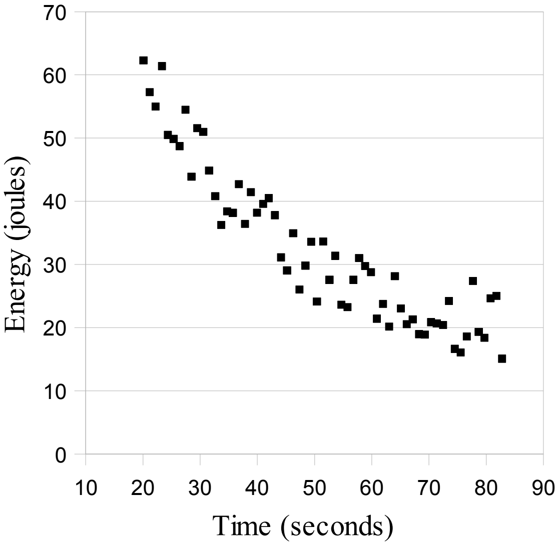

Figure 5.2: Example set of operating points.

In many streaming computations, each unit of work has an associated deadline. For example, each frame of a video stream must be decoded in time for its appearance on the screen, so deadlines are defined by the frame rate of the video. In an energy-sensitive context, such as a mobile device, this computation would ideally be performed so as to minimise the energy per unit of computation while still meeting the deadlines. Existing work typically studies this problem as a dynamic feedback loop, in which the system monitors its own throughput and power consumption and scales performance up or down according to energy and throughput goals, as described in Chapter 3. Certainly this is the approach widely adopted in real systems. Although successful, this approach has two limitations. Firstly, the system must track suitable performance metrics and make real-time decisions, which incurs some energy and performance overhead. Secondly, and more seriously, these systems operate on a "best effort" basis influenced by a complex series of interactions with the environment and the dataset, so it is difficult to provide any hard mathematical guarantees about their performance. This is particularly problematic in a hard real-time context, in which deadlines must not be missed. In this chapter I address these issues by giving an approach that has well-defined mathematical properties, guarantees to find an energy-optimal schedule within the given constraints and, being essentially static, incurs no runtime overhead. Throughout the development of this approach, I illustrate the importance of understanding the broader context of modern mobile and streaming computation.

5.1.1 Definition of an operating point

The basic unit of work in a streaming task will be referred to here as a *frame*. Many factors influence the speed of computation, and consequent energy per frame (EPF), for a given frame and given workload, from hardware to algorithm design to implementation (see Section 5.3). For the purposes of this chapter, it is enough to observe that this diversity in time and energy exists and can be controlled by some means or another. To abstract away the details of how this variety is provided, I introduce the concept of an *operating* *point.*

Definition: An *operating point* describes the following information for a given task and a given platform on which that task is being streamed:

1. A description of a configuration of the hardware and software environment in which a frame might be processed. This might include the state of each hardware component (active, hibernating, depowered and so forth), referred to in Section 3.2 as the performance state; any global decisions about the program that will process the frame, such as what that program is and what its general parameters might be; and any other global indicators for the system, such as whether it is receiving mains or battery power. This part is considered *opaque*: the operating point is not expected to provide a description of this environment in a structured way; it should be seen only as a token that some such configuration exists.
2. A characterisation, in some fashion, of the energy and time required to process a frame in that configuration. This part is transparent; the operating point must provide some means to query what these values are, in whatever manner they are described.

The nature of the energy and time characterisation depends on the purpose for which the operating point is intended. At its simplest, an operating point can be regarded as a point in energy–time space; Figure 5.2 shows an example set. The "time" component represents the worst-case execution time for a frame, measured in wall clock time; the "energy" component represents the average-case energy. The use of the worst case for time on the one hand but average case for energy on the other is motivated by distinct purposes for which these two values will be used: specifically, the need to bound the execution time but optimise the energy. This illustrates the connection between the intended usage and the description given. Note that one cannot say anything specific about the way in which frames can be produced at those energy–time values, but only that there is in fact some suitable configuration.

5.1.2 Assumptions regarding operating points

I make the following assumptions about the use of operating points:

- The operating point can only be changed between frames, not within them. This is necessary because no assumptions are made about the source of variability in the operating points or the application being run. For example, it is not possible in the general case to change the number of threads of execution in the middle of a frame without restarting the frame or incurring some other substantial penalty. Of course other parameters might be variable within a given frame, such as clock frequency, but this is lost in the abstraction.
- Frames are homogeneous; the execution time of all frames is bounded by the same worst-case value or, in the probabilistic case, is drawn from the same distribution. Any problem can be converted into this form by aggregating probabilities or worst-cases, at the cost of losing some information.

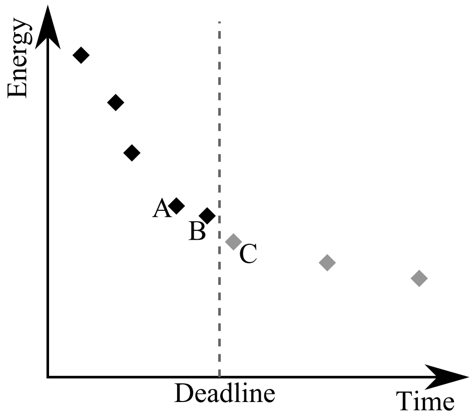

Figure 5.3: An example set of operating points and a per-frame deadline shown in energy–time space.

- Transitions between operating points incur no energy or time penalties. Even if non-zero, switching energy has little impact on the optimal choice of voltage [Swaminathan and Chakrabarty, 2001]. I assume that this result holds for other operating point parameters.
- The set of operating points is finite. In particular, CPU voltage and frequency, if it is variable at all, can only be sampled at finitely many values. For processors that support only a fixed number of frequencies and voltages, this is unproblematic; in practice many processors only support frequencies of the form $f, f/2, f/3, \ldots$ where f is the maximum frequency. For processors that do support essentially arbitrary values, the energy penalty for restricting consideration to a finite subset is negligible, even for an ideal processor [Lee and Sakurai, 2000].
- If buffering of frame results is required at all, the energy overhead it incurs is negligible. As discussed below, this assumption can in fact be weakened without substantial impact on the arguments presented below. However, for the purpose of simplication, buffering will generally be neglected.

In later sections, I extend the manifestation of the "operating point" concept to include more information, but the core energy–time characterisation is maintained. The assumptions regarding buffering and transition costs could be lifted at the expense of increased analytical complexity, but this is not addressed here.

The assumption of frame homogeneity has the important consequence that there is no particular reason to favour processing a later frame before an earlier one; in other words, one can assume without loss of generality that frames are processed in the order in which they arrive. This also alleviates any concerns about data dependencies between frames, since no attempt to reorder them is made.

5.2 Problem statement

The objective of the work in this chapter is simple: to minimise the energy requirement of streaming; in particular, to make the best possible use of the available operating points in order to minimise the EPF. Let OP be the set of operating points for a particular task. The naïve approach to energy optimisation can then proceed straightforwardly: produce the set $OP^\star \subseteq OP$ of points with time component low enough to guarantee the required throughput; then select from $OP^\star$ the point with the lowest energy component. (Other constraints, such as instantaneous power limitations, must be addressed by filtering the operating points in advance.) Figure 5.3 shows a set of operating points in arbitrary units. The grey points are discarded since they are too slow for the deadline shown. Of the remaining points, B has the lowest energy, and therefore the system will process frames at this operating point. However, this chapter explores a more flexible refinement that may make better use of the available operating points. By interleaving frames of different speeds such that the overall throughput still meets or exceeds requirements, it may be possible to reduce the overall EPF for the average frame. Operating points that are slower than the required deadline may be used provided they are interleaved with other, faster operating points. Essentially, the operating points may be *dithered* to produce a better overall result. Figure 5.3 illustrates that points A and C are sufficiently fast that their average time would be to the left of the deadline, and their average energy is lower than the energy of B. Therefore, by processing frames at operating points A and C alternately, the throughput requirement can be respected even though C $\notin OP^\star$, and the average energy per frame can be reduced. My hypothesis is that opportunities for such optimisation occur quite widely in practice.

This is not without its complications. For example, if frames must be held until their deadline, they may require buffering, and the extra power for storage must be taken into account; as above, this overhead is generally discarded here. In other cases, it may be that there is no cost to producing some frames ahead of schedule provided the overall throughput requirement is maintained. There is the additional concern that frames may not *arrive* in time to be processed; respecting this constraint is trivial in the single-point case but may become more involved in the generalisation to multiple operating points. Fortunately, one can always arrange the period so that the slowest operating points occur first. On the other hand, if this limitation is not present (for example, if all the frames are available from the beginning), this decrease in energy may also produce a decrease in start-up latency, since the fastest points can be placed at the start of the period. For example, in Figure 5.3, the first frame could be processed at point A, and would thus be available earlier than under the naïve solution using point B.

I now state the problem precisely, using the concept of operating points. Consider processing an infinite stream of frames, each at a specified operating point. In the parlance of real-time systems, this represents an infinite number of jobs from a single periodic task [Liu, 2000]. I wish to give a finite static description of the assigned operating points, so I consider the mapping of frames to operating points to be periodic; in other words, the algorithm will produce a list of operating points $P_1, \ldots, P_L$ such that frames $1, L+1, 2L+1, \ldots$ are executed at P_1, frames $2, L+2, 2L+2, \ldots$ are executed at P_2 and so on. The aim is to find $P_1, \ldots, P_L \in OP$ (not necessarily distinct) such that the average time required to complete each frame is below some threshold and, subject to this constraint, the objective is to minimise the average energy required per frame. The optimal sequence may require several frames to be computed at the same operating point, and some available points may be unused, requiring too much time or energy. Since the only concern at this optimisation phase is the average time and energy over the period, the ordering of the operating points is not important; a later stage can permute the order according to other constraints, as discussed above. Therefore only the number of uses of each point is relevant.

It might seem surprising in view of the developments of Chapter 4 that the focus of optimisation here is raw energy, rather than energy cost. However, in a mobile context, the two concerns are essentially equivalent. When a battery is the only source of energy, the energy "cost" is fixed with respect to time. This is the degenerate case of the generalised problem, which coincides exactly with the original cost-agnostic solution. The monetary cost of the energy required to charge the battery of a mobile device is negligible compared to, say, the cost of the device itself. Perhaps the cost model might be applicable in some situations; for example, perhaps energy is more valuable when the remaining charge of the battery is low, and indeed modern devices often include a more conservative power mode for this eventuality. However, this is not considered here.

To state the problem precisely: let OP be the set of m points defined by

$$OP = \{(t_1, E_1), \ldots, (t_m, E_m)\} \tag{5.1}$$

with t_i and E_i the time and energy of point i as previously described. Assume without loss of generality that $t_1 \leq \ldots \leq t_m$. For each $i \in \{1, \ldots, m\}$, let $n_i \in \{0, 1, \ldots\}$ be the number of frames executed at operating point i in each period, so that $L = \sum_{i=1}^{m} n_i$ is the length of the period. Let T be the maximum permissible average time per frame, so that $1/T$ is the minimum throughput over the period.

Let Ψ_T be the average time per frame, so

$$\Psi_T = \frac{1}{L} \sum_{i=1}^{m} n_i t_i \tag{5.2}$$

and Ψ_E the average energy per frame, so

$$\Psi_E = \frac{1}{L} \sum_{i=1}^{m} n_i E_i \tag{5.3}$$

Now the problem can be stated as:

Given OP and T, find $n_1, \ldots, n_m$ such that Ψ_E is minimised and $\Psi_T \leq T$.

5.3 Related work

The concept of a hard real-time system is assumed to be broadly familiar, and therefore many of the technical details are omitted as tangential to the main argument. This chapter generally adopts the terminology of Krishna and Shin's *Real-Time Systems* [Krishna and Shin, 1997]; each term is defined inline as it is introduced here.

The idea of operating point dithering is a generalised form of voltage dithering. In voltage dithering, an arbitrary operating voltage is approximated on a processor with a finite number of available voltages by altering the voltage dynamically in a regular pattern. This can accurately approximate arbitrary voltage scaling with as few as four fixed rails [Gutnik and Chandrakasan, 1997]. Voltage dithering alone has been shown to reduce energy consumption by up to 44% for certain applications [Putic et al., 2009]. My approach extends this by allowing other sources of power and performance variability to be exploited.

As indicated above, there are many factors which influence the speed and power of computation. The basic dichotomy, as discussed at length in Chapter 4, is that faster computation typically requires more dynamic power but, since the duration of computation is reduced, expends less energy on static power. If voltage–frequency is the only variable considered, dynamic power is approximately quadratic or cubic with respect to speed [Cho and Melhem, 2008]. However, dynamic power is influenced by many other factors such as cache sizes, depth of pipelining and instruction-level parallelism. Many modern architectures support alteration of some of these parameters dynamically, through technologies like dynamic voltage scaling, and one can thereby construct energy-efficient schedules [Govil et al., 1995]. Other factors are also significant. Memory usage and access patterns can make a substantial difference; for real-time media streaming, focusing on this alone has been seen to reduce power by a factor of 3.6 for only a 5% performance sacrifice for modern decoders such as MPEG-2 [Kulkarni et al., 1999]. For workloads that can be processed in parallel, increasing the number of processors used may actually increase energy efficiency by better amortising the static power [Li and Martínez, 2005]. At the software level, different algorithms for the same problem may have make different time-space trade-offs, which has consequences for energy consumption in terms of execution time and energy expended in the memory hierarchy. Within a given algorithm, different implementations may also have radically different power characteristics, depending on which functional units within the processor are used, which co-processing units are involved, or other external factors such as memory access patterns. Furthermore, compilation techniques exist that trade executable size for execution time, or make more targeted trade-offs with energy efficiency, as discussed in Section 2.5. Additional work in this area considers artificially extending the ranges reported by live variable analysis (LVA), producing a negative impact on efficiency of register allocation, but allowing slower (and more energy efficient) registers to be used [Menon et al., 2003]. Clearly these software choices can be deferred until execution itself if the implementer is willing to provide the relevant alternatives, such as multiple implementations for a given problem. One can even imagine a dynamic despatch model in which more or less energy-hungry code paths could be selected at runtime, although it appears that no compiler supports this natively at present.

A great deal of existing work addresses the problems of measuring and simulating the energy–time values needed for the simplest manifestation of an operating point [Monchiero et al., 2006, Brooks et al., 2000, Muralimanohar et al., 2007]. There are also powerful techniques for inferring large numbers of such values without direct simulation [Lee and Brooks, 2006, Engin and McKee, 2006]. Consequently, the means to generate accurate values for significant quantities of realistic operating points already exists. This chapter explores ways in which this data can be put to good use.

Previous work has investigated best-effort systems for energy efficiency, which trade power for performance according to specified power and throughput goals. These typically focus on extracting trends from previous behaviour and extrapolating to future workloads [Govil et al., 1995, Hong et al., 1999, Leung et al., 1999, Dubach et al., 2010]. This work shows that significant gains are possible, and its ideas have been successfully deployed in practice. In some cases these systems can guarantee a hard deadline while still reducing energy by over 90% under realistic assumptions, although unlike the framework of this chapter they usually do not support general-purpose workloads [Lee and Sakurai, 2000]. Qu and Potkonjak describe an algorithm that learns the relationship between power and performance at run-time and scales accordingly [Qu and Potkonjak, 2000]. Their approach considers this space to be continuous, and is shown empirically to make significant energy savings, although again its results cannot be described precisely. Swaminathan and Chakrabarty describe a scheduling algorithm that guarantees hard deadlines and minimises energy, and give an approximation algorithm fast enough to execute in real time [Swaminathan and Chakrabarty, 2001]. However, their approach allows for only a single processor with a choice of two frequencies and no other dynamic parameters. Gruian explores a probabilistic description of execution times that does not require the assumption of worst-case performance [Gruian, 2001]. This approach is similar to mine although no bounds are proven and again the variability in performance is attributed exclusively to DVFS. Energy savings of 40–80% for tasks are observed for some benchmarks with uniformly distributed execution times.

As mentioned above, it is assumed that frames are processed in order. Frame reordering can be a powerful technique if there is some variation between frames and some simple means of differentiating them [Gruian and Kuchcinski, 2003]. For example, the MPEG video standard defines three types of frame (I, B and P), each of which requires a different technique to decode. By processing frames with more predictable execution times first, more flexibility is created for scheduling of the more variable frames later. However these techniques are not directly comparable to those demonstrated here.

More formal work includes Qiu and Pedram's modelling of power scheduling, which they consider as a continuous-time Markov decision process [Qiu and Pedram, 1999]. This approach models devices with multiple power states and considers transition costs, so it is suitable for practical use. Due to its firm mathematical footing, this approach consistently outperforms heuristic approaches in energy efficiency and latency. Its primary limitation is that solving the relevant Markov equations is rather computationally intensive and therefore difficult to undertake in real time without specialised hardware support. Other work has studied energy efficiency with provable properties by placing tight constraints on the permitted workloads. For example, it may be necessary for the producer of the stream to insert markers denoting the amount of work that the consumer will need to undertake to process each frame [Chandrakasan et al., 1996]. This may be plausible in, for example, an MPEG encoder or other compression frameworks. MPEG decoding in particular has been extensively studied; for example, Choi *et al.* exploit domain-specific nuances to produce energy savings of 80–90% [Choi et al., 2002]. In this case, assuming the worst-case decoding time is wasteful because the inter-frame variability is high, so they describe heuristics to improve decode-time prediction. They also exploit the distinction between types of frame in the stream, which have very different decoding properties. The techniques in this chapter are, in their current formulation, unable to take advantage of such domain-specific features.

5.4 Algorithm description

Allowing arbitrary interleaving of operating points results in exponential growth of the search space with respect to the number of available operating points. However, the similarity of the problem to existing linear programming examples suggests that an efficient dynamic programming algorithm can be found; in particular, there are strong echoes of the integer knapsack problem [Garey and Johnson, 1979]. Indeed there is an analogous algorithm for this problem, although there are significant subtleties.

The algorithm proceeds as follows. Assume the deadline T and all frame execution times t_i to be integers. (This assumption can be satisfied by suitable selection of measurement unit.) Define $e_{t,L}$, for each $t \in \{0, \ldots, T\}$ and $L \in \{1, \ldots\}$, to be the minimal total energy across the period, with total time no greater than t, using exactly L frames. A recursive definition follows. First, there is no combination of L frames with total time less than $t_1 L$, so if $t < t_1 L$ then let $e_{t,L} = \infty$. Otherwise,

$$e_{t,L} = \min\left(e_{t-1,L} \ , \ \min_{t_i \le t} (E_i + e_{t-t_i,L-1}) \right) \tag{5.4}$$

In words, the minimal energy with time limit t and L frames is either:

- the minimal energy with time $t-1$ and L frames, or
- the energy of one frame plus the minimal energy with the remaining time and $L-1$ frames.

The minimal EPF for a given L is then $e_{LT,L}/L$. So the overall minimal EPF is

$$\min_{L \ge 1} \left(\frac{e_{LT,L}}{L} \right) \tag{5.5}$$

Algorithm 1: Dynamic programming algorithm to find minimal energy which meets the throughput requirement.

```
Input: Operating points $OP = \{(t_1, E_1), \ldots, (t_m, E_m)\}$ with each $t_i \in \mathbb{N}$ and
       $E_i \in \mathbb{R}^+$, per-frame time limit $T \in \mathbb{N}$ and maximum period $N \in \mathbb{N}$.
Output: The minimal EPF for $OP$ within the given constraints.
$\epsilon_{opt} := \infty$;
$p := [\infty, \ldots, \infty]$;                          /* $NT$ elements */
for $L = 1, \ldots, N$ do
    $r := [\infty, \ldots, \infty]$;                      /* $NT$ elements */
    foreach $t \in \{t_1 L, \ldots, NT\}$ do
        $r[t] := r[t-1]$;
        for $i = 1, \ldots, m$ do
            if $t_i \le t$ then
                $r[t] := \min(r[t], E_i + p[t - t_i])$;
    $\epsilon_{opt} := \min(\epsilon_{opt}, r[LT]/L)$;
    $p := r$;
return $\epsilon_{opt}$;
```

There is no obvious way to compute an upper bound on L; no matter how great the value of L, it may be that some longer, more complex interleaving would produce further savings. In practice such a bound is usually imposed by the need to buffer results in a queue of finite size, or by some latency requirement from the consumer, so one can reasonably require a parameter N, the upper limit on L, to be provided as an input to the algorithm; in other words, apply the additional constraint that

$$\sum_{i=1}^{m} n_i \leq N \tag{5.6}$$

If the stream is in fact finite then N could be set to its total length, although this is unlikely to be practical. In cases where there is no apparent limit, the algorithm may be retried with larger values of N until a satisfactory solution is reached. Section 5.5 analyses whether this artificial limit has any noticeable effect on the quality of the solutions produced.

Furthermore, if one did wish to impose a penalty for long-period solutions, this could be embedded into Equation 5.5; for example, by choosing a suitable constant c, one could instead compute

$$\min_{L \geq 1} \left(\frac{e_{LT,L}}{L} + cL \right) \tag{5.7}$$

This might be used to represent the energy overhead of buffering, allowing the earlier assumption of zero-overhead buffering to be discarded. It also imposes a natural upper bound on L, since the search could be terminated once cL alone is larger than the best solution found. The algorithms in this chapter could readily be modified to admit this more general case, but for clarity these modifications will not be given explicitly.

This definition is exploited by the dynamic programming solution in Algorithm 1, which iterates L from 1 to N, finding the optimal EPF in each case. The array r holds $e_{t,L}$ for all values of t and the current value of L, while p holds equivalent values for the previous value of L, as required by Equation 5.5.

The time and space characteristics of the algorithm are straightforward to obtain. The time complexity is obviously $O(mN^2T)$ from the structure of the nested loops. As with the integer knapsack problem, this is linear time with respect to m and pseudo-polynomial time with respect to N and T. The algorithm is *pseudo*-polynomial because, while it is polynomial with respect to N, the size of N in the input is $O(\lg N)$, given the compact natural representation of integers, and likewise for T. Therefore, more precisely, one should say that the algorithm requires $O(2^t 4^n m)$ time where n and t are the number of bits in N and T respectively, but for practical purposes this is rather misleading. The space complexity is similarly simple to determine. The arrays r and p are initialised to size NT, and all elements may be non-zero, precluding a sparse representation. The remaining variables are of constant size, so the space requirement is simply $O(NT)$. This is analogous to the best exact result for the knapsack problem [Toth, 1980].

5.4.1 Dominated points

Given the numerous sources of variation that may generate operating points—such as hardware settings, implementation settings, and battery characteristics—it is reasonable to expect that m is quite large in practical cases. Therefore it is worth investigating some general-purpose optimisations to this algorithm.

One valuable observation is that many of the points in a typical OP can never form part of an optimal solution. These points can be eliminated cheaply before the main algorithm is applied, thereby shrinking the effective value of m, and significant execution-time savings may result.

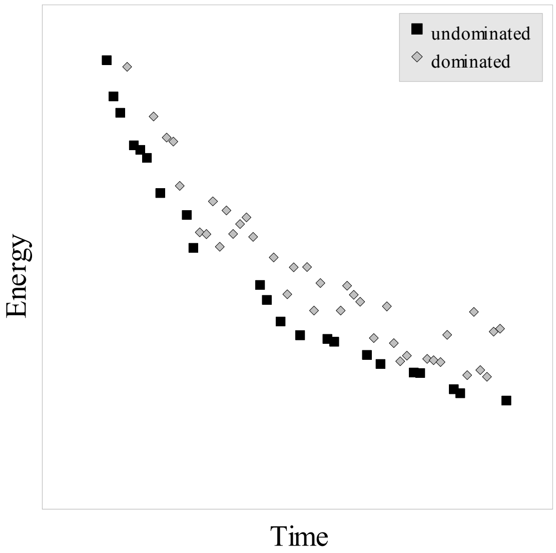

Figure 5.4: Example set of operating points, showing dominated and undominated points, arbitrary units.

Algorithm 2: Algorithm to find undominated operating points.

```
Input: Operating points $OP = \{(t_1, E_1), \ldots, (t_m, E_m)\}$ with $t_1 \leq \ldots \leq t_m$.
Output: The subset of undominated points in $OP$.
r := {};
lowest := $\infty$;
for $i = 1, \ldots, m$ do
    if $E_i \leq lowest$ then
        lowest := $E_i$;
        $r := r \cup \{(t_i, E_i)\}$;
return r;
```

To illustrate this idea, I describe one particular form of eliminable point, analogous to "dominated points" in the integer knapsack problem [Zhu and Broughan, 1997]. I follow that convention and call these points *dominated.* The redundancy of dominated points can be seen by a simple "cut and paste" argument: assume point i requires more time *and* more energy than point j. Then any solution that used point i would be faster and require less energy if all uses of i were replaced by j. Therefore i cannot be part of the optimal solution, so i is dominated and can be eliminated. Figure 5.4 shows an example set of points divided in this way.

Undominated points can be identified in a single pass of OP taking $O(m)$ time and $O(1)$ space; see Algorithm 2. In fact, this problem is well known to economists as it is analogous to finding the Pareto frontier [Gibbons, 1992, p. 88]. In algorithmics this is known as the skyline problem or the 2-dimensional vector maximum problem, and the correctness and efficiency of Algorithm 2 is well-established [Kung et al., 1975]. This algorithm moves forwards through the list of points, which (by earlier assumption) are sorted by their time component, excluding any point requiring more energy than the lowest-energy point occurring before it. Analysing the time and space complexity of this algorithm is simple, but one might ask how many points it is likely to eliminate in a typical case. This question appears not have been answered before; the solution can be estimated by considering the action of the algorithm from an alternative perspective. One can imagine that the algorithm finds the lowest energy values amongst those with time value less than a given maximum, for successively increasing values of that maximum. As the maximum time is increased and new energy lows are found, each of the lows can be seen as a "record-breaker". The expected number of record breakers in m attempts is known to grow with the harmonic sum H_m [Havil, 2003, p. 125], given by

$$H_m = \sum_{i=1}^{m} \frac{1}{i} \tag{5.8}$$

and, surprisingly, this result is independent of the underlying distribution. This sum diverges extremely slowly as m goes to infinity, and can be approximated asymptotically using the result that

$$\lim_{m\to\infty} H_m = \gamma + \ln m \tag{5.9}$$

where $\gamma \approx 0.57721\ldots$ is the Euler-Mascheroni constant [Havil, 2003, pp. 69–73]. So on average the running time of the search is improved from $O(TN^2m)$ to $O(TN^2 \ln m)$. However, this result is subject to the assumptions that the operating point energies are independent and identically-distributed; in reality neither of these assumptions is completely respected, but as an approximation this is promising. Section 5.5.4 evaluates this assumption against real data.

Several other notions of dominance have been described for the integer knapsack problem, each allowing certain points or combinations of points to be eliminated from consideration; for example, collective dominance, threshold dominance, multiple dominance, and modular dominance [Poirriez et al., 2009]. It is possible that they too have analogues in the operating point problem. However these are complex properties, and identifying and exploiting them in a concrete algorithm is rather involved; therefore this avenue is not explored further in this chapter.

5.4.2 Downscaling

Two useful transformations can further reduce the effective values of the parameters. First, all time values can be divided through by their greatest common denominator, $\gcd(\{T, t_1, \ldots, t_m\})$. This is equivalent to selecting the largest possible unit of time that can represent all time values exactly. Assuming momentarily that the values are independent and uniformly random, the probability that these $m + 1$ values have a GCD greater than 1, and therefore this optimisation actually reduces the time values, is given by the following equation [Nymann, 1972].

$$\mathbf{Pr}(GCD(\{X_1, \ldots, X_m\}) > 1) = 1 - \frac{1}{\zeta(m+1)} \tag{5.10}$$

where each X_i is a suitable discrete random variable and ζ is the Riemann zeta function [Apostol, 2010]. Unfortunately this tends rapidly to zero; see Figure 5.5. However, the GCD can be computed in $O(m \ln t_1)$ time in the worst case (assuming t_1 to be the smallest value involved), even using the simple Euclidean algorithm [Knuth, 1997]. This

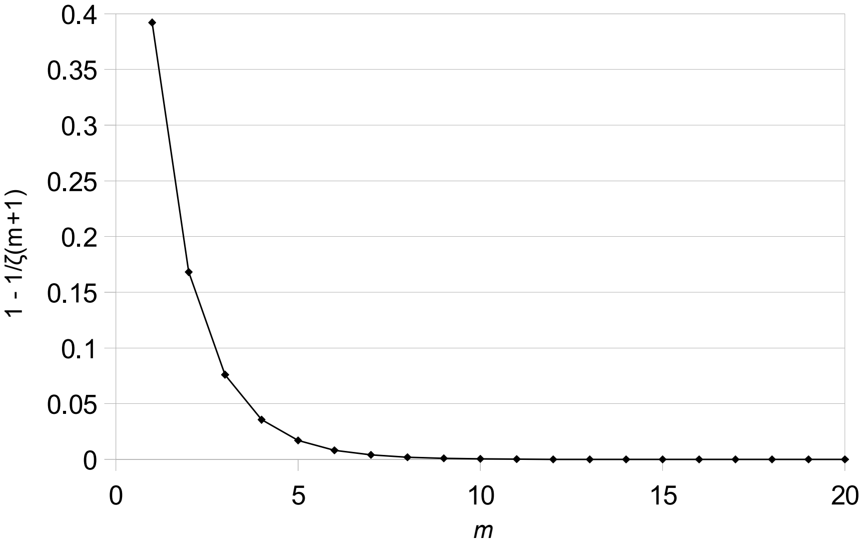

Figure 5.5: Probability that integer time values generated uniformly at random have a common factor, as a function of m, the number of values.

is so inexpensive that my implementation performs it anyway, and for non-uniform values this may well be productive.

Secondly, consider subtracting some constant k from all t_i's. Then

$$\sum_{i=1}^{m} n_i(t_i - k) = \left(\sum_{i=1}^{m} n_i t_i\right) - kL \tag{5.11}$$

and consequently the constraint on Ψ_T can be rewritten as

$$\frac{1}{L}\sum_{i=1}^{m} n_i(t_i - k) \leq T - k \tag{5.12}$$

for any k. Therefore one is also free to subtract any constant from all time values. A natural candidate is $k = t_1$ since this is the smallest time value that appears.

Therefore, one can add or subtract any constant to the time values, and multiply or divide (subject to maintaining the integer constraint) by another constant. In other words, these two results in conjunction permit a linear transformation of the given times, which, by reducing the effective value of T, can be exploited to save both time and space.

5.5 Evaluation

This section measures the performance of the schedules produced by the algorithm, and the amount of work required to produce them. It also evaluates the success of the heuristics described above.

5.5.1 Diminishing returns

First, let us investigate whether the artificial limit on L has a significant effect on the quality of the solutions produced. Let $\Psi_{E[\nu]}$ be the energy of the best solution found

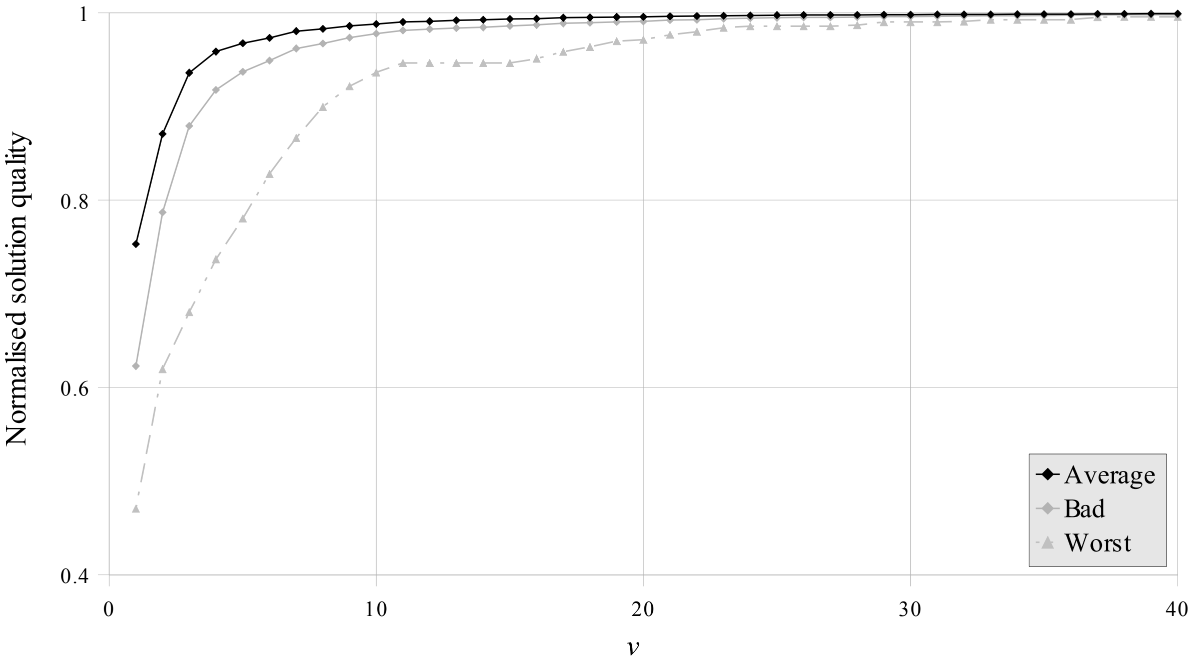

Figure 5.6: q_ν for increasing values of ν, averaged over one hundred random examples. Note that the vertical axis starts at 0.4.

with $N = \nu$. Let $M = \min_i E_i$ be the smallest energy value of any point in OP, so the theoretical limit for any solution would be $\Psi_E = M$, and of course in most cases this is unattainable. Therefore, define the following quality metric:

$$q_\nu = \frac{\Psi_{E[\infty]} - M}{\Psi_{E[\nu]} - M} \tag{5.13}$$

By subtracting M from the achieved energy values, I consider only the fraction of available headroom that the solutions exploit, so this allows a measure of the success of the algorithm somewhat independent from the limitations of the operating points provided. Alternatively, one could see this as shifting the base line of the comparison from zero energy (clearly impossible) to the minimum energy that could be achieved in the absence of a deadline (maybe still impossible, but more reasonable). Note also that this expression is undefined if $\Psi_{E[\nu]} = M$; since this occurs only when the theoretical limit is achieved, take $q_\nu = 1$ in this case.

The algorithm was tested with one hundred sets of random operating points. These points were generated randomly in several ways. Fifty sets were generated by jittering BSOM data (see Section 5.5.2); in other words, applying a small random offset to each point. A further fifty were generated *ab initio*, with a normal or uniform distribution providing each parameter value. The points were filtered to remove negative time and energy values, and then further filtered to remove dominated points. Each set contained exactly thirty points after filtering.

Figure 5.6 shows the behaviour of the quality metric for increasing values of ν. The "average" curve shows the mean values of q_ν over these random sets. Evidently, the optimal solution is usually found by $\nu = 20$ and there is little penalty even for values as small as $\nu = 10$. The "bad" curve shows the mean minus one standard deviation, representing

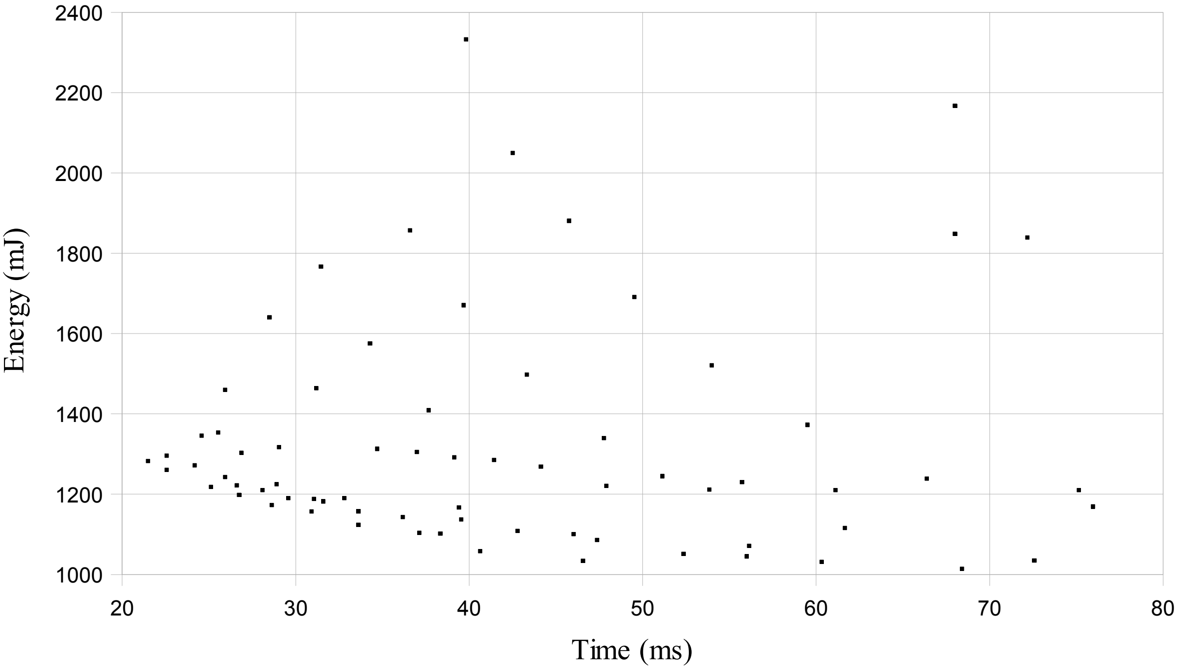

Figure 5.7: Energy–time diagram for real BSOM operating points.

the quality of somewhat more difficult examples. These typically require larger limits to achieve good solutions but even then $\nu = 20$ is almost indistinguishable from the optimal solution. Finally the "worst" curve shows the lowest values of q_ν observed over all OP sets for each value of N. Evidently even the most difficult cases yield an almost perfect solution for $\nu = 40$. One can reasonably conclude that this artificial limit does not significantly reduce the quality of the solutions found, and the N^2 factor in the algorithm's time complexity can be kept under control.

5.5.2 Simulation

So far I have made a theoretical argument for my hypothesis that operating-point dithering can make measurable savings, and in this section I proceed to test the hypothesis against real data. This experiment was conducted on the energy and time values measured for execution of the data mining application *BSOM* [Li and Martínez, 2005]. For this application, energy–time values were readily available for a range of operating points, with various voltage and frequency levels and up to sixteen processors. In total this made for 71 operating points, shown in Figure 5.7. BSOM is not intended as a streaming algorithm, so successive runs of the algorithm can be interpreted as separate frames, guaranteeing the independence axiom.

For comparison, I also simulated the performance of a greedy dynamic algorithm. This algorithm selects the lowest-energy point among those which will complete before the next deadline. This choice is repeated after each iteration, so slack time accumulated from earlier fast frames can be used to choose slower but potentially more efficient points in later iterations. The simulation was run for 10 000 iterations and the average energy per frame recorded. This greedy dynamic algorithm is not a full reflection of a real dynamic

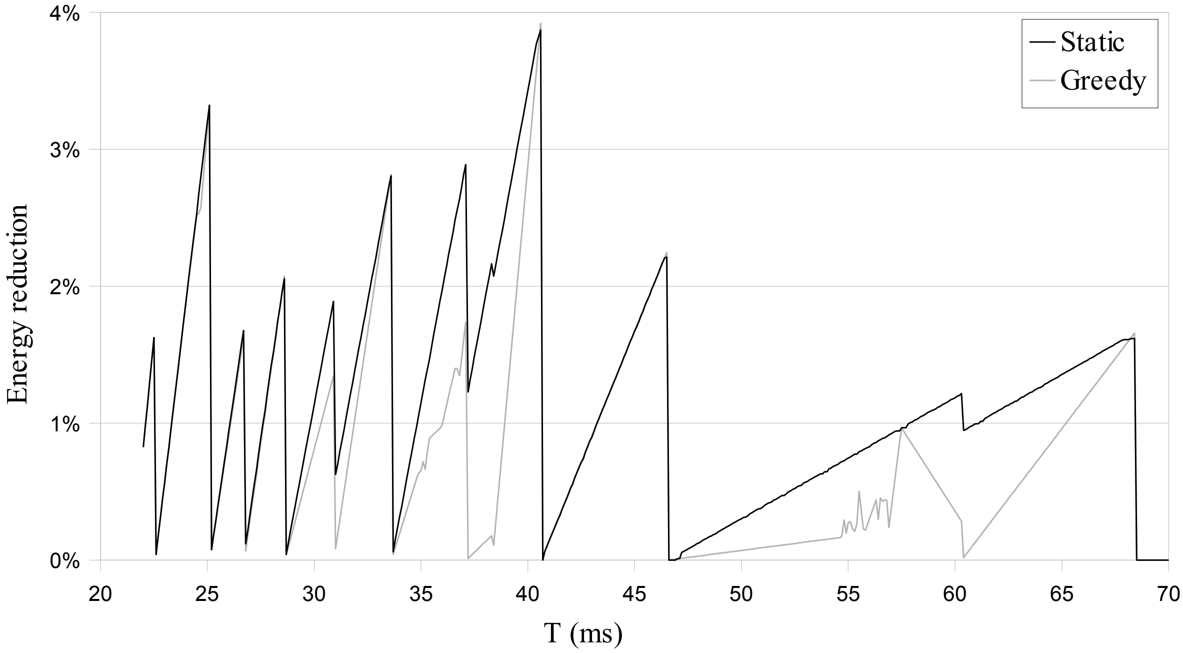

Figure 5.8: Comparison of the per-frame energy reduction achieved by my static algorithm and a dynamic greedy algorithm.

algorithm, because part of the value of a dynamic algorithm comes from its ability to respond to tasks completed in less than their worst-case execution time; my current model assumes that tasks always take the full time, so the greedy algorithm is somewhat neutered in this respect. (This limitation is lifted in Section 5.6.) Both algorithms are compared with the naïve static algorithm, which simply selects the lowest-energy point that meets the timing constraint at the beginning and persists with it, as described in Section 5.2.

Figure 5.8 shows the performance of the two algorithms normalised by the naïve solution. I sweep the entire valid range of T values for the data in Figure 5.7 to avoid accidental cherry-picking. The static algorithm uses $N = 40$ and, as shown above, this will find an almost-perfect solution in almost every case, providing a tight lower bound for the algorithm's performance. The behaviour with respect to T is quite volatile as different operating points become viable. However, the overall trend is clear: my algorithm outperforms the greedy algorithm across the board, although the margin is quite variable, reaching 2% and averaging around 0.3%. In some cases the two algorithms produce indistinguishable results. It might appear disappointing that the peak energy saving is barely 4% and my approach is only marginally better than a greedy real-time algorithm. However, there are two counterpoints to consider. First, the behaviour of my static solutions is better-defined than that of the real-time algorithm, which is not predictable without simulation and can vary chaotically with small changes in T (as seen in the region around $T = 56$ ms); therefore my solutions are in some sense preferable even if the energy saving is no greater. Also, the baseline comparison is somewhat unfair to both algorithms because I have not normalised for M_1; in other words, it may be that greater savings than these few percent are simply impossible with the operating points available. This normalisation is applied in Figure 5.9. Evidently, the margin between the two algorithms grows considerably; on average, my approach exploits over 10% more of the available headroom,

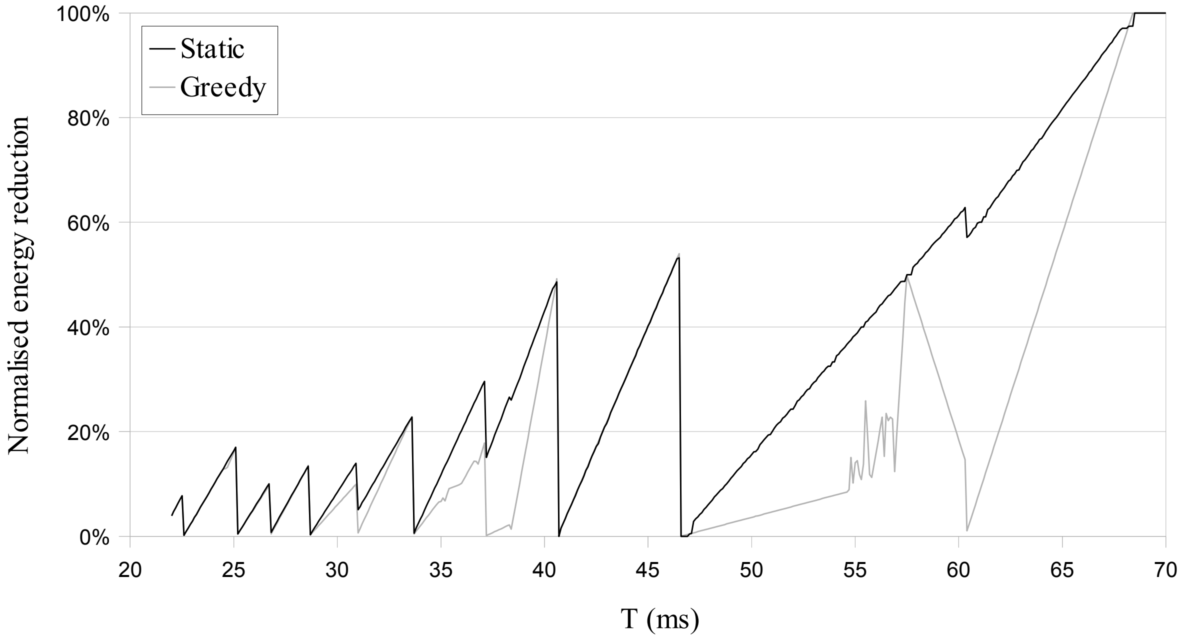

Figure 5.9: Comparison of the per-frame energy reduction achieved, as a fraction of the available maximum.

Optimisations	Time *(ms)*
None	5520
+ Dominated-point removal	1390
+ Downscaling	758

Table 5.1: Average solution time.

with the differential reaching 56% in some cases. So while in real terms the savings may only amount to an increase in battery life of 4% or less, this is largely a limitation of the dynamic range permitted by the hardware and not the algorithm. To put it another way, it is possible to get considerably closer to the optimal deadline-free energy, but in absolute terms there is not, in this example, much room for improvement.

5.5.3 Heuristic improvements

Sections 5.4.1 and 5.4.2 proposed two heuristics that, while not affecting the result returned, might improve the practical performance of the algorithm. Table 5.1 shows the effect of these optimisations on the time taken for the algorithm to run on the BSOM data, averaged over $T = 30, 31, \ldots, 50$. The timings are taken from a rudimentary implementation on a commodity desktop machine.[2] Evidently both forms of optimisation prove productive; the overall speedup is over 86%. With all optimisations, the average time is less than a second, which is clearly tolerable for offline usage. Micro-optimised implementations could undoubtedly push this considerably lower.

---

[2] The machine in question is a 2.40 GHz four-core Intel Core 2 Quad with 6 GB RAM running Ubuntu 10.04 ("Lucid Lynx"). The algorithm is implemented in single-threaded Java, although the Java Virtual Machine itself may take advantage of additional cores internally.

Point type	Number of points
All points	71
Undominated points only	13
H_n	4.85...
$\gamma + \ln n$	4.84...

Table 5.2: Point counts for BSOM data, and comparison to heuristic estimates.

5.5.4 Evaluation of domination

Table 5.2 shows the number of points in the real example alongside the estimated counts given in Section 5.4.1. Evidently the approximation $H_n \approx \gamma + \ln n$ is already quite precise for $n = 71$, but these are nevertheless significant underestimates for the true number remaining after the removal of dominated points. This presumably reflects the fact that the values are not, in reality, generated independently from an underlying probability distribution, but rather bear an approximate correlation: slower points will, in general, tend to require less energy. However as an asymptotic approximation the discrepancy is tolerable.

5.6 Probabilistic generalisation

So far I have assumed the time values to be worst-case limits, and that therefore the timing constraint can be guaranteed. In many situations, this is inadequate. For some classes of streaming computation, the worst-case frame is significantly slower than the average case, and therefore this would be a highly inefficient solution to the problem. In other situations, there may be no finite worst case for particular frames, so no approach to scheduling can guarantee a deadline, and these problems would simply be insoluble in the framework presented thus far.

These problems can be handled by generalising the problem and extending the values contained within the operating point. In particular, a probabilistic approach can provide a much richer description that encompasses the sort of problems described above.

5.6.1 Probabilistic model

Here I attempt to give the simplest possible probabilistic model, in order to generalise the earlier deterministic model with as few extra assumptions as possible.

First, since it must now be possible for deadlines to be missed (in order to support frames of unbounded size), this requires a more general metric for "quality of service" than a simple guarantee to meet all deadlines. This section adopts the parameter p which is simply a limit on the probability that any given frame misses its deadline. More complex models could certainly be given; for example, modelling the amount by which the deadline is missed, how many deadlines can be missed in any given period of time, and so on; however, even the simplest probabilistic model provides numerous challenges. I do not relinquish the requirement that all frames must eventually be completed, so this analysis does not consider when or how frames might be dropped. The operating point is also extended probabilistically, as the worst-case time is replaced with a probability distribution over possible execution times; I do not require that there be any maximum value to this pdf. In pursuit of simplicity, I assume that a given operating point requires the same power throughout its execution; therefore, the operating point no longer provides an energy value but a power constant, denoting the instantaneous power of computing at that point. In other words, the energy consumed is assumed to be proportional to time spent, with the constant of proportionality being the power value.

The formal representations of average frame and energy must also be redefined. To this end, let T_i be the random variable denoting the execution time of the ith frame in the period, and let P_i be the power constant of this frame. The time per frame is a random variable defined by the following linear combination:

$$\Psi_T = \sum_{i=1}^{m} \frac{1}{L} T_i \tag{5.14}$$

and likewise the energy per frame is

$$\Psi_E = \sum_{i=1}^{m} \frac{P_i}{L} T_i \tag{5.15}$$

As before, let n_i denote the number of frames in each period executed at operating point i. Finally, the formal probabilistic problem statement can be given.

Given OP, T and p, find $n_1, \ldots, n_m$ such that $\mathbb{E}(\Psi_E)$ is minimised and the following timing constraint is met:

$$\mathbf{Pr}(\Psi_T > T) \le p \tag{5.16}$$

5.6.2 Suitability for practical soft real-time problems

Since deadlines may now be missed, this model can capture so-called soft real-time problems, in addition to the earlier hard real-time problems. In soft real-time, the system is not required to absolutely guarantee each deadline. In fact many real-time problems must be prepared to miss frame deadlines no matter their internal schedule, due to external effects; consider, for example, streaming video over a network with congestion and packet loss, in which the input may not even arrive before the deadline [Kumar and Srivastava, 2001]. This section considers how accurately the description given here can actually capture a practical soft real-time problem such as MP3 decoding.

MP3 frames require a variable amount of work to decode, and this can be described by a probability distribution, which fits the model neatly, although the distribution is not a convenient parametric one. Deadlines occur at regular intervals, governed by the frequency of the sampling (typically 44.1kHz), although frames themselves are not of a fixed size and may contain a variable number of samples. The model of deadline misses given here is, however, less well-suited to audio decoding. If a frame is not decoded in time, a streaming decoder will typically drop the frame completely, since the impact is usually not audible. (Note that the MP3 standard actually requires decoders to be bitstream compliant, meaning that they must produce the same decoding as the reference implementation, so for MP3 processing and transcoding this will not occur; but this is not typically observed by real-time decoders with direct output to the human listener.)

There is a further problem: a simple overall probability is probably not adequate to describe the sort of performance that the decoder must provide. For a human listener, there is clearly a marked difference between a decoder that drops every tenth frame of a song and a decoder that drops the last tenth of the song! In other words, proximity matters; a flat probability is too simple to describe the psychoacoustic properties that MP3 is designed to exploit. However, this is somewhat ameliorated by the periodic nature of the schedules generated here; as described in Section 5.2, operating points can rearranged within each period, so could be permuted *post hoc* to minimise the audible impact of the variability, if this were felt to be a problem at all.

Despite these limitations, the simple model described can provide a reasonable overall approximation to the real problem. Similar arguments can be made for other common examples such as video decoding or graphics processing.

5.6.3 Assumption of normality

For the purpose of this section, I make the strong assumption that the distribution of the frame execution times is normal. Of course, this is unlikely to be true in many real examples. However, there are two important points in defence of the normal distribution.

The first is that it greatly simplifies the algebra shown, while the techniques themselves could equally be applied to any distribution, parametric or non-parametric (at the cost of increased complexity of logic and implementation). The particularly convenient property of the normal distribution is that the sum of two normal distributions is itself normal; to be precise, given independent random variables X and Y with $X \sim \text{Normal}(\mu_X, \sigma_X^2)$ and $Y \sim \text{Normal}(\mu_Y, \sigma_Y^2)$, the distribution of their sum is simply

$$X + Y \sim \text{Normal}(\mu_X + \mu_Y, \sigma_X^2 + \sigma_Y^2) \tag{5.17}$$

The second point in favour of the normal distribution is more profound. In practice, one might wish to avoid changing operating point after every frame; for example, perhaps the assumption that the operating point transition latencies and energy penalties are effectively zero is too strong. One way to address this problem within the current framework is to aggregate several frames into one large “meta-frame”, and schedule each of these meta-frames as if they were single frames. Then, by the Central Limit Theorem, the execution time and energy of the meta-frames is normally distributed no matter the distribution of the underlying frames [Lampert, 1966, pp. 66–67]. Of course this is contingent on aggregating a reasonably large number of frames into each meta-frame. To be precise, let μ and σ^2 be the mean and variance of the frame time distribution; these values must be finite. Let $X_1, \ldots, X_L$ be random variables denoting the execution time of each frame in the L-frame period, so they are all mutually independent by earlier assumption. Then, by the Central Limit Theorem,

$$\frac{1}{\sqrt{L}} \left(\frac{1}{L} \sum_{i=1}^{L} X_i - \mu \right) \xrightarrow{d} \text{Normal}(0, \sigma^2) \tag{5.18}$$

for sufficiently large L, where $\xrightarrow{d}$ denotes convergence in distribution. Therefore the random variable denoting the meta-frame execution time is

$$\sum_{i=1}^{L} X_i \xrightarrow{d} \text{Normal}(L\mu, L^3\sigma^2) \tag{5.19}$$

and an analogous expression can be given for energy. In fact, under the Lyapunov formulation of the Central Limit Theorem, an analogous convergence holds even if the frames are not drawn from the same distribution and have distinct means and variances [Lampert, 1966, p. 69]. This would be the case in, for example, the MPEG scheduler described in Section 5.3. However, this result is subject to certain additional conditions and therefore is not explored further here.

5.6.4 Probabilistic manifestation

This section now proceeds under the assumption of normality. The operating point may then be reified as follows: an operating point is a tuple (μ_i, σ_i, P_i), where a frame at that operating point requires execution time with the distribution $\text{Normal}(\mu_i, \sigma_i^2)$ and energy proportional to its execution time, with the constant of proportionality P_i.

Therefore, the probabilistic operating point set is $OP = \{(\mu_1, \sigma_1, P_1), \ldots, (\mu_m, \sigma_m, P_m)\}$. If the jth point in the period is executed at operating point i, one can simply state that

$$T_j \sim \text{Normal}(\mu_i, \sigma_i^2) \tag{5.20}$$

There are two further standard properties of expectation and variance that are of relevance here [Whittle, 2000, p. 15]. For any constant a,

$$\mathbb{E}(aX) = a\mathbb{E}(X) \tag{5.21}$$

and [Whittle, 2000, p. 22]

$$\mathbf{Var}(aX) = a^2\mathbf{Var}(X) \tag{5.22}$$

The equations of Section 5.6.1 may now be concretised as follows. The average time per frame is

$$\Psi_T \sim \text{Normal}\left(\frac{1}{L}\sum_{i=1}^{m} n_i\mu_i, \frac{1}{L^2}\sum_{i=1}^{m} n_i\sigma_i^2\right) \tag{5.23}$$

The energy per frame is simply the time scaled by the relevant P_i, so the average energy per frame is

$$\Psi_E \sim \text{Normal}\left(\frac{1}{L}\sum_{i=1}^{m} n_i P_i\mu_i, \frac{1}{L^2}\sum_{i=1}^{m} n_i P_i^2\sigma_i^2\right) \tag{5.24}$$

5.6.5 Probabilistic algorithm

One might naturally look to extend Algorithm 1 to handle the probabilistic case. However, the interplay of mean and standard deviation is less amenable to a dynamic programming solution; it appears hard to decide locally whether a point with lower variance and higher

Algorithm 3: Brute-force algorithm to find frequencies which minimise expected energy and meet the probabilistic timing constraint.

Input: Operating points $OP = \{(\mu_1, \sigma_1, E_1), \ldots, (\mu_m, \sigma_m, E_m)\}$, frame time limit T, miss tolerance p and maximum period N.

Output: The minimal energy per frame which satisfies the time limit subject to the given miss tolerance.

```
1  function minEnergyProb($i$, $\lambda$, $\tau_{mean}$, $\tau_{var}$, $\epsilon$)
2     if $\lambda > 0$ and $(T\lambda - \tau_{mean})/\sqrt{\tau_{var}} > \Phi^{-1}(1-p)$ and $\epsilon/\lambda < \epsilon_{opt}$ then
3        $\epsilon_{opt} := \epsilon/\lambda$;
4     if $i \leq m$ then
5        foreach $n_i \in \{0, \ldots, N - \lambda\}$ do
6           $\lambda' := \lambda + n_i$;
7           $\tau'_{mean} := \tau_{mean} + n_i\mu_i$;
8           $\tau'_{var} := \tau_{var} + n_i\sigma_i^2$;
9           $\epsilon' := \epsilon + n_i\mu_i P_i$;
10          minEnergyProb($i+1, \lambda', \tau'_{mean}, \tau'_{var}, \epsilon'$);

11 $\epsilon_{opt} := \infty$;
12 minEnergyProb(1, 0, 0, 0, 0);
13 return $\epsilon_{opt}$;
```

mean is globally preferable to one with higher variance and lower mean. Without discounting the possibility of a dynamic solution, I turn to a simple exhaustive search, presented in Algorithm 3: a basic depth-first search on frame counts. The arguments to the recursion are:

- i, the index of the next frame to be considered. I assume frames are indexed from 1 to m.
- λ, the total number of frames accumulated so far, in other words

$$\lambda = \sum_{j=1}^{i-1} n_j \tag{5.25}$$

  The constraint of Equation 5.6 is enforced by the loop limits of line 5.
- τ_{mean} and τ_{var}, the partial sums for the mean and variance of the time distribution so far, analogous to Equation 5.25. Therefore, at the bottom of the recursion, τ_{mean}/λ and τ_{var}/λ are the mean and variance respectively of Ψ_T.
- ϵ, the sum of the expected energies so far. Hence ϵ/λ is the partial sum for the expectation of Ψ_E.

The recursion then explores all possible assignments of n_i values, subject to Equation 5.6. Line 11 initialises the mutable global variable ϵ_{opt}, which holds the lowest average energy seen so far and ultimately the solution to the problem; the values of n_i required to produce ϵ_{opt} are readily available from the recursion so these can also be returned by a practical implementation. Finally, line 12 invokes the base case of the recursion with the obvious values and line 13 returns the algorithm's output.

One non-trivial component of the algorithm presented is the condition

$$(T\lambda - \tau_{mean})/\sqrt{\tau_{var}} > \Phi^{-1}(1-p) \tag{5.26}$$

on line 2. This is an alternative formulation of Equation 5.16, where Φ is the standard normal cumulative distribution function (cdf).

Proof. Let Z be a random variable with standard normal distribution. The probabilistic time constraint is

$$\mathbf{Pr}(\Psi_T > T) \leq p$$

so, reversing the signs,

$$\mathbf{Pr}(\Psi_T < T) \geq 1 - p$$

and substituting the parametrised normal for a standard normal gives

$$\mathbf{Pr}\left(Z < \frac{T - \tau_{mean}/\lambda}{\sqrt{\tau_{var}}/\lambda}\right) \geq 1 - p$$

Then multiply the inner expression through by λ and introduce Φ to give

$$\Phi\left(\frac{T\lambda - \tau_{mean}}{\sqrt{\tau_{var}}}\right) \geq 1 - p$$

Finally, using the monotonicity of Φ, this can be inverted to give Equation 5.26 as required. □

The advantage of this formulation is that $\Phi^{-1}(1-p)$, which is relatively expensive to calculate, is independent of any recursive variables, so can be computed and stored during initialisation.

One can also apply some simple branch-and-bound heuristics to the search but for the sake of simplicity these are not shown.

5.6.6 Generalised domination

The dominated point heuristic from Section 5.4.1 can be transferred to the probabilistic case, although because the problem is more complex the optimisation is weaker. In order for the earlier cut-and-paste argument to apply, it is necessary to show that a point Y can be replaced in any context with another point X with the resulting schedule being at least as likely to make any deadline, and using less (or equal) energy on average. Let X and Y be the probabilistic operating points (μ_X, σ_X, P_X) and (μ_X, σ_X, P_X) respectively. Then let T_X and T_Y be the time distributions for X and Y respectively, so that $T_X \sim N(\mu_X, \sigma_X^2)$ and $T_Y \sim N(\mu_Y, \sigma_Y^2)$. Then a sufficient condition for X to dominate Y is for the following equations to hold:

$$\mu_X \leq \mu_Y \tag{5.27}$$

and

$$\sigma_X \leq \sigma_Y \tag{5.28}$$

and

$$\mu_X P_X \leq \mu_Y P_Y \tag{5.29}$$

Proof. By Equation 5.27, for any t,

$$t - \mu_X \geq t - \mu_Y \tag{5.30}$$

Dividing through by σ_X (since $\sigma_X > 0$) and using Equation 5.28, gives

$$\frac{t - \mu_X}{\sigma_X} \geq \frac{t - \mu_Y}{\sigma_X} \geq \frac{t - \mu_Y}{\sigma_Y} \tag{5.31}$$

Now take the left and right side of this inequality and substitute the expressions into the cdf for a standard normal Z, using the monotonicity of Φ, to give

$$\Phi\left(\frac{t - \mu_X}{\sigma_X}\right) \geq \Phi\left(\frac{t - \mu_Y}{\sigma_Y}\right) \tag{5.32}$$

Expanding the definition of Φ gives

$$\mathbf{Pr}\left(Z < \frac{t - \mu_X}{\sigma_X}\right) \geq \mathbf{Pr}\left(Z < \frac{t - \mu_Y}{\sigma_Y}\right) \tag{5.33}$$

and so, rearranging,

$$\mathbf{Pr}(Z\sigma_X + \mu_X < t) \geq \mathbf{Pr}(Z\sigma_Y + \mu_Y < t) \tag{5.34}$$

Now $(Z\sigma_X + \mu_X)$ is an identical distribution to T_X and likewise for T_Y, so

$$\mathbf{Pr}(T_X < t) \geq \mathbf{Pr}(T_Y < t) \tag{5.35}$$

and this establishes the condition for maintaining the deadline probability.

The reduction in expected energy follows directly from Equation 5.29. Let E_X and E_Y be random variables denoting the energy of executing a frame at point X and Y respectively; then

$$\mathbb{E}(E_X) = \mu_X P_x \leq \mu_Y P_y = \mathbb{E}(E_Y) \tag{5.36}$$

Hence X is at least as likely to make any deadline, and has lower or equal expected energy consumption. □

Detection of dominated points is now the 3-dimensional vector maximum problem, over the 3-tuples $(\mu_X, \sigma_X, \mu_X P_x)$, and this can be solved in $O(n \lg n)$ time [Kung et al., 1975]. For simplicity, my implementation uses the naïve quadratic-time algorithm, an exhaustive comparison of every pair of points for domination, since this phase of the algorithm requires so little time compared to the exhaustive search; Amdahl's Law clearly mitigates against much optimisation in this area [Amdahl, 1967].

5.6.7 Probabilistic evaluation

Once again I measure the performance of the dithering algorithm against a naïve static algorithm and a greedy dynamic algorithm. The principles of the compared algorithms are analogous to those in Section 5.5; both select the lowest-energy point from those with $\mathbf{Pr}(\text{time} > T) < p$. For the greedy algorithm, I again simulate 10 000 iterations and allow slack time to accumulate, so that the deadline may extend or contract with

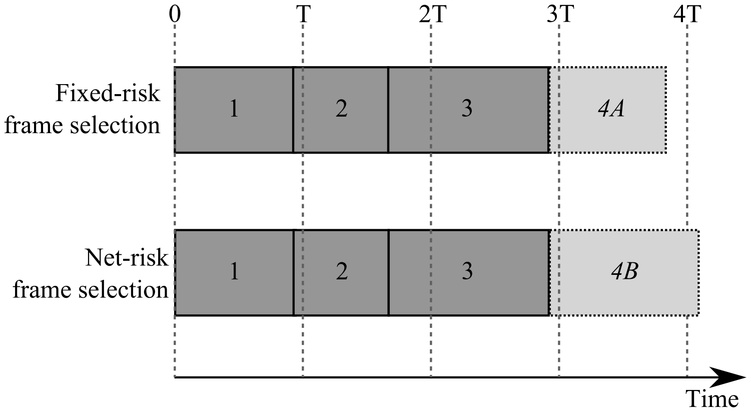

Figure 5.10: Diagram showing the variable deadline encountered by the dynamic frame scheduling algorithm.

successive iterations. One complication is that, if the algorithm takes too many risks or encounters a run of unexpectedly long frames, there may be no operating points fast enough to meet the next deadline, leaving the algorithm to select from an empty set. Since the normal distribution has infinite support, no schedule can ever eliminate this possibility completely. I consider this the "panic case", and respond by selecting the point with minimal $\mathbf{Pr}(\text{time} > T)$ regardless of its energy demands.

There is one important complication to consider, which can be described as follows. After a series of fast frames, the dynamic algorithm may accumulate enough slack to move to a slower operating point, so in effect the value of T may vary from iteration to iteration. However, the effective value of p does not vary; even if the algorithm meets a large number of deadlines in a row, the next frame selected must have a probability of missing the deadline less than p, even though overall the algorithm could be much less cautious. Consequently, the dynamic algorithm chooses conservatively at each iteration, and the largest admissible value of $\mathbf{Pr}(\text{time} > T)$ amongst the available operating points may still be markedly smaller than p, leading to substantial inefficiency. Experimentally, the algorithm can be seen to miss deadlines at frequencies between $0.5p$ and $0.01p$; in other words, the algorithm was missing the deadline between 1% and 50% of the maximum permissible frequency, and therefore presumably working rather harder than necessary. By contrast, the static algorithm can routinely achieve $0.98p$. In an attempt to combat this, I consider an alteration which allows more risks to be taken if past performance has been more than adequate (and *vice versa*). Let I be the total number of iterations so far, and let D be the number of deadline misses. Then replace the selection constraint with the requirement that

$$\frac{\mathbf{Pr}(\text{time} > T) + D}{I + 1} \leq p \tag{5.37}$$

In other words, the expected number of errors after this iteration, including all previous errors, must not exceed the threshold fraction. Figure 5.10 illustrates the contrast with a low deadline threshold such as $p = 0.8$. The darker boxes 1, 2, and 3 show the execution times of past frames, each of which has met its deadline. The two algorithms differ in their selection of a suitable operating point for the fourth frame. The fixed-risk approach (top) requires an average time like that of box 4A, since the fourth frame must meet its deadline with at least 80% probability. However, the net-risk approach (bottom) can

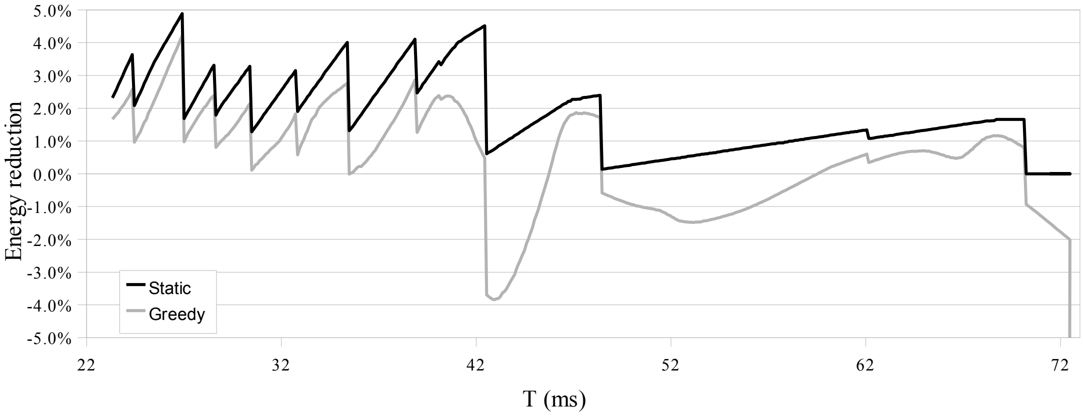

Figure 5.11: Comparison of the energy reduction achieved by my static algorithm and a dynamic greedy algorithm for normally distributed execution times.

tolerate a longer average time like that of box 4B. This frame may then meet its individual deadline with less than 80% probability, but since the previous three frames already met their deadlines, the algorithm still respects the overall 80% success rate. If 4B has a lower expected energy, as it typically will, this is the superior choice; if not, the algorithm is still free to select 4A.

In practice, this refinement was found to be largely unsuccessful. It seems that the algorithm constantly pushes the threshold of maximum risk, and thereby triggers the panic case far more often. The energy penalty for panicking usually exceeds the gains from exploiting the extra risk by a considerable margin. Nevertheless, to give my algorithm the most stringent test, both versions of the greedy algorithm were run for every case, and the canonical value taken to be the lower energy of the two.

To provide suitable test data, I again made use of the BSOM data to provide the μ_i and implied P_i values, as shown in Figure 5.7. Unfortunately the published data did not include any way to estimate σ_i^2 values. Since it is not possible to know the sort of σ_i values that might be seen in practice, I instead provide several result sets with ersatz standard deviations generated in different ways:

- In result set A, all σ_i values are set to the same constant value.
- In result set B, each σ_i is given as a fixed fraction of the relevant μ_i.
- In result set C, each σ_i is given a value selected uniformly at random from a chosen range.

These result sets are intended to span a reasonable range of possibilities for the actual characteristics of σ_i, which are unknown.

As an example, set each σ_i to a nominal value of 1, and take $p = 0.1$. Figure 5.11 shows the comparison behaviour of the two algorithms for a range of T values. Some of these percentages are now negative, indicating that the naïve approach actually does better in some cases than the dynamic one. The average distance between the greedy and static approaches is now 1.4%, with the static approach reaching 4.7% improvement in some cases.

Set	**p**	$\sigma_\mathbf{i}$	$\mathbf{A_{static}}$	$\mathbf{A_{dynamic}}$	$\mathbf{B_{static}}$	$\mathbf{B_{dynamic}}$	$\mathbf{Diff}_A$	$\mathbf{Diff}_B$
A	0.001	1.0	2.0%	0.7%	45.6%	6.8%	1.3%	38.8%
	0.01	1.0	1.8%	0.4%	43.8%	4.7%	1.4%	39.2%
	0.1	1.0	1.6%	0.1%	41.4%	1.3%	1.4%	40.2%
	0.001	0.1	1.2%	-0.3%	37.7%	-3.3%	1.5%	41.0%
	0.01	0.1	1.1%	-0.4%	37.6%	-3.5%	1.5%	41.1%
	0.1	0.1	1.1%	-0.4%	37.5%	-3.7%	1.5%	41.2%
B	0.001	$\mu_i/10$	4.0%	4.0%	59.9%	54.4%	0.1%	5.5%
	0.01	$\mu_i/10$	3.4%	3.2%	55.8%	48.1%	0.2%	7.6%
	0.1	$\mu_i/10$	1.2%	0.9%	38.6%	27.7%	0.3%	10.9%
	0.001	$\mu_i/100$	1.4%	1.1%	40.1%	29.5%	0.3%	10.5%
	0.01	$\mu_i/100$	1.3%	1.0%	38.8%	28.0%	0.3%	10.8%
	0.1	$\mu_i/100$	1.2%	0.9%	38.6%	27.7%	0.3%	10.9%
C	0.001	$U(0.1, 1)$	1.8%	-14.5%	42.1%	-562.1%	16.3%	604.2%
	0.01	$U(0.1, 1)$	1.5%	-9.4%	41.2%	-338.8%	10.9%	380.0%
	0.1	$U(0.1, 1)$	1.3%	-0.8%	39.9%	-35.4%	2.2%	75.3%

Table 5.3: Power reductions measured for the BSOM benchmark. For meaning of column headings, see text.

Table 5.3 shows results for other values of p and various methods of generating σ_i. I define A_{static} and $A_{dynamic}$ to be the percentage energy reduction of the static and dynamic algorithms respectively compared to the naïve algorithm. These numbers tend to be small because there is a certain minimum energy that no schedule can go below, determined by $\min_i(\mu_i P_i)$, and the amount of optimisation headroom between this value and the naïve algorithm's energy value may be small. Therefore I also define B_{static} and $B_{dynamic}$ by subtracting out this minimum value from the ratio, normalising for the dynamic range of the available operating points; in other words,

$$B_{static} = \frac{\text{naïve energy - static algorithm energy}}{\text{naïve energy } - \min_i(\mu_i P_i)} \tag{5.38}$$

and $B_{dynamic}$ likewise. All quantities can be negative if the naïve algorithm performs better. The columns Diff_A and Diff_B show the average distance between the performance of the two algorithms, unnormalised and normalised respectively, over the full range of T. I experiment with a range of σ_i values; some constant, some scaled according to the mean time, and some generated uniformly at random over the specified range. Evidently the dynamic algorithm performs markedly better for all choices of parameter, although the margin of average differential varies over two orders of magnitude.

5.6.8 Probabilistic summary

As shown in Table 5.3, my static algorithm can outperform the example dynamic algorithm in the presence of runtime volatility; this is largely attributable to the better handling of risk that can be achieved by taking a long-term view. The static algorithm typically finds a solution with an error rate only fractionally below the allowable maximum, while the dynamic algorithm tends to oscillate with the short-term fluctuations of statistical chance.

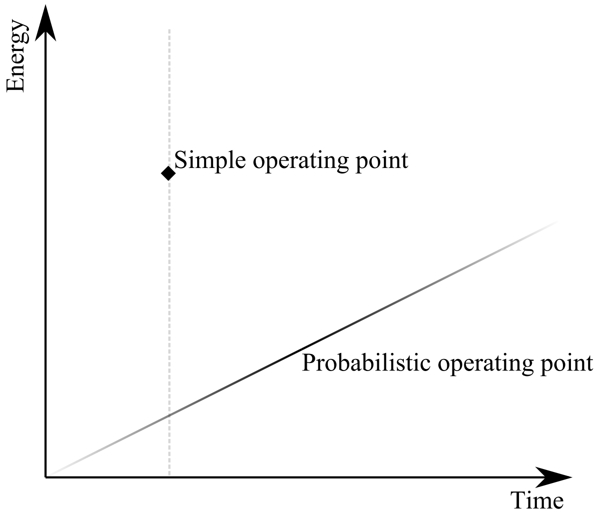

Figure 5.12: Two different forms of operating point energy–time description, represented graphically.

Interestingly, there is a significant disparity in the variety of normalised differentials (the Diff_B column) between result sets; in set A, the differential is around 40% irrespective of p and σ_i; for set B, both p and σ_i affect the norm but by a factor of less than two; for set C, the results vary by a factor of more than eight. This suggests that it is more important to characterise the general manner in which σ_i is related to the other variables of the operating point than to discover the exact values. This presents a direction for future work.

5.7 Conclusion

The first major contribution of this chapter is the introduction of the operating point, a concept that abstracts away the complex web of interactions between hardware, software, and computational energy, and replaces it with a clean interface to the relevant data for a stream scheduler. A concrete operating point can take different forms depending on the demands placed upon it; this chapter has explored two particular cases. The first is perhaps the simplest possible manifestation: an expected energy consumption and a worst-case execution time. The second is, analogously, probably the simplest possible characterisation for the probabilistic case: the parameters of a normal distribution over execution time, and a power constant. Both cases can be seen as describing certain points in the energy–time diagram; Figure 5.12 generalises Figure 5.3 to illustrate the point. The first case is simply a point in energy–time space, and states only that the actual time will occur somewhere to the left of the dashed line; it says little about what the energy might be, except that it has a given mean. In the particular reification chosen for the probabilistic case, an operating point describes a line through energy–time space, passing through the origin, and with the gradient implied by the power constant; any given execution occurs somewhere on this line, with probability determined by the mean and standard deviation of the underlying normal. These are just two of the many forms that an operating point could take on.

Another key concept in this chapter is that operating points may be dithered to produce more favourable energy outcomes than any individual operating point can offer. Just as the operating point generalises configuration variables such as the operating voltage, operating point dithering generalises and subsumes techniques such as voltage dithering. In both the deterministic and probabilistic cases, the empirical evidence of this chapter suggests that this approach does actually provide such energy reductions on real data; furthermore, it shows that a static algorithm can often get much closer to the ideal deadline-free case than a dynamic approach with the same information, and with less chaotic behaviour with respect to the deadline.

In terms of the algorithmic mechanism for calculating these dithered operating point schedules, there are some important differences between the two forms of operating point considered here. The deterministic case is shown to have a deep connection to the famed "integer knapsack problem", one of Karp's canonical examples of NP-completeness [Karp, 1972]. This leads naturally to an efficient dynamic programming solution, and suggests several heuristics that might be translated to the problem at hand; this translation was actually performed for single-point domination. The introduction of N, an artificial limit on the length of the period in the frame cycle, presents a technical blemish, but the results here show that it is not a significant problem in practice. Aside from this minor defect, the problem can solved cleanly and efficiently. The probabilistic case, even in the greatly simplified presentation given here, proves much more difficult to solve, and I have not been able to devise a better method than basic branch-and-bound exhaustive search. Furthermore, the heuristic mechanisms for the deterministic case allow examples with large numbers of operating points to be solved; this is important for realistic examples, which may have many axes along which operating points can be generated. In the probabilistic case, this is simply impossible with the implementation presented, since the exponential time requirement of the algorithm is overwhelming even in an offline context. A key challenge for future work will be developing algorithms to handle these problems, perhaps in the direction of bounded-error polynomial-time approximation. It would also be particularly desirable to handle general (even non-parametric) probability distributions, since the requirement for normally-distributed execution times is, despite its advantages, rather limiting.

This chapter opened with a definition of streaming computation that I believe embraces most of the current informal uses of the term, and continued by developing methods to improve energy efficiency for computations within this definition. This chapter therefore sits in opposition to the previous: streaming is, perhaps, the opposite facet of the energy problem to the long-running computations considered in Chapter 4. The contrasts are many. Stream computations are generally at fine granularity, as each unit of work is typically small, while the computation as a whole may be infinite; the previous chapter, by contrast, focused on large, long-running tasks and finite computation. Furthermore, as formulated here, streaming computation may include a probabilistic element, since new tasks arrive in real time and the size of the workload they require is governed by a probability distribution. The previous chapter, on the other hand, focused exclusively on the deterministic case: workloads with a fixed size known from the beginning. The two forms of computation are also encountered at different stages of the same service; compare, for example, Google's PageRank computation (essentially a huge, long-running matrix multiplication) with the fast Web-driven search queries that are executed against the PageRank data in a streaming fashion [Page et al., 1999]. The two chapters consequently provide a neat bracketing of the energy efficiency spectrum, again illustrating the value of considering the "energy problem" in the broadest possible context.

Chapter 6

Conclusions

Our understanding of the totality of the energy problem is still in its infancy. This thesis has taken an overview of the practical situation and, on the other hand, produced concrete contributions in specific cases by largely mathematical means. In this final chapter, I synthesise the work of individual chapters into a single concluding argument, and show how the thesis leads naturally to certain areas of future work.

6.1 Concluding argument

This thesis has argued that energy efficiency is a diverse problem and that it is necessary to understand the whole picture in order to make progress in specific areas, especially if we seek a unified “theory of energy”. In this section I draw together the arguments of previous chapters to show how, taken together, they demonstrate the central thesis.

Chapter 3, in presenting arguments for and routes towards energy-aware computing, demonstrated the pressing nature of energy awareness in modern computing, and identified the swelling consumption of energy by servers as particularly problematic. Provision of adequate power resources to a modern data centre was shown to be a significant logistical challenge, as was the compounding cost of cooling and other site infrastructure. This established that practical energy-efficiency measures must be taken. This view from industry then motivates the work of Chapter 4, which developed digital voltage–frequency scaling (DVFS) techniques for long-running computations as might be undertaken in a data centre. Following the observation that server management is primarily motivated by economic and not ecological concerns, I proposed the novel idea of *energy cost optimisation* as a generalisation of traditional energy optimisation, and described a suitable mathematical framework. I further showed that the structure of the electrical grid—in particular, its lack of storage capacity—leads to the surprising result that using more total energy may be “greener” as well as more financially sound, and that the energy cost model can capture this important consideration. I established that the earlier cost-agnostic results, such as the DVFS “energy-efficient frequency”, arise as special cases in this framework. I also derived formal results for particular cost functions, and developed some general techniques for solving other classes of cost function analytically. Further, this chapter measured the efficiency gains of these analytic solutions against simpler heuristic approaches for some realistic parameter sets; this is important because past work, while demonstrating its mathematical utility, has often stopped short of providing precise numbers by which to compare it to existing approaches. However, the chapter also recognised the limitations of analytic solutions, and described situations in which only a numerical method is reasonable. Overall I determined that this approach has the potential to be deployed successfully in practice.

To return to an earlier point in the argument, Chapter 3 also characterised the sort of constraints that a practical energy efficiency methodology must respect, with regards to performance and responsiveness, and outlined the current mechanisms by which these may be executed (such as ACPI). In particular, it described the tickless kernel project, which exemplifies the central thesis: a broad understanding of the problem (kernel non-quiescence) driving concrete, specific improvements (new kernel APIs and an array of kernel- and user-space improvements). The end-user sees a substantial improvement in battery life for only a fractional drop in performance. I concluded that opportunities for improvements of this form will become more commonplace as the subtlety of the underlying hardware characteristics increases, and often can only be seen by consideration of the full system stack.

To illustrate the generality of the central argument, Chapter 5 concerned a complementary area of the energy efficiency problem: streaming, particularly on a mobile device. The contrast was illustrated by comparing the continuous, infinite-streaming model of computation used in this chapter with the traditional discrete-workload problem of the previous. Motivated by the wider view of mobile platforms as hugely diverse in terms of hardware, operating system, network capabilities and so forth, I then defined and defended the concept of an operating point, an abstraction of hardware configuration and performance that allows energy-efficient schedules to be derived without becoming tangled in the details of implementation. I argued that more efficient solutions to streaming problems can be found by a generalisation of voltage dithering to the level of operating points. The chapter then framed the problem formally and presented algorithms to solve it, efficiently in the deterministic case and more exhaustively in the stochastic case. Analogies with the integer knapsack problem allowed further heuristics to be discovered, increasing the applicability of the algorithm to large sets of operating points as might be seen on a modern mobile phone or tablet. Again, empirical data showed the gains that can be made with these methods compared to the greedy reactive methods that are typically used in practice, and demonstrated that the novel methods are sufficiently superior that they could be applicable in practice.

To summarise, this thesis has established the following:

- Energy awareness in computing is important for reasons of economy, ecology, and logistics.
- Energy cost is superior to raw energy as a metric of optimisation.
- The operating point is a valuable abstraction of execution details, shifting the focus to higher-level energy optimisation algorithms.
- Formal methods can produce substantial, measurable savings compared to simpler heuristic methods.

6.2 Future directions

My central thesis, presented in Section 1.1, suggests the broad direction that future work on this problem might take: we should continue to develop our understanding of individual areas of the energy problem while looking for opportunities for unification in pursuit of a complete theory of energy.

There are two strands of future work that would arise naturally from this thesis. The first is to pursue the grand vision of the unified theory of energy suggested in Chapter 1, and the second is to explore the more specific lines of future work from the individual chapters.

6.2.1 Future work on the theory of energy-efficient computing

On the surface this goal seems rather abstract, and it is not clear how to approach it; as Taleb says, “in the real world one has to guess the problem more than the solution”.[1] A more concrete question is to ask whether it would be possible to develop an operating system and software stack that could scale proportionally, in both performance and power terms, from the most heavyweight computing devices to the simplest devices capable of general purpose computing; spaces currently occupied, at least in mass production, by high-end data centre machines and mobile phones respectively. Turing-completeness assures us that the programs from one end of this scale can in principle be run on the other, but offers no guarantee that this scaling would be proportional to the energy cost incurred. (One might even ask whether such a system could scale to non-general purpose devices, such as wristwatches, but in the absence of Turing completeness the question becomes more limited.) Perhaps Linux is the closest approximation to such an OS that currently exists, but clearly the forms of Linux run on high-end server machines are materially different to mobile-targeted distributions such as Google Android and Kubuntu Mobile. In the software stack the discrepancy is even greater; one could not expect to run a MySQL database server from a phone in an energy-efficient way, nor *Angry Birds* on a high-end server. The sort of behaviours that are energy-efficient or energy-inefficient are too different between the platforms; for example, polling on a mobile device prevents the CPU entering an idle state (as seen in the discussion of the tickless kernel) and therefore is a substantial battery drain, while on a busy server it incurs little more overhead than a context switch. The examples of database servers and video games might seem frivolous, but between them lie many plausible ideas which are currently rendered unworkable by the impossibility of scaling the software to the hardware; for example, perhaps it would be desirable to run a temporary web server from a mobile phone, and to make use of the highly refined existing web server software rather than building a customised low-power implementation for the platform. This is not currently possible because existing programming abstractions, while providing suitable interfaces to other hardware features such as available memory or multi-processing capability, do not provide any equivalent for energy-awareness, but rather abstract it away altogether. Energy consumption is invisible to the programmer but all too visible to the end-user.

I do not believe it would be possible to unify all scales of all devices simultaneously; the problem is simply too vast. A more promising direction is already seen in the merging of operating systems for similar classes of hardware: for example, Windows NT 6.0 as the underlying architecture for the workstation-targeted Windows Vista and server-targeted Windows Server 2008. Likewise, Google's Android and Apple's iOS target both mobile phones and tablets simultaneously. It appears the boundary between mobile and mains-powered devices is currently the hardest to cross, and perhaps new challenges will emerge as new points in the device space come into existence (as occurred, for example, with Apple's creation of the tablet market almost overnight). At the software level, application programmers will perhaps need greater access to the power characteristics of the underlying machine, and even a suitable description of these characteristics is not yet established. Perhaps intelligent runtime and compile-time systems may also have a part to play. It is not yet known how this might be done, and clearly this is a key challenge for the future.

---

[1] From Nassim Nicholas Taleb, *Fooled by Randomness*, p. *x*, Random House Publishing, 2005.

6.2.2 Future work on cost-efficient computing

Chapter 4 describes a model of the relationship between power and performance, and develops methods to optimise energy cost under this model. Most obviously, future work could look at deploying these techniques in practice; perhaps a multi-user system could be developed in which each user has a budget and can specify the deadline and workload they require for each task in order to meet this budget. Cluster computing frequently provides an interface to relevant constraints (such as time and space), but present implementations do not generally consider energy to be a relevant resource.

Inevitably, both power and cost models could be further generalised. Of particular practical relevance would be the exploitation of parallelism, since this opens new avenues for energy efficiency without sacrificing performance [Cho and Melhem, 2008]. The approach given here might be extended to include this, especially with support for fine-grained power control for individual cores. One could also look at extending some of the inter-machine coordination work described in Section 2.4.2 to support cost models. Support for soft deadlines, with some penalty for overshooting, might also be of practical benefit. A reasonably efficient online algorithm for a more general case has been proposed, and could be extend to embrace the variable cost approach [Yao et al., 1995]. Algebraic solutions become increasingly unlikely with each extra layer of complexity but, if current predictions for the future of computer architecture are to be believed, would be highly beneficial.

6.2.3 Future work on energy-efficient streaming

Chapter 5 developed techniques for selecting operating points to process streams in an energy-efficient way. This was tested on simulated operating point data, so a natural next step would be to deploy these techniques on a real mobile device and measure the resulting energy values. In addition to its obvious primary purpose of testing the theoretical work in a practical context, this experiment would also serve to suggest directions in which the model could be improved. For example, perhaps the constant power assumption that the execution energy scales linearly with the execution time—in other words, that power is independent of the input data and execution pathway taken—is too restrictive. These questions can only be answered by empirical data.

The work of Chapter 5 progressed by successively weakening its assumptions. First, I discarded the assumption that all frames must be processed at a single operating point, which leads to the concept of operating point dithering. Then I removed the restriction of describing frame execution only by its worst-case, and allowed a fuller description in terms of probability distributions. A natural question is then to ask what further restrictions might be removed. In fact there are many. In particular the *inputs* to the algorithm are rather demanding, requiring a detailed understanding of the workload and operating point characteristics. Decreasing these demands would be a valuable direction for future work; some detailed suggestions on future work in this area are presented in Appendix A.

Appendix A

Future work: online learning in hard real-time

A.1 Introduction

This appendix describes what I believe to be an interesting direction for future work, and some of the key challenges involved in this area. It assumes a degree of familiarity with various topics outside the scope of this thesis, since there is not space here to introduce them all from first principles; rather than consulting specific citations, the reader is referred to the general background in Section A.6.

The progression of this thesis, especially in Chapter 5, has been to throw off successive layers of assumption; first assuming that tasks require their worst-case execution time, then relaxing this to include execution times with a normal distribution, then making a brief exploration of more general distributions. This could certainly be pushed further. A relevant research question might be: what could be achieved if one made no *a priori* assumptions about the workload or the machine on which it is executed?

A.2 Motivation

The future of hardware manufacturing is surely towards greater inter-device variation. In a processor with, say, one thousand cores, the machine may well boot up to discover that a few cores have perished for some reason, and it is perfectly reasonable to expect that such a machine could continue to function with only a proportional drop in performance. Performance degradation may continue to occur over the lifetime of the machine. If only a few cores are damaged, it is probably uneconomical to replace the processor completely. On the other hand, for some specialised situations, such as deep space missions or medical nanorobotics, it may be impossible for any outside agent to repair the system, and the system should simply attempt to function (at a reduced level) for as long as possible.

Therefore it would be highly desirable for such a system to have the ability to discover its own capabilities, in terms of power and performance, in an online and on-going way. One can certainly imagine a new mobile phone learning about the particular point in the power-performance spectrum into which it has been "born", given the natural variation between components. To push the idea further, perhaps there is a connection with the field of *autonomous robotics*; in particular, the area of self-discovery, in which a robot may for example be asked to discover, without direct programmer intervention, that it has hands, and that hands are useful in the world.

Hard real-time, in which the system must absolutely guarantee to complete each task before its specified deadline, is not often associated with energy efficiency. It is usually assumed that if a system's deadlines are worthy of being designated hard then we are willing to expend any amount of power to meet them. However, there are several reasons that this is not necessarily true. Firstly, energy is never free and, on a sufficient scale, even mains electricity has a non-negligible impact on the total cost of ownership. Secondly, such systems may be forced onto battery power at some point, either as a matter of course or in the case of mains power failure, and battery power is a valuable and finite commodity. Thirdly, we would like such systems to proliferate as widely as possible, and constraints based on energy might well be limiting in this respect. And fourthly, many soft real-time systems could provide strong Quality of Service guarantees if they were internally constrained as hard real-time; consider, for example, a portable video player guaranteed never to skip a frame while still providing reasonable battery life. Therefore we consider it worthwhile to explore ways in which hard real-time streaming might be made more energy efficient.

There are several key problems here: we do not necessarily know a great deal about the workload in advance, especially for increasingly general-purpose computing devices such as mobile phones; modern devices often provide numerous hardware variables that can be adjusted to trade performance for power efficiency, such as voltage scaling, clock gating, cache disabling and so on, and the benefit or otherwise of these trade-offs cannot necessarily be predicted statically; and at the software level, there may be various methods for handling each frame, perhaps with varying levels of hardware support (for example, use of a co-processor). These factors taken together demonstrate that is generally impossible to give offline predictions for the power and performance of every configuration. Therefore, we would prefer to use some kind of online machine learning in order to deduce these values and scale the performance of the system appropriately, such that energy usage is reduced while still guaranteeing all deadlines.

Of course, online learning in a hard real-time context is difficult because we are constrained in our ability to explore the state-space. I propose handling this by *online learning in the accumulated slack time*. What this really means is described below.

A.3 Problem outline

In this section, I outline a particularly simple formulation of the problem. Certainly there are many directions in which this could be generalised, but this presentation crystallises the basic idea. Assume, as in Chapter 5, that the task is to process a stream of discrete frames. Each frame requires a finite amount of work, although the stream may contain an infinite number of frames.

From the hardware side, assume a set of "operating points" OP, as described in Section 5.1.1. Each frame must be executed at a given operating point; in other words, one cannot process half of a frame at one point and the second half at another. This restriction alleviates concerns about which inter-operating point transitions are permitted and how they might be effected. However, a frame can, at any time, be restarted and processed at a new operating point, which has the embedded assumption that processing is side-effect free until completion.

Each frame must be completed by its associated deadline, and these deadlines occur with a fixed period T, so that frame i must be processed before time iT. As in Chapter 5, data dependencies may exist between the workloads, and they must therefore be completed in the order presented. One particular operating point $op^* \in OP$ is, by definition, fast enough to compute any frame in less than time T; such a point must be known if a hard real-time guarantee is to be made. Informally, one might say something like

$$\mathbb{P}(\text{execution time at } op^* > T) = 0 \tag{A.1}$$

although this probability is yet to be defined precisely.

The aim, as ever, is to process the frames such that every deadline is met and overall energy is minimised.

A.4 Proposed methodology

The guarantee of A.1 is, of course, driven by the worst-case execution time (WCET) of a frame. Real execution times will tend to be faster, and thereby the system will accumulate slack time with each successive frame. Therefore, the system should execute frames at op^* until there is sufficient slack before the next deadline; the meaning of “sufficient” is yet to be determined, but for the sake of argument, let us say there is $3T$ time until the next deadline. At this point, the system moves to another point $op_1 \in OP$—presumably offering lower power and reduced performance—and attempts to process the next frame. If the frame is completed in less than $2T$ time, the system learns something about the energy and performance characteristic of op_1, and moves on to the next frame, making a new decision about which operating point to select. On the other hand, if the frame has not completed once time $2T$ has elapsed, the system must abandon the computation at op_1 and restart the frame at op^*, thereby maintaining the hard real-time guarantee. The process is repeated each time adequate slack is accumulated, with the system constantly improving its knowledge of the characteristics of the operating points. Ultimately it is able to make highly efficient decisions about whether operating points can be usefully employed.

A.5 Challenges

This is by no means the first suggestion to use slack reclamation and online learning to drive energy efficiency. However, there is novelty in the work in terms of its application to *hard* real-time and in the high-dimensional learning implied by the use of operating points (as opposed to simple processor DVFS as has usually been deployed in the past).

This simple methodology conceals a host of challenges. On the surface, this problem recalls the canonical multi-armed bandit problem: given a selection of levers that produce rewards according to some unknown distribution, how should we select which levers to pull to maximise the long-term payoff? This is usually formulated as an explore-exploit dilemma and is widely studied as such. Several good approaches for this problem are known. However, there are some important differences in this situation:

- Neither of the measured quantities (time and energy) are rewards as such; rather it is desirable to know about them in order to gain some more abstract reward (energy efficiency).
- Relatedly, no single point is ever adjudged the "best". Even if one did have complete knowledge of the distributions, the ideal point varies depending on the amount of available slack time. Perhaps assuming this knowledge would be one way to approach a simplified formulation of the problem.
- Observations may be *right-censored*: in the case where the explorative operating point is aborted, the system learns only that the time and energy required for a given frame was *at least* some value. To make use of this information requires more complex learning techniques. Perhaps a relevant technique could be found in survival analysis, a technique from medical statistics that is used to analyse data in which subjects may disappear or drop out of a study midway through. The Kaplan-Meier estimator, for example, allows non-parametric learning of probability distributions with right-censored data.
- The result of one lever affects our ability to explore other levers; there is feedback between the success of exploitation and the ability to explore.

Therefore, the traditional explore-exploit dilemma is inadequate to fully capture this problem, and new work is needed. I suspect it would be very challenging to develop a "zero-regret" strategy for this situation, which is the typical goal in explore-exploit dilemmas.

Note that I have not restricted consideration to finite OP sets. If one considers continuously variable operating voltages, OP naturally becomes an infinite set, and this further increases the complexity of the learning required. I leave it to future work to decide whether this added complexity is justified by the benefits to model fidelity.

Additionally, I have so far presented OP as an unstructured set. In reality, there is typically at least some additional information that may aid learning. For example, one could quite readily construct some sort of partial order of power-performance over the set; it would generally be safe to assume that, say, disabling a cache would reduce (or at least maintain) the system power and increase (or at least maintain) the system latency. A suitable Hasse diagram could be provided as input to the learning algorithm. How exactly this might be done is left to future work.

A.6 Related work

The interested reader is encouraged to review the following articles and books, and other work by these authors.

- **Multi-armed bandit problem**

  - *Multi-armed bandit algorithms and empirical evaluation*, J. Vermorel and M. Mohri.
  - *Competing in the dark: an efficient algorithm for bandit linear optimization*, J. Abernethy, E. Hazan and A. Rakhlin.

- **Explore-exploit dilemmas**
  - *Explore/exploit strategies in autonomy*, S. W. Wilson.
  - *An autonomous explore/exploit strategy*, A. McMahon, D. Scott, and W. Browne.

- **Statistical techniques**
  - *Survival Models and Data Analysis*, R. Elandt-Johnson and N. Johnson.
  - *Nonparametric estimation from incomplete observations*, E. L. Kaplan and P. Meier.

- **Robotic self-discovery and self-organisation**
  - *What can I control? A framework for robot self-discovery*, A. Edsinger and C. C. Kemp.
  - *Taming the beast: guided self-organization of behavior in autonomous robots*, G. Martius and J. M. Hermann.

Bibliography

Charles J. Alpert, Zhuo Li, Michael D. Moffitt, Gi-Joon Nam, Jarrod A. Roy, and Gustavo Tellez. What makes a design difficult to route. In *Proceedings of the 19th International Symposium on Physical Design (ISPD '10)*, pages 7–12, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-920-6. doi: 10.1145/1735023.1735028.

Gene M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the Spring Joint Computer Conference (AFIPS '67, Spring)*, pages 483–485, New York, NY, USA, April 1967. ACM. doi: 10.1145/1465482.1465560.

Tom M. Apostol. *NIST Handbook of Mathematical Functions*, chapter 25. Cambridge University Press, 2010.

Naveen Arulselvan and Randall Berry. Efficient power allocations in wireless ARQ protocols. In *Proceedings of the 5th International Symposium on Wireless Personal Multimedia Communications (WPMC '02)*, volume 3, pages 976–980, October 2002. doi: 10.1109/WPMC.2002.1088323.

Ana Azevedo, Ilya Issenin, Radu Cornea, Rajesh Gupta, Nikil Dutt, Alex Veidenbaum, and Alex Nicolau. Profile-based dynamic voltage scheduling using program checkpoints. In *Proceedings of Design Automation and Test in Europe, DATE '02*, pages 168–176, March 2002.

Nikhil Bansal, Tracy Kimbrel, and Kirk Pruhs. Dynamic speed scaling to manage energy and temperature. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, pages 520–529, October 2004. doi: 10.1109/FOCS.2004.24.

Jeffrey A. Barnett. Dynamic task-level voltage scheduling optimizations. *IEEE Transactions on Computers*, 54(5):508–520, May 2005. ISSN 0018-9340. doi: 10.1109/TC.2005.77.

Luiz Andre Barroso and Urs Hölzle. The case for energy-proportional computing. *Computer*, 40:33–37, 2007. ISSN 0018-9162. doi: 10.1109/MC.2007.443.

Ricardo Bedin França, Denis Favre-Felix, Xavier Leroy, Marc Pantel, and Jean Souyris. Towards formally verified optimizing compilation in flight control software. In Philipp Lucas, Lothar Thiele, Benoit Triquet, Theo Ungerer, and Reinhard Wilhelm, editors, *Predictability and Performance in Embedded Systems (PPES 2011)*, volume 18 of *OpenAccess Series in Informatics (OASIcs)*, pages 59–68, Grenoble, France, March 2011. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi: 10.4230/OASIcs.PPES.2011.59. URL `http://hal.inria.fr/inria-00551370/en/`.

Luca Benini, Alessandro Bogliolo, Giuseppe A. Paleologo, and Giovanni De Micheli. Policy optimization for dynamic power management. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18:813–833, 1998.

Charles H. Bennett. Logical reversibility of computation. *IBM Journal of Research and Development*, 17(6):525–532, 1973.

Charles H. Bennett and Rolf Landauer. The fundamental physical limits of computation. *Scientific American*, 253(1):48–56, 1985.

Claude Berrou, Alain Glavieux, and Punya Thitimajshima. Near Shannon limit error-correcting coding and decoding: turbo-codes. In *Proceedings of the IEEE International Conference on Communications 1993 (ICC '93)*, volume 2, pages 1064–1070, May 1993. doi: 10.1109/ICC.1993.397441.

Shekhar Borkar. Thousand core chips: a technology perspective. In *Proceedings of the 44th annual Design Automation Conference (DAC '07)*, pages 746–749, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-627-1. doi: 10.1145/1278480.1278667.

Stephen P. Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004. ISBN 9780521833783. URL `http://www.stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf`.

Mark G. Brockington. A taxonomy of parallel game-tree search algorithms. *Journal of the International Computer Chess Association*, 19(3):162–174, September 1996.

David Brooks, Vivek Tiwari, and Margaret Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pages 83–94. ACM, June 2000.

David J. Brown and Charles Reams. Toward energy-efficient computing. *ACM Queue*, 8 (2):30–43, 2010. doi: 10.1145/1716383.1730791.

Thomas D. Burd and Robert W. Brodersen. Energy efficient CMOS microprocessor design. In *Proceedings of the Hawaii International Conference on System Sciences (HICSS '95)*, pages 288–297, 1995.

Alfred J. Cavallo. High-capacity factor wind energy systems. *Journal of Solar Energy Engineering*, 117(2):137–143, 1995. doi: 10.1115/1.2870843. URL `http://link.aip.org/link/?SLE/117/137/1`.

Anantha P. Chandrakasan, Samuel Sheng, and Robert W. Brodersen. Low power CMOS digital design. *IEEE Journal of Solid State Circuits*, 27:473–484, 1995.

Anantha P. Chandrakasan, Vadim Gutnik, and Thucydides Xanthopoulos. Data driven signal processing: an approach for energy efficient computing. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED '96)*, pages 347–352, Piscataway, NJ, USA, 1996. IEEE Press. ISBN 0-7803-3571-6.

Jian-Jia Chen and Chin-Fu Kuo. Energy-efficient scheduling for real-time systems on dynamic voltage scaling (DVS) platforms. In *Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA '07)*, pages 28–38, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 0-7695-2975-5. doi: 10.1109/RTCSA.2007.37.

Sangyeun Cho and Rami G. Melhem. Corollaries to Amdahl's Law for energy. *IEEE Computer Architecture Letters*, 7:25–28, 2008.

Kihwan Choi, Karthik Dantu, Wei-Chung Cheng, and Massoud Pedram. Frame-based dynamic voltage and frequency scaling for a MPEG decoder. In *Proceedings of the 2002 IEEE/ACM International Conference on Computer-Aided Design (ICCAD '02)*, pages 732–737, New York, NY, USA, 2002. ACM. ISBN 0-7803-7607-2. doi: 10.1145/774572.774680.

Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *Proceedings of the 2nd Symposium on Networked Systems Design & Implementation (NSDI '05)*, volume 2, pages 273–286, Berkeley, CA, USA, 2005. USENIX Association. URL `http://dl.acm.org/citation.cfm?id=1251203.1251223`.

Byron Cook, Andreas Podelski, and Andrey Rybalchenko. Terminator: Beyond safety. In Thomas Ball and Robert Jones, editors, *Computer Aided Verification*, volume 4144 of *Lecture Notes in Computer Science*, pages 415–418. Springer Berlin / Heidelberg, 2006. ISBN 978-3-540-37406-0. doi: 10.1007/11817963/37.

Puyan Dadvar and Kevin Skadron. Potential thermal security risks. In *Semiconductor Thermal Measurement and Management Symposium, 2005 IEEE Twenty First Annual IEEE*, pages 229–234, March 2005. doi: 10.1109/STHERM.2005.1412184.

Shamik Das, Anantha Chandrakasan, and Rafael Reif. Three-dimensional integrated circuits: performance, design methodology, and CAD tools. In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI '03)*, pages 13–18, February 2003. doi: 10.1109/ISVLSI.2003.1183348.

Robert H. Dennard, Fritz H. Gaensslen, V. Leo Rideout, Ernest Bassous, and Andre R. LeBlanc. Design of ion-implanted MOSFET's with very small physical dimensions. *IEEE Journal of Solid-State Circuits*, 9(5):256–268, October 1974. ISSN 0018-9200. doi: 10.1109/JSSC.1974.1050511.

Lewie Donckers, Paul J. M. Havinga, and Lodewijk Theodoor Smit. Energy efficient TCP. In *2nd Asian International Mobile Computing Conference (AMOC '02)*, pages 18–28. ACM Sigmobile, 2002. URL `http://doc.utwente.nl/38357/`.

Ronald G. Dreslinski, Michael Wieckowski, David Blaauw, Dennis Sylvester, and Trevor Mudge. Near-threshold computing: reclaiming Moore's law through energy efficient integrated circuits. *Proceedings of the IEEE*, 98(2):253–266, February 2010. ISSN 0018-9219. doi: 10.1109/JPROC.2009.2034764.

Christophe Dubach, Timothy M. Jones, Edwin V. Bonilla, and Michael F. P. O'Boyle. A predictive model for dynamic microarchitectural adaptivity control. In *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '10)*, pages 485–496, Washington, DC, USA, 2010. IEEE Computer Society. ISBN 978-0-7695-4299-7. doi: 10.1109/MICRO.2010.14.

Ipek Engin and Sally A. McKee. Efficiently exploring architectural design spaces via predictive modeling. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 195–206, 2006.

Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. Power provisioning for a warehouse-sized computer. In *Proceedings of the 34th annual International Symposium on Computer Architecture (ISCA '07)*, pages 13–23, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-706-3. doi: 10.1145/1250662.1250665.

Alexandra Fedorova. *Operating system scheduling for chip multithreaded processors.* PhD thesis, Harvard University, September 2006. URL `http://www.eecs.harvard.edu/~fedorova/thesis.pdf`.

Alexandra Fedorova, Juan Carlos Saez, Daniel Shelepov, and Manuel Prieto. Maximizing power efficiency with asymmetric multicore systems. ACM Queue, November 2009. URL `http://queue.acm.org/detail.cfm?id=1658422`.

Richard P. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21:467–488, 1982.

Jason Flinn and Mahadev Satyanarayanan. PowerScope: a tool for profiling the energy usage of mobile applications. In *2nd IEEE Workshop on Mobile Computing Systems and Applications, 1999 (WMCSA '99)*, pages 2–10, February 1999. doi: 10.1109/MCSA.1999.749272.

Aviezri Fraenkel and David Lichtenstein. Computing a perfect strategy for $n \times n$ chess requires time exponential in n. In Shimon Even and Oded Kariv, editors, *Automata, Languages and Programming*, volume 115 of *Lecture Notes in Computer Science*, pages 278–293. Springer Berlin / Heidelberg, 1981. ISBN 978-3-540-10843-6. 10.1007/3-540-10843-2_23.

Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

Matthew Garrett. Powering down. *ACM Queue*, 5:16–21, November 2007. ISSN 1542-7730. doi: 10.1145/1331287.1331293.

Robert Gibbons. *A Primer in Game Theory.* Financial Times/Prentice Hall, first edition, June 1992. ISBN 978-0745011592.

Donald B. Gillies. Three new Mersenne primes and a statistical theory. *Mathematics of Computation*, 18(85):93–97, 1964. ISSN 00255718. URL `http://www.jstor.org/stable/2003409`.

Chris Gottbrath, Jeremy Bailin, Casey Meakin, Todd Thompson, and J. J. Charfman. The effects of Moore's law and slacking on large computations, 1999. URL `http://arxiv.org/abs/astro-ph/9912202`.

Kinshuk Govil, Edwin Chan, and Hal Wasserman. Comparing algorithms for dynamic speed-setting of a low-power CPU. In *Proceedings of the 1st International Conference on Mobile Computing and Networking (MobiCom '95)*, pages 13–25, New York, NY, USA, 1995. ACM. ISBN 0-89791-814-2. doi: 10.1145/215530.215546.

Izrail Solomonovich Gradshteyn, Alan Jeffrey, and Iosif Moiseevich Ryzhik. *Table of integrals, series, and products; 4th corrected and revised ed.* Academic Press, New York, NY, 1980. Translated from the 4th Russian edition, Moscow, 1963.

Goetz Graefe. The five-minute rule twenty years later, and how flash memory changes the rules. In *Proceedings of the 3rd international workshop on Data Management on New Hardware (DaMoN '07)*, pages 6:1–6:9, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-772-8. doi: 10.1145/1363189.1363198.

Jim Gray and Goetz Graefe. The five-minute rule ten years later, and other computer storage rules of thumb. *Record of the ACM Special Interest Group on Management of Data (SIGMOD)*, 26:63–68, December 1997. ISSN 0163-5808. doi: 10.1145/271074.271094.

Jim Gray and Franco Putzolu. The 5 minute rule for trading memory for disc accesses and the 10 byte rule for trading memory for CPU time. *Record of the ACM Special Interest Group on Management of Data (SIGMOD)*, 16:395–398, December 1987. ISSN 0163-5808. doi: 10.1145/38714.38755.

Flavius Gruian. Hard real-time scheduling for low-energy using stochastic data and DVS processors. In *Proceedings of the 2001 international symposium on Low power electronics and design (ISLPED '01)*, pages 46–51, New York, NY, USA, 2001. ACM. ISBN 1-58113-371-5. doi: 10.1145/383082.383092.

Flavius Gruian and Krzysztof Kuchcinski. Uncertainty-based scheduling: energy-efficient ordering for tasks with variable execution time. In *Proceedings of the 2003 International Symposium on Low Power Electronics and Design (ISLPED '03)*, pages 465–468, New York, NY, USA, 2003. ACM. ISBN 1-58113-682-X. doi: 10.1145/871506.871621.

Vadim Gutnik and Anantha P. Chandrakasan. Embedded power supply for low-power DSP. *IEEE Transactions on Very Large Scale Integrated Systems*, 5:425–435, December 1997. ISSN 1063-8210. doi: 10.1109/92.645069. URL `http://portal.acm.org/citation.cfm?id=271045.271075`.

D.M. Harris, B. Keller, J. Karl, and S. Keller. A transregional model for near-threshold circuits with application to minimum-energy operation. In *2010 International Conference on Microelectronics (ICM '10)*, pages 64–67, December 2010. doi: 10.1109/ICM.2010.5696207.

Jahangir Hasan, Ankit Jalote, T. N. Vijaykumar, and Carla E. Brodley. Heat stroke: power-density-based denial of service in SMT. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA '05)*, pages 166–177. IEEE Computer Society, 2005.

Julian Havil. *Gamma: Exploring Euler's Constant.* Princeton University Press, 2003. ISBN 0-691-09983-9.

Reinhold Heckmann and Christian Ferdinand. Worst-case execution time prediction by static program analysis. In *18th International Parallel and Distributed Processing Symposium (IPDPS 2004)*, pages 26–30. IEEE Computer Society, 2004.

Inki Hong, Gang Qu, Miodrag Potkonjak, and Mani B. Srivastava. Synthesis techniques for low-power hard real-time systems on variable voltage processors. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS '98)*, pages 178–187, Washington, DC, USA, 1998. IEEE Computer Society. ISBN 0-8186-9212-X. URL `http://portal.acm.org/citation.cfm?id=827270.829022`.

Inki Hong, Darko Kirovski, Gang Qu, Miodrag Potkonjak, and Mani B. Srivastava. Power optimization of variable-voltage core-based systems. *IEEE Transactions on Computer-Aided Design*, 18:1702–1714, 1999.

Tibor Horvath, Tarek Abdelzaher, Kevin Skadron, and Xue Liu. Dynamic voltage scaling in multitier web servers with end-to-end delay control. In *IEEE Transactions on Computers*, volume 56, pages 444–458. IEEE Computer Society, April 2007. doi: 10.1109/TC.2007.1003. URL `http://dl.acm.org/citation.cfm?id=1263121.1263166`.

Chung-Hsing Hsu and Ulrich Kremer. The design, implementation, and evaluation of a compiler algorithm for CPU energy reduction. In *Proceedings of the ACM SIGPLAN 2003 conference on Programming Language Design and Implementation (PLDI '03)*, pages 38–48, New York, NY, USA, 2003. ACM. ISBN 1-58113-662-5. doi: 10.1145/781131.781137.

Wen-Mei Hwu, Christopher Rodrigues, Shane Ryoo, and John Stratton. Compute Unified Device Architecture application suitability. *Computing in Science and Engineering*, 11: 16–26, May 2009. ISSN 1521-9615. doi: 10.1109/MCSE.2009.48. URL `http://dl.acm.org/citation.cfm?id=1550395.1550469`.

Sandy Irani, Sandeep Shukla, and Rajesh Gupta. Algorithms for power savings. *ACM Transactions on Algorithms*, 3, November 2007. ISSN 1549-6325. doi: 10.1145/1290672.1290678.

Tohru Ishihara and Hiroto Yasuura. Voltage scheduling problem for dynamically variable voltage processors. In *Proceedings of the 1998 International Symposium on Low Power Electronics and Design (ISLPED '98)*, pages 197–202, New York, NY, USA, 1998. ACM. ISBN 1-58113-059-7. doi: 10.1145/280756.280894.

Shigeki Iwata and Takumi Kasai. The Othello game on an $n \times n$ board is PSPACE-complete. *Theoretical Computer Science*, 123(2):329–340, 1994. ISSN 0304-3975. doi: 10.1016/0304-3975(94)90131-7. URL `http://www.sciencedirect.com/science/article/pii/0304397594901317`.

Mohammad Reza Kakoee, Ashoka Sathanur, Antonio Pullini, Jos Huisken, and Luca Benini. Automatic synthesis of near-threshold circuits with fine-grained performance tunability. In *Proceedings of the 16th ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED '10)*, pages 401–406, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0146-6. doi: 10.1145/1840845.1840934.

Richard M. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

Randy H. Katz. Tech titans building boom. *IEEE Spectrum*, February 2009. URL `http://www.spectrum.ieee.org/green-tech/buildings/tech-titans-building-boom`.

Wonyoung Kim, Meeta S. Gupta, Gu-yeon Wei, and David Brooks. System level analysis of fast, per-core DVFS using on-chip switching regulators. In *Proceedings of the 14th International Symposium on High-Performance Computer Architecture (HPCA '08)*, 2008.

Donald E. Knuth. *The Art of Computer Programming*, volume 2 (Seminumerical Algorithms). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, third edition, 1997. ISBN 0201896842. URL `http://portal.acm.org/citation.cfm?id=270146`.

Jonathan G. Koomey, Stephen Berard, Marla Sanchez, and Henry Wong. *Assessing trends in the electrical efficiency of computation over time.* Oakland: Analytics Press, 2009.

Jonathan G. Koomey, Stephen Berard, Marla Sanchez, and Henry Wong. Implications of historical trends in the electrical efficiency of computing. *IEEE Annals of the History of Computing*, 33(3):46–54, March 2011. ISSN 1058-6180. doi: 10.1109/MAHC.2010.28.

C. M. Krishna and Kang G. Shin. *Real-Time Systems.* McGraw-Hill, 1997. ISBN 9780070570436.

Kelin J. Kuhn. Moore's law past 32nm: Future challenges in device scaling. In *13th International Workshop on Computational Electronics (IWCE '09)*, pages 1–6, May 2009. doi: 10.1109/IWCE.2009.5091124.

Chidamber Kulkarni, Dennis Moolenaar, Lode Nachtergaele, Francky Catthoor, and Hugo J. de Man. System-level energy-delay exploration for multimedia applications on embedded cores with hardware cache. *Journal of VLSI Signal Processing Systems*, 22:45–57, August 1999. ISSN 0922-5773. doi: 10.1023/A:1008121818984. URL `http://dl.acm.org/citation.cfm?id=331769.331779`.

Pavan Kumar and Mani Srivastava. Power-aware multimedia systems using run-time prediction. In *Proceedings of the The 14th International Conference on VLSI Design (VLSID '01)*, pages 64–78, Washington, DC, USA, 2001. IEEE Computer Society. ISBN 0-7695-0831-6. URL `http://portal.acm.org/citation.cfm?id=580549.835351`.

Rakesh Kumar, Keith I. Farkas, Norman P. Jouppi, Parthasarathy Ranganathan, and Dean M. 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, December 2003. doi: 10.1109/MICRO.2003.1253185.

Hsiang-Tsung Kung, Fabrizio Luccio, and Franco P. Preparata. On finding the maxima of a set of vectors. *Journal of the ACM*, 22:469–476, 1975.

John Lampert. *Probability: A survey of the mathematical theory.* Mathematics Monograph Series. W. A. Benjamin Inc., first edition, 1966. ISBN 978-0471154075.

Rolf Landauer. Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development*, 5:183–191, 1961.

Benjamin C. Lee and David M. Brooks. Accurate and efficient regression modeling for microarchitectural performance and power prediction. *SIGOPS Oper. Syst. Rev.*, 40: 185–194, October 2006. ISSN 0163-5980. doi: 10.1145/1168917.1168881.

Dennis Lee. Energy management issues for computer systems, 2000. URL `http://www.cs.washington.edu/homes/dlee/frontpage/mypapers/generals.ps.gz`.

Seongsoo Lee and Takayasu Sakurai. Run-time power control scheme using software feedback loop for low-power real-time application. In *Proceedings of the 2000 Asia and South Pacific Design Automation Conference (ASP-DAC '00)*, pages 381–386, New York, NY, USA, 2000. ACM. ISBN 0-7803-5974-7. doi: 10.1145/368434.368693.

John P. Lehoczky, Lui Sha, and Ye Ding. Rate-monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proceedings of the 11th IEEE Real-time Systems Symposium*, pages 166–171, December 1989.

Ian M. Leslie, David McAuley, Richard Black, Timothy Roscoe, Paul Barham, David Evers, Robin Fairbairns, and Eoin Hyden. The design and implementation of an operating system to support distributed multimedia applications. *IEEE Journal on Selected Areas in Communications*, 14(7):1280–1297, September 1996. ISSN 0733-8716. doi: 10.1109/49.536480.

Oliver Yuk-Hang Leung, Chung-Wai Yue, Chi-ying Tsui, and Roger S. Cheng. Reducing power consumption of turbo code decoder using adaptive iteration with variable supply voltage. In *Proceedings of the 1999 International Symposium on Low Power Electronics and Design (ISLPED '99)*, pages 36–41, New York, NY, USA, 1999. ACM. ISBN 1-58113-133-X. doi: 10.1145/313817.313836.

Adam Leventhal. Flash storage today. ACM Queue, July 2008. URL `http://queue.acm.org/detail.cfm?id=1413262`.

Jian Li and José F. Martínez. Power-performance considerations of parallel computing on chip multiprocessors. *ACM Transactions on Architecture and Code Optimization*, 2: 397–422, 2005.

Zhuo Li, Charles J. Alpert, Shiyan Hu, Tuhin Muhmud, Stephen T. Quay, and Paul G. Villarrubia. Fast interconnect synthesis with layer assignment. In *Proceedings of the 2008 International Symposium on Physical Design (ISPD '08)*, pages 71–77, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-048-7. doi: 10.1145/1353629.1353648.

David Lichtenstein and Michael Sipser. Go is polynomial-space hard. *J. ACM*, 27:393–401, April 1980. ISSN 0004-5411. doi: 10.1145/322186.322201.

Jane W.-S. Liu. *Real-time systems*. Prentice Hall, 2000. ISBN 978-0-13-099651-0.

Jacob R. Lorch and Alan Jay Smith. Improving dynamic voltage scaling algorithms with PACE. *SIGMETRICS Perform. Eval. Rev.*, 29:50–61, June 2001. ISSN 0163-5999. doi: 10.1145/384268.378429.

Peter Loscocco and Stephen Smalley. Integrating flexible support for security policies into the Linux operating system. In *Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference*, pages 29–42, Berkeley, CA, USA, 2001. USENIX Association. ISBN 1-880446-10-3. URL `http://dl.acm.org/citation.cfm?id=647054.715771`.

Wolfgang Lutz, Warren Sanderson, and Sergei Scherbov. The end of world population growth. *Nature*, 412, 2001. doi: 10.1038/35087589.

John Markoff and Saul Hansell. Hiding in plain sight, Google seeks more power. New York Times, 2006. URL `http://www.nytimes.com/2006/06/14/technology/14search.html?pagewanted=all`.

Mitsuru Matsui. How far can we go on the x64 processors? In *Fast Software Encryption, 13th International Workshop, FSE 2006*, pages 341–358. Springer-Verlag, 2006. URL `http://www.iacr.org/cryptodb/archive/2006/FSE/3246/3246.pdf`.

Rami Melhem, Nevine AbouGhazaleh, Hakan Aydin, and Daniel Mossé. *Power management points in power-aware real-time systems*, pages 127–152. Kluwer Academic Publishers, Norwell, MA, USA, 2002. ISBN 0-306-46786-0. URL `http://portal.acm.org/citation.cfm?id=783060.783068`.

Rami Melhem, Daniel Mossé, and Elmootazbellah (Mootaz) Elnozahy. The interplay of power management and fault recovery in real-time systems. *IEEE Transactions on Computers*, 53(2):217–231, February 2004. ISSN 0018-9340. doi: 10.1109/TC.2004.1261830.

Amitabh Menon, S. K. Nandy, and Mahesh Mehendale. Multivoltage scheduling with voltage-partitioned variable storage. In *Proceedings of the 2003 international symposium on Low power electronics and design (ISLPED '03)*, pages 298–301, New York, NY, USA, 2003. ACM. ISBN 1-58113-682-X. doi: 10.1145/871506.871580.

Eytan Modiano. An adaptive algorithm for optimizing the packet size used in wireless ARQ protocols. *Wireless Networks*, 5:279–286, 1999. ISSN 1022-0038. doi: 10.1023/A:1019111430288.

Jeffrey C. Mogul, Eduardo Argollo, Mehul Shah, and Paolo Faraboschi. Operating system support for NVM + DRAM hybrid main memory. In *Proceedings of the 13th Hot Topics in Operating Systems (HotOS)*, 2009.

Matteo Monchiero, Ramon Canal, and Antonio González. Design space exploration for multicore architectures: a power/performance/thermal view. In *Proceedings of the 20th annual International Conference on Supercomputing (ICS '06)*, pages 177–186, New York, NY, USA, 2006. ACM. ISBN 1-59593-282-8. doi: 10.1145/1183401.1183428.

Matthias Müller, David Charypar, and Markus Gross. Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer animation (SCA '03)*, pages 154–159, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association. ISBN 1-58113-659-5. URL `http://dl.acm.org/citation.cfm?id=846276.846298`.

Naveen Muralimanohar, Rajeev Balasubramonian, and Norman P. Jouppi. Cacti 6.0: A tool to model large caches. Technical report, School of Computing, University of Utah, 2007.

James E. Nymann. On the probability that positive integers are relatively prime. *Journal of Number Theory*, 4:469–473, 1972.

Takanori Okuma, Tohru Ishihara, and Hiroto Yasuura. Real-time task scheduling for a variable voltage processor. In *Proceedings of the 12th International Symposium on*

System Synthesis (ISSS '99), pages 24–29, Washington, DC, USA, 1999. IEEE Computer Society. ISBN 0-7695-0356-X. URL `http://portal.acm.org/citation.cfm?id=857198.857958`.

Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank citation ranking: bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999. URL `http://ilpubs.stanford.edu:8090/422/`. Previous number = SIDL-WP-1999-0120.

Massoud Pedram and Qing Wu. Design considerations for battery-powered electronics. In *Proceedings of the 36th annual ACM/IEEE Design Automation Conference (DAC '99)*, pages 861–866, New York, NY, USA, 1999. ACM. ISBN 1-58113-109-7. doi: 10.1145/309847.310089.

Vincent Poirriez, Nicola Yanev, and Rumen Andonov. A hybrid algorithm for the unbounded knapsack problem. *Discrete Optimization*, 6:110–124, 2009. doi: 10.1016/j.disopt.2008.09.004. URL `http://hal.inria.fr/inria-00335065/en/`. Hubert Curien French-Bulgarian partnership RILA 2006 No 15071XF.

Mateja Putic, Liang Di, Benton H. Calhoun, and John Lach. Panoptic DVS: a fine-grained dynamic voltage scaling framework for energy scalable CMOS design. In *Proceedings of the 2009 IEEE International Conference on Computer Design (ICCD '09)*, pages 491–497, Piscataway, NJ, USA, 2009. IEEE Press. ISBN 978-1-4244-5029-9. URL `http://portal.acm.org/citation.cfm?id=1792354.1792447`.

Qinru Qiu and Massoud Pedram. Dynamic power management based on continuous-time Markov decision processes. In *Proceedings of the 36th annual ACM/IEEE Design Automation Conference (DAC '99)*, pages 555–561, New York, NY, USA, 1999. ACM. ISBN 1-58113-109-7. doi: 10.1145/309847.309997.

Gang Qu and Miodrag Potkonjak. Achieving utility arbitrarily close to the optimal with limited energy. In *Proceedings of the 2000 International Symposium on Low Power Electronics and Design (ISLPED '00)*, pages 125–130, New York, NY, USA, 2000. ACM. ISBN 1-58113-190-9. doi: 10.1145/344166.344545.

Vijay T. Raisinghani and Sridhar Iyer. Cross-layer design optimizations in wireless protocol stacks. *Computer Communications*, 27:720–724, 2004.

Renewable Energy Research Laboratory. Wind power: capacity factor, intermittency, and what happens when the wind doesn't blow? *Community Wind Power Fact Sheet*, 2009.

Kurt W. Roth and Kurtis McKenney. Energy consumption by consumer electronics in U.S. residences. *Final Report to the Consumer Electronics Association*, January 2007.

Kaushik Roy, Saibal Mukhopadhyay, and Hamid Mahmoodi-Meimand. Leakage current mechanisms and leakage reduction techniques in deep-submicrometer CMOS circuits. *Proceedings of the IEEE*, 91(2):305–327, February 2003. ISSN 0018-9219. doi: 10.1109/JPROC.2002.808156.

Walter Rudin. *Real and complex analysis.* McGraw-Hill Book Co., New York, third edition, 1987. ISBN 0-07-054234-1.

Jonathan Schaeffer, Neil Burch, Yngvi Björnsson, Akihiro Kishimoto, Martin Müller, Robert Lake, Paul Lu, and Steve Sutphen. Checkers is solved. *Science*, 317(5844): 1518–1522, 2007. doi: 10.1126/science.1144079. URL `http://www.sciencemag.org/content/317/5844/1518.abstract`.

Semiconductor Industry Association. Annual report, 2005. URL `http://www.chiphistory.org/exhibits/ex_moores_law_SIAar/SIA_AR_2005.pdf`.

Mingoo Seok, Scott Hanson, Yu-Shiang Lin, Zhiyoong Foo, Daeyeon Kim, Yoonmyung Lee, Nurrachman Liu, Dennis Sylvester, and David Blaauw. The phoenix processor: A 30pw platform for sensor applications. In *2008 IEEE Symposium on VLSI Circuits*, pages 188–189, June 2008. doi: 10.1109/VLSIC.2008.4586001.

Kiran Seth, Aravindh Anantaraman, Frank Mueller, and Eric Rotenberg. FAST: frequency-aware static timing analysis. In *IEEE Real-time Systems Symposium*, pages 40–51, 2003.

Daniel Shelepov, Juan Carlos Saez Alcaide, Stacey Jeffery, Alexandra Fedorova, Nestor Perez, Zhi Feng Huang, Sergey Blagodurov, and Viren Kumar. HASS: A scheduler for heterogeneous multicore systems. *SIGOPS Oper. Syst. Rev.*, 43:66–75, April 2009. ISSN 0163-5980. doi: 10.1145/1531793.1531804.

Suresh Siddha, Venkatesh Pallipadi, and Arjan Van De Ven. Getting maximum mileage out of tickless. In Intel Open Source Technology Center, editor, *Proceedings of the Linux Symposium*, June 2007. URL `http://software.intel.com/sites/oss/pdfs/maximum_tickless.pdf`.

Gert Smolka. The Oz programming model. In *Computer Science Today, Lecture Notes in Computer Science*, pages 324–343. Springer-Verlag, 1995.

Yusuke Soejima, Akihiro Kishimoto, and Osamu Watanabe. Evaluating root parallelization in Go. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(4): 278–287, December 2010. ISSN 1943-068X. doi: 10.1109/TCIAIG.2010.2096427.

Jean Souyris, Erwan Le Pavec, Guillaume Himbert, Victor Jégu, and Guillaume Borios. Computing the worst case execution time of an avionics program by abstract interpretation. In *Proceedings of the 5th International Workshop on Worst-Case Execution Time (WCET) Analysis*, pages 21–24, 2005.

Mircea R. Stan and Wayne P. Burleson. Bus-invert coding for low-power I/O. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 3(1):49–58, March 1995. ISSN 1063-8210. doi: 10.1109/92.365453.

Mark Stemm and Randy H. Katz. Measuring and reducing energy consumption of network interfaces in hand-held devices. *Institute of Electronics, Information and Communication Engineers: Transactions on Communications*, August 1997.

John E. Stone, David Gohara, and Guochun Shi. OpenCL: A parallel programming standard for heterogeneous computing systems. *Computing in Science Engineering*, 12 (3):66–73, May–June 2010. ISSN 1521-9615. doi: 10.1109/MCSE.2010.69.

Ching-Long Su, Chi-Ying Tsui, and Alvin M. Despain. Low power architecture design and compilation techniques for high-performance processors. In *1994 IEEE Compcon*, pages 489–498, March 1994a. ISBN 0-8186-5380-9.

Ching-Long Su, Chi-Ying Tsui, and Alvin M. Despain. Saving power in the control path of embedded processors. *IEEE Design & Test*, 11:24–30, October 1994b. ISSN 0740-7475. doi: 10.1109/54.329448. URL `http://dl.acm.org/citation.cfm?id=622176.622570`.

Vishu Swaminathan and Krishnendu Chakrabarty. Investigating the effect of voltage-switching on low-energy task scheduling in hard real-time systems. In *Proceedings of the 2001 Asia and South Pacific Design Automation Conference (ASP-DAC '01)*, pages 251–254, New York, NY, USA, 2001. ACM. ISBN 0-7803-6634-4. doi: 10.1145/370155.370337.

Yuan Taur and Tak H. Ning. *Fundamentals of modern VLSI devices*, volume 1. Cambridge University Press, 1998.

Vivek Tiwari, Sharad Malik, and Andrew Wolfe. Compilation techniques for low energy: an overview. In *Digest of Technical Papers from IEEE Symposium on Low Power Electronics*, pages 38–39, October 1994.

Paolo Toth. Dynamic programming algorithms for the zero-one knapsack problem. *Computing*, 25:29–45, 1980. ISSN 0010-485X. URL `http://dx.doi.org/10.1007/BF02243880`. 10.1007/BF02243880.

Vassilis Tsaoussidis, Hussein Badr, Xiaocheng Ge, and Kostas Pentikousis. Energy/throughput tradeoffs of TCP error control strategies. In *Proceedings of the Fifth IEEE Symposium on Computers and Communications (ISCC '00)*, pages 106–126, Washington, DC, USA, 2000. IEEE Computer Society. ISBN 0-7695-0722-0. URL `http://dl.acm.org/citation.cfm?id=844383.845463`.

U.S. Environmental Protection Agency. Report to Congress on server and data center energy efficiency, 2010. URL `http://www.energystar.gov/ia/partners/prod_development/downloads/EPA_Datacenter_Report_Congress_Final1.pdf`.

Amin Vahdat, Alvin Lebeck, and Carla Schlatter Ellis. Every joule is precious: the case for revisiting operating system design for energy efficiency. In *Proceedings of the 9th workshop on ACM SIGOPS European workshop: beyond the PC: new challenges for the operating system*, EW 9, pages 31–36, New York, NY, USA, 2000. ACM. doi: 10.1145/566726.566735.

Vincent R. Von Kaenel, Peter Macken, and Marc G.R. Degrauwe. A voltage reduction technique for battery-operated systems. *IEEE Journal of Solid-State Circuits*, 25(5): 1136–1140, October 1990. ISSN 0018-9200. doi: 10.1109/4.62134.

William W. Wadge and Edward A. Ashcroft. *LUCID, the dataflow programming language*. Academic Press Professional, Inc., San Diego, CA, USA, 1985. ISBN 0-12-729650-6.

Yefu Wang, Xiaorui Wang, Ming Chen, and Xiaoyun Zhu. Power-efficient response time guarantees for virtualized enterprise servers. In *Real-Time Systems Symposium*, pages 303–312, December 2008. doi: 10.1109/RTSS.2008.20.

Peter Whittle. *Probability via expectation.* Springer Texts in Statistics. Springer, 2000. ISBN 9780387989556.

Enhua Wu and Youquan Liu. Emerging technology about GPGPU. In *IEEE Asia Pacific Conference on Circuits and Systems*, pages 618–622, December 2008. doi: 10.1109/APCCAS.2008.4746099.

Ruibin Xu, Daniel Mossé, and Rami Melhem. Minimizing expected energy in real-time embedded systems. In *Proceedings of the 5th ACM International Conference on Embedded Software (EMSOFT '05)*, pages 251–254, 2005.

Frances Yao, Alan Demers, and Scott Shenker. A scheduling model for reduced CPU energy. *Proceedings of 36th Annual Symposium on Foundations of Computer Science*, pages 374–382, 1995.

Christian Zamfir, Colin Perkins, and Peter Dickman. Live migration of virtual block devices. In *EuroSyS 2007*. Association of Computing Machinery, 2007. URL `http://eprints.gla.ac.uk/43159/`.

Wensheng Zhang, Mahmut Taylan Kandemir, Narayanan Vijaykrishnan, Mary Jane Irwin, and Vivek K. De. Compiler support for reducing leakage energy consumption. In *Design, Automation and Test in Europe Conference and Exhibition, 2003*, pages 1146–1147, 2003. doi: 10.1109/DATE.2003.1253774.

Yumin Zhang, Xiaobo (Sharon) Hu, and Danny Z. Chen. Efficient global register allocation for minimizing energy consumption. *SIGPLAN Not.*, 37:42–53, April 2002. ISSN 0362-1340. doi: 10.1145/510857.510867.

Dakai Zhu. Reliability-aware dynamic energy management in dependable embedded real-time systems. In *Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 397–407, 2006.

Dakai Zhu, Rami Melhem, Daniel Mossé, and Elmootazbellah (Mootaz) Elnozahy. Analysis of an energy efficient optimistic TMR scheme. In *Proceedings of the 10th International Conference on Parallel and Distributed Systems (ICPADS '04)*, pages 559–568, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2152-5. doi: 10.1109/ICPADS.2004.20.

Nan Zhu and Kevin Broughan. On dominated terms in the general knapsack problem. *Operations Research Letters*, 21(1):31–37, 1997. ISSN 0167-6377. doi: 10.1016/S0167-6377(97)00018-7. URL `http://www.sciencedirect.com/science/article/B6V8M-3SX27T9-5/2/b5088d374dfcc0f1e2061e12bbb5158b`.

Hubert Zimmermann. OSI reference model—the ISO model of architecture for open systems interconnection. *IEEE Transactions on Communications*, 28(4):425–432, April 1980. ISSN 0090-6778. doi: 10.1109/TCOM.1980.1094702.

Glossary

ACPI Advanced Configuration and Power Interface.

ARQ Automatic Repeat Request.

BSOM Batch Self-Organising Map, a data-mining benchmark.

CAGR combined annual growth rate.

cdf cumulative density function.

CMOS complementary metal–oxide–semiconductor, the standard technology for integrated circuits.

CPU central processing unit.

crowdsourcing the combined effort of a distributed group of people to achieve a shared goal.

DDR double data rate.

DES Data Encryption Standard, a once-popular encryption–decryption algorithm.

DIMM dual in-line memory module.

DVFS digital voltage–frequency scaling.

DVS digital voltage scaling.

EPA Environmental Protection Agency.

EPF energy per frame.

FEC forward error correction.

GB gigabyte, 2^{30} bytes.

Gb gigabit, 2^{30} bits.

GIMPS Great Internet Mersenne Prime Search.

GPGPU general-purpose processing on the GPU.

GPU graphics processing unit.

HVAC heating, ventilation, and air-conditioning.

I/O input/output.

IP Internet Protocol.

ISA instruction set architecture.

LVA Live Variable Analysis.

Mersenne number an integer of the form $2^n - 1$ for some integer n.

MOSFET metal–oxide–semiconductor field-effect transistor.

MPEG Moving Picture Experts Group.

NTC near-threshold computing.

operating point a configuration of hardware and software, coupled with the execution time and energy requirements of computing in that configuration. A full definition is given in Section 5.1.1 (p. 79).

OS operating system.

OSI Open Systems Interconnection.

pdf probability density function.

PSPACE a complexity class: the set of problems for which an algorithm exists that can solve instances of that problem in an amount of space that grows polynomially with respect to the size of the instance.

RAID Redundant Array of Independent Disks.

RPM revolutions per minute.

RWL routed wire-length.

SDRAM synchronous dynamic random-access memory.

SIMD single-instruction multiple-data.

SRAM static random-access memory.

superscalability the degree to which a processor makes use of instruction-level parallelism.

support the subset of a function's domain over which its value is non-zero.

TCP Transmission Control Protocol.

teraflops one trillion (10^{12}) floating-point operations per second.

TPS transactions per second.

WCEC worst-case execution cycles.

WCET worst-case execution time.