Energy-Aware Performance Engineering in High Performance Computing

by

Stephen Ian Roberts

A thesis submitted to The University of Warwick

in partial fulfilment of the requirements

for admission to the degree of

Doctor of Philosophy

Department of Computer Science

The University of Warwick

June 2017

Abstract

Advances in processor design have delivered performance improvements for decades. As physical limits are reached, however, refinements to the same basic technologies are beginning to yield diminishing returns. Unsustainable increases in energy consumption are forcing hardware manufacturers to prioritise energy efficiency in their designs. Research suggests that software modifications will be needed to exploit the resulting improvements in current and future hardware. New tools are required to capitalise on this new class of optimisation.

This thesis investigates the field of energy-aware performance engineering. It begins by examining the current state of the art, which is characterised by ad-hoc techniques and a lack of standardised metrics. Work in this thesis addresses these deficiencies and lays stable foundations for others to build on.

The first contribution made includes a set of criteria which define the properties that energy-aware optimisation metrics should exhibit. These criteria show that current metrics cannot meaningfully assess the utility of code or correctly guide its optimisation. New metrics are proposed to address these issues, and theoretical and empirical proofs of their advantages are given.

This thesis then presents the Power Optimised Software Envelope (POSE) model, which allows developers to assess whether power optimisation is worth pursuing for their applications. POSE is used to study the optimisation characteristics of codes from the Mantevo mini-application suite running on a Haswell-based cluster. The results obtained show that of these codes TeaLeaf has the most scope for power optimisation while PathFinder has the least.

Finally, POSE modelling techniques are extended to evaluate the system-wide scope for energy-aware performance optimisation. System Summary POSE allows developers to assess the scope a system has for energy-aware software optimisation independent of the code being run.

Acknowledgements

I am indebted to many people for their support, guidance and friendship during my time at the University of Warwick. It gives me great pleasure to acknowledge a small number of them here.

First and foremost I am grateful to my supervisor, Prof. Stephen Jarvis, who gave me the opportunity to undertake this research. He has remained a constant source of advice and encouragement throughout my Ph. D.

I am also extremely grateful to Dr. Suhaib Fahmy for his help and guidance. Suhaib has always gone the extra mile to help, and my research into optimisation metrics in particular would not have been possible without his input.

I would like to thank my lab partners and colleagues in the Department of Computer Science, including Dr. Steven Wright, Dr. Philip Taylor, Dr. Richard Bunt, Dr. Arshad Jhumka, Dr. Adam Chester, Tim Law, Huanzhou Zhu and Andrew Owenson. Their perspectives and insights have proven invaluable.

I also wish to thank the Center of Information Services and High Performance Computing (ZIH) at TU Dresden. Much of this work has benefited from access to their power instrumented supercomputing hardware. A special vote of thanks is owed to Thomas Ilsche for his patience and generosity.

Returning to academia from industry was a leap of faith which would not have been possible without the support of past colleagues. Special thanks in this regard go to Dr. Enrico Scalavino for our many discussions on the subject. Thanks also go to Zheng Huang, Nic Quilici and Amaury Chamayou.

Finally, I would like to thank my friends and family for their support and kindness. Mum, Dad, my cousins David, Dereck and William, my sister Annie and aunt Ann, and also to my uncle Grahame, who sadly left us before this work was finished. This thesis is dedicated to them, with love and gratitude.

Declarations

This thesis is submitted to the University of Warwick in support of my application for the degree of Doctor of Philosophy. It has been composed by myself and has not been submitted in any previous application for any degree.

Parts of this thesis have been published by the author:

[104] S. I. Roberts, S. A. Wright, D. Lecomber, C. January, J. Byrd, X. Oró, and S. A. Jarvis. POSE: A Mathematical and Visual Modelling Tool to Guide Energy Aware Code Optimisation. In *Proceedings of the 6th International Green and Sustainable Computing Conference (IGSC '15)*, December 2015

[105] S. I. Roberts, S. A. Wright, S. A. Fahmy, and S. A. Jarvis. Metrics for Energy-Aware Software Optimisation. *Lecture Notes in Computer Science (LNCS)*, 10266:413–430, June 2017

[103] S. I. Roberts, S. A. Wright, S. A. Fahmy, and S. A. Jarvis. The Power-Optimised Software Envelope. *ACM Transactions on Architecture and Code Optimisation*, in preparation

Sponsorship and Grants

The research presented in this thesis was made possible in part by the support of the following benefactors:

- Technology Strategy Board project number 131197 (Energy-Efficiency Tools for High-Performance Multi- and Many-core Applications).

Abbreviations

ADP	Area Delay Product	50
ALU	Arithmetic Logic Unit	38
ATX	Advanced Technology eXtended	12
CISC	Complex Instruction Set Computing	16
CMOS	Complimentary Metal Oxide Semiconductor	34
CPU	Central Processing Unit	1
CRCW	Concurrent Read Concurrent Write	29
CREW	Concurrent Read Exclusive Write	29
DAG	Directed Acyclic Graph	24
DMM	Distributed Memory Machine	8
DVFS	Dynamic Voltage and Frequency Scaling	33
EDD	Energy Delay Distance	60
EDP	Energy Delay Product	48
EDS	Energy Delay Sum	60
ENIAC	Electronic Numerical Integrator and Computer	35
ERCW	Exclusive Read Concurrent Write	29
EREW	Exclusive Read Exclusive Write	29
FLOPS	Floating Point Operations per Second	17
FoM	Figure of Merit	15
FPE	Feasible Performance Envelope	71
FPGA	Field-Programmable Gate Array	2
GPU	Graphics Processing Unit	2

HDEEM	High Density Energy Efficiency Monitoring	13
HPC	High Performance Computing	1
IBS	Instruction Based Sampling	20
IC	Integrated Circuit	3
ICC	Intel C++ Compiler	66
ILP	Instruction Level Parallelism	26
IPS	Instructions Per Second	16
ITUE	Information Technology Power Usage Effectiveness	17
MIMD	Multiple Instruction Multiple Data	8
MISD	Multiple Instruction Single Data	8
MOO	Multi-Objective Optimisation	15
MSR	Model Specific Register	13
MTTF	Mean Time To Failure	4
NASA	National Aeronautics and Space Administration	45
NUMA	Non-Uniform Memory Access	9
PDP	Power Delay Product	50
PEBS	Precise Event-Based Sampling	20
POSE	Power Optimised Software Envelope	5
PRAM	Parallel Random Access Machine	28
PUE	Power Usage Effectiveness	17
RAPL	Running Average Power Limit	13
RISC	Reduced Instruction Set Computing	16
SIMD	Single Instruction Multiple Data	3
SISD	Single Instruction Single Data	8
SMM	Shared Memory Machine	8

SMP	Symmetric Multiprocessing	9
SST	Structural Simulation Toolkit	31
STFC	Science and Technology Facilities Council	118
TCO	Total Cost of Ownership	4
TDP	Thermal Design Power	42
TLP	Thread Level Parallelism	26

Contents

List of Figures

List of Tables

CHAPTER 1
Introduction

Scientific computing and numerical simulation have become indispensable tools in many areas of science and engineering. Simulations allow scientists to test their theories in domains where physical experimentation would be prohibitively costly, impractical, or dangerous. As a result, computational methods have joined theory and experiment as central pillars of scientific investigation [57].

Maximising performance is paramount in scientific computing. Higher performance means more calculations can be carried out, allowing scientists to increase the size, complexity or resolution of their simulations. This demand for performance has led to the development of *supercomputers*, large machines orders of magnitude more powerful than desktop computers.

Supercomputers are typically constructed by linking many smaller nodes together to form a *cluster*. Specialist tools and programming models are then used to write software that can be run on several nodes in parallel. These nodes communicate over an *interconnect* network, collaborating to run simulations and produce results faster than a single node could manage in isolation.

The field of High Performance Computing (HPC) exists to improve the performance of supercomputers and the software which they run. HPC covers a broad spectrum of disciplines. At one extreme, domain experts write high-level simulation software to model phenomena of interest. At the other, hardware engineers design the processors and other components that make up supercomputers. *Performance engineering* bridges the gap between these extremes, seeking ways to optimise software to make better use of the available hardware.

Moore's law states that transistor density doubles every 18-24 months [93]. This trend has delivered exponential increases in Central Processing Unit (CPU) performance for decades. Dennard scaling, which states that the power use of transistors is proportional to their size [28], kept energy consumption in check as increasing numbers of transistors were packed into CPUs. Together, these laws led to a period known as the "Free Lunch", when rising clock speeds delivered regular performance increases with no additional power cost.

Dennard scaling ended around 2006 [52], and Moore's law is also showing signs of failure [116]. Refinements to the same underlying technologies are yielding diminishing returns, and the "Free Lunch" is now over [115]. Energy consumption is rapidly becoming a limiting factor for continued progress in scientific computing as a result [109].

The end of Dennard scaling has forced hardware engineers to prioritise energy efficiency in their designs. This has lead to both modifications in existing platforms as well as the development of new HPC technologies. Some of these novel technologies are pre-existing products which have been repurposed for scientific computing. Examples of this kind include Field-Programmable Gate Arrays (FPGAs) [29], general purpose Graphics Processing Units (GPUs) [46] and Intel's Xeon Phi coprocessors [21]. Others, like NEC's Aurora Vector Engine, were designed specifically for the HPC market.

Performance engineers are also feeling the effects of this drive towards energy efficiency. One obvious example is the emergence of new programming models like OpenACC, OpenCL and CUDA which allow developers to target the novel energy-efficient accelerator technologies listed above. More subtly, research suggests that targeted modifications to existing software will be required to fully exploit the energy efficiency improvements in modern hardware [111]. New energy-aware performance engineering techniques are being developed to identify and capitalise on this new class of optimisation.

This work investigates how conventional performance engineering techniques can be adapted to support energy-aware software optimisation. It highlights challenges which must be overcome before this new class of optimisation can be widely exploited. It also seeks to quantify the benefits which can realistically be expected as a result of energy-aware optimisation.

1.1 Motivation

Moore's law was first proposed in 1965 and quickly became a self-fulfilling prophecy as hardware manufacturers were forced to keep up with it or face being overtaken by their competition. This resulted in a doubling of transistor density every 18-24 months, fuelled by advances in Integrated Circuit (IC) fabrication, circuit design and processor architectures. Moore's Law has driven the development of computer hardware in this way for decades.

The area occupied by individual transistors halves with every doubling of transistor density. Power density (i.e. the rate of power consumption per unit area) remained constant under Dennard scaling, so the power consumed by each transistor was also halved. This in turn led to faster clock speeds as the maximum switching frequency of a transistor is inversely proportional to its peak power consumption [61].

Clock speeds increased exponentially with each new generation of processors while Dennard scaling persisted. The "Free Lunch" period resulted from this link between transistor density and processor speed. The link was broken when Dennard scaling ended, causing clock speeds to stagnate even as transistor densities continued to rise. Hardware designers now rely on architectural changes such as vectorisation, superscalar architectures and multiple cores to deliver performance improvements [95].

Higher clock speeds deliver performance improvements without developer input, hence the term "Free Lunch". Conversely, software modifications are required to take advantage of novel hardware features. New instructions are needed for Single Instruction Multiple Data (SIMD) vectorisation, for example, and applications must be parallelised to run on multiple cores. Although compilers can perform some of this work, performance engineers typically have to rewrite their applications in order to achieve maximum performance [75].

Processor power density has been rising since the end of Dennard scaling. This trend has been partially offset by one-off advances in fabrication processes and the use of exotic materials in transistors. Such advances only provide temporary reprieve, however, and the overall trend is expected to continue [37].

Rising power density poses a number of challenges to the field of HPC. First, energy costs are soaring as increased per-processor power draw is compounded by the growing number of processors used in modern supercomputers. These costs already represent a large share of the Total Cost of Ownership (TCO) for supercomputing systems and are expected to rise still further [108].

Secondly, higher power densities lead to increased operating temperatures, which can cause problems with hardware reliability [112]. At present, around 20 % of the available compute time on large-scale supercomputers is lost due to hardware failures [34]. This trend is also exacerbated by high processor counts, as shortening the Mean Time To Failure (MTTF) of individual processors has a cumulative effect on the MTTF of machines as a whole.

Finally, power density cannot continue to grow indefinitely. There are limits to how much power can be delivered to processors and how quickly the resulting heat can be removed. If performance improvements cannot be decoupled from increasing power density then these too will come to an end.

Hardware designers are responding to these challenges by prioritising energy efficiency in their processor designs. Improving energy efficiency through architectural changes closely parallels the way in which performance improvements are currently delivered. The expected outcome is also the same; code changes will be needed in order to maximise the benefits of energy efficient hardware features. Energy-aware performance engineering techniques will therefore be required as power becomes a first-class constraint in HPC.

The US Department of Energy has identified energy efficiency as a primary constraint for exascale systems [109]. New performance engineering approaches will be required soon if the current rate of progress in HPC is to be maintained. Fortunately, performance engineers have built up a wealth of tools and experience adapting software to maximise performance on new hardware. This thesis aims to show how these existing tools and techniques can be updated to consider energy as well as runtime.

Energy-aware performance optimisation is still in its infancy, characterised by ad-hoc techniques and a lack of standardised metrics. This thesis contributes a set of metrics which are shown to be more suitable for use in guiding energy-aware optimisation than current alternatives. It also presents a pair of related approaches for identifying the potential for energy-aware optimisation, one for individual codes and the other for entire systems.

The techniques described in this thesis are notable for their generality. They are not platform or application specific and impose very few prerequisites on their use. Despite this, they are able to provide immediate, actionable insights to performance engineers and software developers.

1.2 Thesis Contributions

This thesis makes the following specific contributions:

- New metrics are developed to guide and assess energy-aware code optimisations. In the absence of better alternatives, performance engineers have turned to metrics developed by the hardware community. These hardware metrics are ill-suited to software optimisation, and the lack of standardisation makes comparing results between studies impossible. This thesis seeks to address both of these shortcomings by introducing a common set of metrics along with rigorous justification of their utility.

- This thesis presents the Power Optimised Software Envelope (POSE), a model which helps performance engineers to determine whether energy or runtime optimisation will provide the greatest benefits for their code. The POSE model is platform agnostic, meaning it can be applied to any hardware and at any scale.

- The POSE model is extended to provide a model for system-wide power optimisation characteristics. System Summary POSE is able to derive upper limits for the benefit of energy-aware software optimisation on a given system. This allows developers to determine how amenable a system is to energy optimisation and hence whether it may be worth pursuing on their chosen platform in general, independent of any specific codes.

1.3 Thesis Overview

The remainder of this thesis is structured as follows:

Chapter 2 provides an account of core concepts, techniques and terminology employed in the field of HPC. Contemporary performance engineering tools and practices are described, and their suitability for energy-aware software optimisation is assessed. This chapter includes an overview of relevant performance engineering literature.

Chapter 3 details the evolution of parallel computing hardware with an emphasis on energy efficiency. The problems which motivate this work arise because hardware development is failing to maintain past trends in power consumption. This chapter provides a generalised model of hardware power consumption, and introduces features found in modern processors designed to minimise it. A key aim of this chapter is to highlight various ways in which performance engineers can influence energy efficiency.

Chapter 4 examines the metrics currently used to guide energy-aware performance optimisation. A good metric should provide meaningful values for a single experiment, allow fair comparison between experiments, and drive optimisation in a sensible direction. This chapter shows that established metrics are unable to fulfil these basic requirements then proposes new metrics which can.

Chapter 4 concludes with theoretical and empirical proofs of the advantages of these new metrics over established alternatives.

Chapter 5 introduces the POSE model. POSE serves as a preliminary “first cut” modelling technique intended to guide energy-aware optimisation efforts. This model presents an asymptotic analysis of the scope a code has for optimsation in both the power and runtime domains. By identifying the limits of each approach, POSE allows performance engineers to make informed decisions about where to focus their efforts in order to achieve the best results.

Chapter 6 builds on previous chapters by extending POSE to model system-wide optimisation criteria. Conventional POSE models use the runtime and energy costs of a code to calculate the scope that code has for power and runtime optimisation on a given system. Conversely, System Summary POSE is a meta-heuristic which determines the range of results POSE models could produce for a given system. This bound-of-bounds analysis places limits on the system-wide scope for power optimisation independent of any specific codes.

Chapter 7 concludes this work with a summary of results and contributions made, and discusses their implications for performance engineers. It also considers the future direction of energy-aware performance engineering and provides an overview of ongoing and future research.

CHAPTER 2

Energy-Aware Performance Engineering

This chapter introduces core concepts, techniques and terminology in the field of High Performance Computing (HPC). These topics are divided into seven areas, namely: Architectures, Measurement, Metrics, Benchmarking, Profiling, Modelling, and finally Optimisation. Energy-aware performance engineering requires new developments to be made in each of these areas. Recent developments are discussed, and areas where progress is lacking are highlighted.

2.1 Architectures

In simplest terms, supercomputers are nothing more than large collections of processing elements working together to solve complex problems [7]. This description is general enough to encompass the wide range of architectures which have been used to construct HPC systems over the years.

Flynn's taxonomy classifies computer architectures based on how many instructions and data items they can handle concurrently [39]. In this taxonomy, Single Instruction Single Data (SISD) architectures are those which do not exhibit any kind of parallelism. Single Instruction Multiple Data (SIMD) architectures execute their instructions sequentially, but each instruction operates on multiple data elements in parallel. Multiple Instruction Single Data (MISD) architectures execute multiple instructions on the same piece of data in parallel. Finally, Multiple Instruction Multiple Data (MIMD) architectures execute multiple instructions on their own independent data in parallel.

Modern supercomputers predominantly use MIMD architectures [102]. These systems can be divided into Distributed Memory Machines (DMMs) and Shared Memory Machines (SMMs), depending on how their memory is organised.

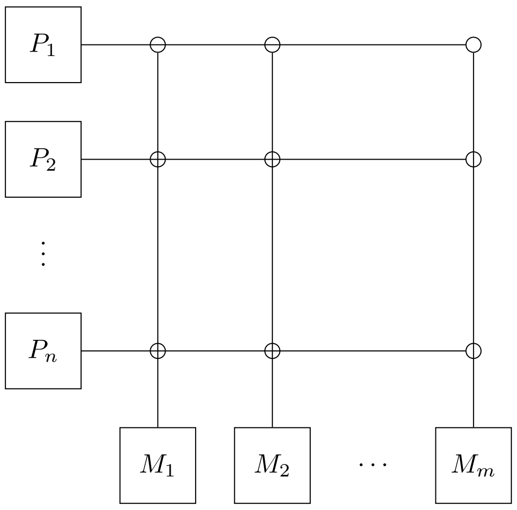


Figure 2.1: Crossbar Network Topology

DMMs consist of multiple compute nodes connected to each other through a shared interconnect. These nodes are independent units with their own processors, memory and peripherals. Data held by a node in local memory is private and cannot be accessed directly by other nodes. Explicit message passing protocols are used to allow groups of nodes to collaborate and share data.

SMMs consist of multiple processors all connected to a large pool of shared memory made up of many discrete memory modules. A common address space allows processors to access shared data transparently, regardless of its physical location. SMMs are further sub-divided into Symmetric Multiprocessing (SMP) and Non-Uniform Memory Access (NUMA) machines.

Processors in SMP machines have fast access to all areas of shared memory. Conceptually, all $M \times N$ pairs of memory modules and processors are connected by a flat network topology like the crossbar network shown in Figure 2.1. The scalability of SMP machines is limited by resource contention and the need for expensive, densely connected interconnects.

NUMA systems improve scalability by giving processors faster access to their own local memory. Other machines can still access this memory, however they must do so over a network. This reduces contention for applications which exhibit data locality as only remote memory accesses travel over the network.

Flynn's taxonomy also applies at the level of individual processors. Modern CPUs support both SIMD and MIMD operations through vector instructions and multiple cores respectively. There are no MISD implementations in widespread use for HPC, however some Field-Programmable Gate Array (FPGA) design patterns come close [5].

An ongoing trend in HPC is the shift towards *heterogeneous* computing [77]. In addition to conventional CPUs, heterogeneous systems also incorporate specialised compute devices called *accelerators* or *coprocessors* to handle particular tasks. Devices like Graphics Processing Units (GPUs), FPGAs and Intel's Xeon Phi coprocessors can be used to speed up execution of computationally intensive codes while also reducing system energy consumption [35].

2.2 Measurement

Accurate measurement is fundamental to performance engineering. Processors incorporate built-in clocks to maintain synchronisation and schedule interrupts. Engineers can use these clocks to measure the runtime performance of their code. Energy monitoring capabilities are also appearing in new processor designs.

Energy is the integral of power over time, or $E = \bar{P}t$. Energy consumption cannot be measured directly as a consequence, and must instead be calculated from measurements of power draw and time.

Various methods have been used to measure power draw in HPC systems, both at system and component levels. One approach uses thermal cameras to measure the temperature of different components and hence estimate their power draw. This works because the energy used by computers is converted to waste heat in accordance with the first law of thermodynamics. Mesa-Martinez et al. used thermal cameras and custom heat sinks to measure Central Processing Unit (CPU) power consumption [92], while Hackenberg et al. followed a similar approach to measure system-wide power consumption [48].

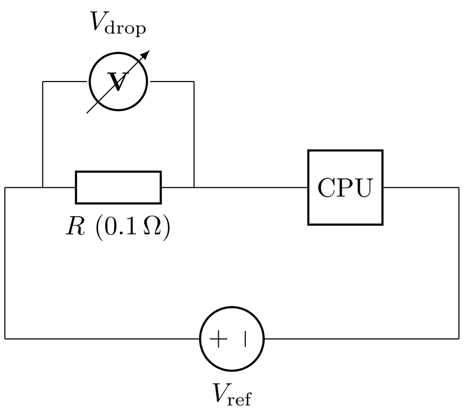


Figure 2.2: PowerMon Current Sense Resistor Circuit

The main advantage of this approach is its high spatial resolution; thermal imagery is able to show how power draw varies between components, and even across different areas of the same component. One disadvantage is poor temporal resolution; materials absorb heat and release it over time, meaning thermal emissions correspond to a moving average of power consumption. Another disadvantage is the requirement for expensive cameras and custom heat sinks, which makes this approach prohibitively costly when applied at scale.

Higher temporal resolution can be obtained at relatively low cost by instrumenting computing platforms with dedicated power sensors. Bedard et al. developed PowerMon, a scheme for measuring component-level power draw in commodity systems [10]. PowerMon works by measuring the voltage drop V_{drop} across resistors placed inline between the power supply and other system components. These resistors are calibrated to ensure they provide a particular resistance R (0.1 Ω is typical). Figure 2.2 shows a simplified circuit diagram for their apparatus.

$$I = \frac{V}{R} \tag{2.1}$$

PowerMon uses Ohm's law as stated in Equation 2.1 to calculate current flow I through a resistor based on V_{drop} and R. Resistors used to measure current flow in this manner are referred to as ‘sense’ or ‘shunt’ resistors. The same amount of current flows through both the sense resistor and the component being measured because current is conserved throughout series circuits. Furthermore, supply voltage V_{ref} takes on a known value depending on the type of component being powered. The Advanced Technology eXtended (ATX) standard mandates power supply output voltages of 12 V, 5 V, 3.3 V and −12 V, for example [23]. Component power draw can then be calculated as the product of I and V_{ref} as per Equation 2.2.

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

Sense resistors have three major drawbacks when used in HPC systems. First, some energy is lost as heat within the resistor, increasing overall power draw. Secondly, their resistance varies with temperature, limiting accuracy and introducing non-linearities in their results. Finally, they require direct electrical connections to the power supply and the component being measured. Any short-circuits or other manufacturing defects could easily damage sensitive hardware.

An alternative approach to power measurement relies on the magnetic fields induced when current flows through a wire. Ampere’s law for straight conductors, given by Equation 2.3, states that the magnetic field strength $|\vec{B}|$ at distance r from the wire is proportional to the original current I. The constant μ corresponds to the magnetic permiability of air.

$$|\vec{B}| = \frac{\mu I}{2\pi r} \tag{2.3}$$

Hall effect sensors measure magnetic field strength, and can therefore be used to determine current flow while remaining electrically isolated from the conductor. Laros et al. developed PowerInsight, a production quality power monitoring platform which uses Hall effect sensors rather than sense resistors to improve accuracy and reliability [81]. Equation 2.2 is again used to calculate power consumption from current measurements.

Hackenberg et al. instrumented a large HPC cluster called Taurus with commercial power sensors which exploit the Hall effect. The resulting High Density Energy Efficiency Monitoring (HDEEM) infrastructure can be used to measure component-level power and energy consumption across large numbers of nodes at high sample rates [51].

Intel introduced Running Average Power Limit (RAPL) to support power-aware frequency scaling in their Sandy Bridge Processors [27]. As a side effect, performance engineers gained access to an interface capable of reporting CPU power consumption. RAPL exposes a number of Model Specific Registers (MSRs) which can be read by user code to determine the rate of power draw. Early versions of RAPL were model based, but more recent processors incorporate dedicated power sensors.

Other manufacturers have also added power measurement capabilities to their hardware. AMD added a scheme similar to RAPL with equivalent functionality starting with their Bulldozer CPUs [1]. Similar schemes also exist for GPUs [18] and Xeon Phi [82] hardware.

HPC Vendors and system integrators are also beginning to include power monitoring capabilities in their products. Cray's XC line of supercomputers expose energy measurements through systems [55]. Similarly, IBM servers incorporate current measuring hardware based on sense resistors which can be read using their Amester tool [16].

2.3 Benchmarking

Modern processors include hardware designed to accelerate specific operations. Vendors quote *peak performance* figures which assume that all these hardware features can be kept fully occupied. In practice, applications only perform a subset of the relevant operations, and memory bandwidth limits often prevent those features which are used from achieving maximum throughput.

Benchmarks are programs designed to collect real-world performance data.

Performance benchmarks serve a dual purpose. First, benchmarks can be used to measure and compare the performance realistically achievable by different machines and architectures. Secondly, benchmarks serve as good platforms to investigate the effects of different optimisations in a controlled manner.

Micro-benchmarks are simple programs designed to target specific aspects of system performance. Linpack is a well known micro-benchmark which measures a system's capacity for sustained floating point throughput [30]. Linpack results form the basis of the Top500 and Green500 supercomputer rankings [38]. Other micro-benchmarks include STREAM [90], which measures memory bandwidth; SKaMPI, which measures network performance [4]; and IOR, which measures file system performance [110].

Application benchmarks are larger programs which test how well systems handle complex applications. They are often built from simplified versions of production applications in order to ensure realistic workloads. The Mantevo project is a suite of application benchmarks developed at Sandia National Laboratories [56] and used extensively throughout this thesis.

Existing benchmarks can be repurposed for power and energy studies [72], and dedicated power benchmarks have also started to emerge. One example is FIRESTARTER [50], a micro-benchmark specifically designed to trigger near-peak power consumption across a range of x86_64 CPUs and NVIDIA GPUs. It contains hand optimised assembly routines which raise processor activity above the level attainable with high level languages. A small assembly micro-benchmark designed to minimise power consumption while keeping CPUs active is also described in this thesis.

SPECpower is a commercial benchmarking suite which provides an application benchmark called "SPECPower_ssj2008" along with a framework for measuring application energy efficiency and performance [79]. The SPECpower benchmark is a Java program which simulates a transactional workflow running under varying amounts of load. It is designed to mimic the behaviour of common enterprise computing applications such as web servers or relational databases which must handle bursts in utilization. SPECpower measures performance at 11 different target loads, starting at 100% utilization and reducing this in steps of 10% until it reaches idle.

2.4 Metrics

Performance engineers use metrics to assess the performance of HPC hardware and software. Individual metrics capture particular properties of a system under investigation. Some of these properties can be measured directly, while others must be derived from multiple observations.

Metrics enable meaningful comparison between different platforms and can be used to quantify the effects of code changes. They can be divided into two categories depending on the types of comparison they allow; namely Figure of Merit (FoM) and Non-FoM metrics.

2.4.1 Figure Of Merit Metrics

Some metrics act as utility functions which measure the cost of running different programs. These FoM metrics can be used to rank different implementations of the same algorithm in order to identify valid optimisations [53]. Runtime and energy consumption are both examples of FoM metrics.

Until recently, runtime optimisation was ubiquitous in HPC while energy optimisation has been confined to domains like embedded systems and mobile robotics. Although energy consumption is becoming a constraint for scientific computing, minimising runtime is still an important optimisation objective.

Optimising software according to multiple properties simultaneously is known as Multi-Objective Optimisation (MOO). MOO requires FoM metrics that strike the right balance between the potentially conflicting requirements imposed by different optimisation objectives.

Gonzalez et al. proposed Energy Delay Product, a FoM metric which combines the energy and runtime costs incurred by processors [45]. Martin et al.

generalised this into the Et^n family of FoM metrics, with parameters E and t corresponding to energy and time [88]. They argue that Et^2 provides the best balance for microprocessor design. Srinivasan et al. reached the same conclusion, although for slightly different reasons [113].

Many authors have adopted these metrics from the hardware community and applied them to software optimisation problems. Vincent et al. describe a technique which minimises Et^1 using CPU throttling [41]. Bingham and Greenstreet use Et^n metrics to analyse runtime constraints imposed by a fixed energy budget for various algorithms [12]. Laros et al. use Et^n metrics to assess a number of production applications and state that Et^3 strikes the right balance between runtime and energy for HPC [80]. Et^1 has also been used extensively to quantify the efficiency of resource provisioning and scheduling in cloud computing environments [107, 122].

Bekas and Curioni further generalised Et^n metrics to the form $E \cdot f(t)$, a product between energy and an application dependent function of time [11]. They argue that this formalisation is able to drive software optimisation, assuming an appropriate application specific function $f(t)$ can be identified.

Chapter 4 covers these metrics in more detail. In particular, it shows that metrics originating from the hardware community are not suitable for measuring software performance. It goes on to introduce new metrics which are designed to support energy-aware performance optimisation.

2.4.2 Non-Figure of Merit Metrics

Although FoM metrics are required to identify optimisations, non-FoM metrics also play an important role in performance engineering.

Instructions Per Second (IPS) was an early measure of processor throughput. Although it makes intuitive sense, this metric does not allow comparison between different architectures. Reduced Instruction Set Computing (RISC) processors may need several instructions to perform the same operation as a single instruction on a Complex Instruction Set Computing (CISC) processor, for example. RISC and CISC processors exhibit different levels of performance at the same IPS rate [66].

Floating Point Operations per Second (FLOPS) is a metric designed to address some of the deficiencies of IPS. It quantifies performance in a portable manner by counting basic arithmetic operations (addition, subtraction, multiplication, division and the like) rather than platform specific instructions. FLOPS captures the throughput of arithmetic operations, or equivalently the rate at which an application converts runtime into floating point results. As a result it can give a better indication of real world performance than IPS, especially on the numerically intensive codes common in HPC.

A related metric is FLOPS per Watt, which combines the number of Floating Point Operations per Second with the rate of power consumption. Despite its name, this metric is quoted in units of Operations per Joule (1 Joule is defined as 1 Watt-Second). While conventional FLOPS measures the number of operations carried out per second elapsed, FLOPS per Watt counts the number of operations carried out per Joule of energy consumed. In effect, FLOPS per Watt measures how effective an application is at converting energy into floating point results.

More recent developments in energy-aware metrics include Power Usage Effectiveness (PUE) and Information Technology Power Usage Effectiveness (ITUE). PUE is the ratio of energy used by computer hardware to total facility energy consumption, which also includes secondary functions like cooling, lighting and power supply losses [87]. A PUE of one is optimal as this would suggest that all energy is being used by computer hardware to complete primary tasks with none being lost to overheads.

A drawback of PUE noted by Patterson et al. is that it treats all energy consumed by computer hardware the same. They contend that this simplification is problematic for HPC, where large systems typically have extensive cooling and power delivery subsystems integrated within them. Their solution is to extend PUE to consider internal subsystems. They call their metric ITUE, which they describe as “PUE inside the IT”. ITUE is defined as the ratio of energy used for compute to total energy use by computer hardware [99].

Metrics are also used to measure the parallel performance and scalability of code. The speed-up S_n observed by running a program on n processors in parallel is defined as the ratio between its serial runtime T_1 and parallel runtime T_n as shown by Equation 2.4:

$$S_n = \frac{T_1}{T_n} \tag{2.4}$$

A program that runs n times faster on n processors is said to exhibit *linear speed-up*. This is the maximum possible speed-up which can be attributed to increased processing power. Linear speed-ups are uncommon because they require a code which can be split into multiple independent tasks without any additional overhead being introduced.

Super-linear speed-ups sometimes occur when serial runtime is limited by factors other than processor throughput [120]. A typical example would be a large simulation exhausting memory and causing *thrashing* as data is repeatedly paged out to disk. Adding nodes will increase the available memory and reduce thrashing, resulting in a super-linear speed-up. It is worth noting that these super-linear speed-ups will cease once the entire simulation fits into memory.

Parallel efficiency measures how well a code makes use of the available hardware. This metric is calculated by dividing total speed-up by processor count, and can therefore be interpreted as per-processor speed-up:

$$E_n = \frac{S_n}{n} = \frac{T_1}{n \cdot T_n} \tag{2.5}$$

Most scientific computing workloads require communication and synchronization between tasks on different processors. These secondary operations increase runtime overheads without contributing to the calculation of results. Codes with low parallel overheads are said to be efficient, while codes which spend much of their time dealing with these overheads are said to be inefficient. The maximum value for efficiency is one, which corresponds to a linear speed-up.

2.5 Profiling

Profilers are tools which measure performance characteristics over one or more runs of a target application. Software developers use these tools to identify performance bottlenecks in their code. Profilers are categorized as either event-based or sampling, depending on their approach to collecting measurements.

Event-based profilers like VampirTrace [96] measure application state each time a specific event occurs. Samples may be taken when a specific function is called, or when memory is allocated, for example. Runtimes are calculated from timestamps within each sample. Additional metrics like performance counter readings or power consumption may also be recorded.

Profiling events can be specified in several ways. The most direct approach is for developers to manually instrument their code with profiling hooks. Other approaches perform instrumentation at compile time, link time, run time, or a combination of all three.

Event-based profilers take measurements every time a sampling event occurs, making them excellent for capturing detailed traces. Although they are good at timing specific functions, they are less useful for identifying which functions are causing performance issues in the first place. Doing so would require every function call to be instrumented, but this would severely impact performance and lead to skewed results [94].

Statistical profilers sample program state at regular intervals to build up a summary of program behaviour. How often a particular code path is encountered during sampling reflects its overall contribution to runtime.

The accuracy of statistical profilers depends on their sampling frequency. If this is set too high then application performance will suffer, invalidating the results. If it is too low then important details may be missed entirely. It is also important to prevent sampling periods from becoming synchronized with periodic events inside an application. One strategy to avoid these aliasing effects is to offset each sample by a random delay.

Statistical profiling is not limited to working in the runtime domain. Some profilers can also operate in periods determined by hardware performance events like cache misses, instructions retired and memory writes. Tools like Perfmon2 [36] can be configured to take samples every time a set number of events has occurred. Intel and AMD chips include hardware support for this through their respective Precise Event-Based Sampling (PEBS) and Instruction Based Sampling (IBS) technologies [117].

The ability to sample in domains other than time allows performance engineers to analyse different aspects of their code's performance. For example, samples taken at fixed increments of cache misses will tend to cluster around code which stresses memory subsystems. If a performance engineer knows their code is memory bound, they can perform this kind of analysis to find optimisation targets which would otherwise be missed.

Statistical profilers operating in intervals of energy would be able to produce a breakdown of energy costs by code path. The PAPI library attempts to provide this functionality [118]. Because energy cannot be measured directly, however, this approach requires profilers to repeatedly sample power draw in order to calculate cumulative energy consumption. This is equivalent to taking samples at short runtime intervals, and then sub-sampling from these based on estimated energy consumption.

The sampling distribution observed from such 'hybrid' approaches is not representative of either energy consumption or runtime. Code paths missed by runtime sampling will never show up in the final results regardless of how much energy they consume. Higher sampling frequencies would reduce this source of error, but would increase sampling overhead leading to skewed results.

Stochastic samplers offer a possible solution which avoids the need to calculate energy altogether. Rather than relying on fixed sampling intervals, samples are taken with probability $p < 1$ each clock cycle. For fixed p, this scheme produces a runtime sampler with an average sampling interval of $1/p$ cycles. Alternatively, if p was proportional to instantaneous power draw, then the distribution of samples would correspond to per-instruction power draw. Combining per-instruction power and runtime figures would produce an accurate picture of instruction-level energy consumption.

2.6 Modelling

Performance engineers use models to reason about the performance of their codes in a number of ways. First, they can help developers identify factors contributing to poor performance. Secondly, they can be used to predict application performance and scalability characteristics. Finally, they can be used to estimate the performance implications of new hardware architectures.

2.6.1 Heuristic Modelling

Heuristic models provide simplified analogies which help developers understand the performance of their code. This is the most abstract approach to performance modelling as no attempt is made to faithfully represent real systems. Models in this category ignore implementation details in favour of generality, and usually focus on a single aspect of system behaviour.

Understanding how different factors impact performance is the first step towards targeted optimisation. Their ability to produce clear insights without extensive benchmarking or profiling means heuristic models are well suited to the early stages of optimisation.

Arguably the best known heuristic performance model is Amdahl's Law [8], which states that parallelisation gains are limited by the serial portions of a code. A program's serial runtime T_1 can be broken down into W_s time spent performing inherently serial work and the remaining W_p time spent performing work which could be parallelised:

$$T_1 = W_s + W_p \tag{2.6}$$

Running a program across n processors in parallel will reduce W_p while leaving W_s unchanged, assuming the program is efficiently parallelisable. Excluding the possibility of super-linear speed-ups yields the following expression:

$$T_n \geq W_s + \frac{W_p}{n} \tag{2.7}$$

Amdahl's law is obtained by substituting Equation 2.6 and Equation 2.7 into the definition of speed-up given by Equation 2.4 to yield Equation 2.8 below. It can also be defined in terms of the serial fraction of a code f_s, where $W_s = f_s\, T_1$ and $W_p = (1 - f_s)\, T_1$, resulting in Equation 2.9:

$$S_n \leq \frac{W_s + W_p}{W_s + W_p/n} \tag{2.8}$$

$$\Leftrightarrow \quad S_n \leq \frac{1}{f_s + (1 - f_s)/n} \tag{2.9}$$

Amdahl's law states that parallel speed-up is limited for all codes with $f_s > 0$, even given access to an unlimited number of processors. Figure 2.3 shows how the speed-ups given by Equation 2.9 quickly reach a plateau even for codes with relatively tiny ($f_s = 0.1\,\%$) serial portions.

Figure 2.4 illustrates an important corollary of Amdahl's law. The parallel efficiency (given by Equation 2.5) of any code with a serial portion will always decrease as more processors are added.

Amdahl's law only takes two parameters, yet despite this simplicity it is able to provide valuable insights into application scalability. If application performance follows Amdahl's law at high processor counts then serial code is the biggest barrier to scalability. Conversely, if observed performance is worse than predicted, then parallel overhead is likely to blame.

Amdahl proposed his law in 1967 to demonstrate "the continued validity of the single processor approach and of the weaknesses of the multiple processor approach" [8]. Despite this, the multiple processor approach went on to become a significant driver of performance improvements in HPC.

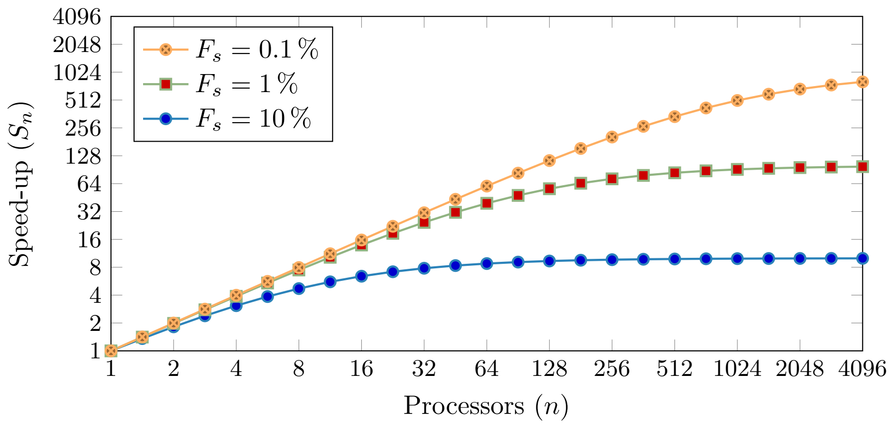


Figure 2.3: Amdahl's Law Speed-up Limits

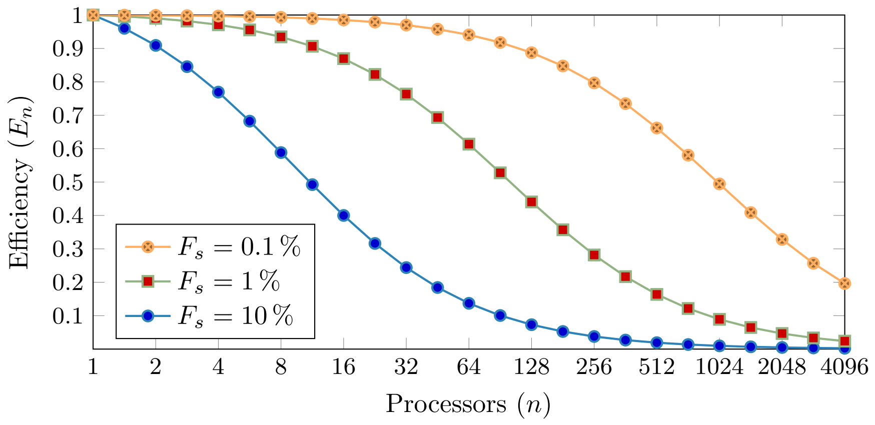


Figure 2.4: Amdahl's Law Efficiency Limits

Gustafson resolved this apparent paradox by observing that problem sizes tend to grow to fill the available computing power [47]. This is because for many HPC codes larger data sets translate to improved resolution, accuracy or scale. Scientific computing workloads typically involve a serial setup phase followed by repeatedly performing the same calculation on each element of a dataset in parallel [9]. Most of the extra work associated with larger data sets can be parallelised, meaning W_p increases faster than W_s. A smaller fraction of runtime is spent on serial code when this observation holds, and speed-ups improve as a result.

Amdahl's and Gustafson's laws are both valid, and the choice of which to use depends on circumstances. That said, performance engineers are often tasked with optimising code for a specific platform and problem size. They cannot rely on arbitrarily large problem sizes and processor counts, and are therefore bound by the limits of Amdahl's law in most cases.

Amdahl's and Gustafson's laws only consider perfect parallelism, which applies when tasks can be executed independently and in any order. Imperfect parallelism happens when dependencies impose partial orderings on the tasks performed by a parallel algorithm. While tasks in the same sequence must be executed in order, multiple independent sequences can be processed in parallel.

The work-span model represents algorithms as a set of tasks connected by their dependencies to form a Directed Acyclic Graph (DAG). Time T_1 is called an algorithm's *work*; the cumulative runtime of all its sub-tasks. Time T_∞ is called an algorithm's *span*; the total runtime of its *critical path*. The critical path is the longest chain of tasks which must be executed sequentially. Algorithms can never run faster than T_∞, even with access to unlimited processors.

Figure 2.5 shows an example of the work-span model in which all tasks take unit time. This example does 15 units of work, one for each task, and has a span of 6, one for each task on the critical path highlighted.

The work-span model provides two bounds on parallel performance. The

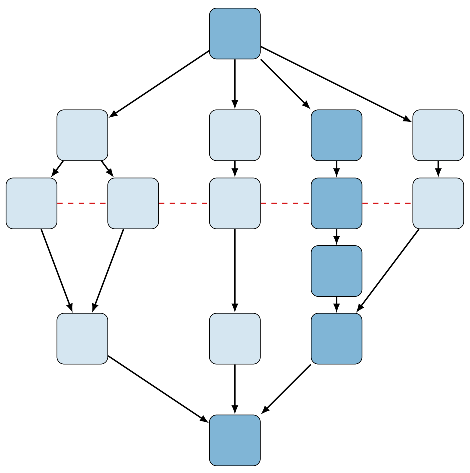


Figure 2.5: Work, Span and Maximum Cut

ratio of T_1 and T_∞ provide the upper bound to speed-up shown by Equation 2.10:

$$S_n \leq \frac{T_1}{T_\infty} \tag{2.10}$$

The bound given by Equation 2.11 comes from examining the best case scenario, that all $T_1 - T_\infty$ work off the critical path can be perfectly parallelised.

$$\begin{aligned} & T_n \geq T_\infty + \frac{T_1 - T_\infty}{n} \\ \Leftrightarrow \quad & T_n \geq \frac{T_1 + (n-1)\, T_\infty}{n} \\ \therefore \quad & S_n \leq \frac{n\, T_1}{T_1 + (n-1)\, T_\infty} \end{aligned} \tag{2.11}$$

Work span DAGs can also be used to find the maximum degree of parallelism exhibited by an algorithm. The maximum cut of the DAG (or more precisely of its conjugate, i.e. cutting across nodes rather than edges) corresponds to the largest number of tasks which can be executed concurrently. The red dashed line in Figure 2.5 shows that at most 5 processors can be kept active at any one time. Any processors added above this limit will remain idle.

Roofline is a more recent heuristic model which frames application performance in terms of two system bottlenecks, namely off-chip memory bandwidth

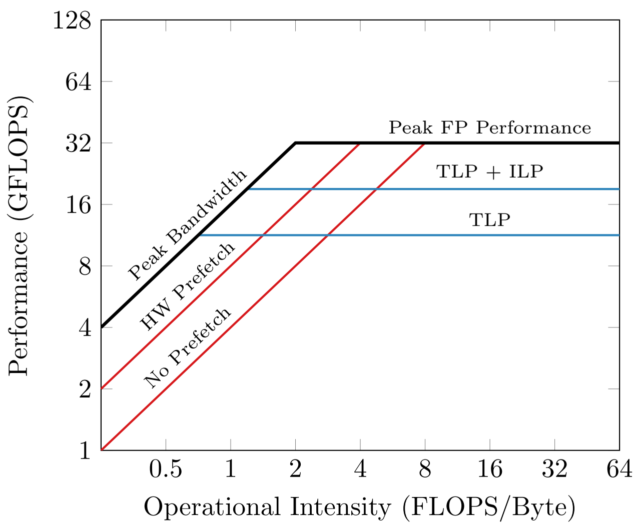


Figure 2.6: Example Roofline Model

and floating point performance [121]. *Operational intensity*, the ratio between work done and memory traffic, is then used to determine whether a code is compute or memory bound.

Figure 2.6 shows an example of the Roofline model. Horizontal lines correspond to floating point performance limits and diagonal ones to memory bandwidth limits. A Roofline consists of one performance and one memory limit. Platforms can exhibit many different Rooflines depending on which hardware performance features can be used.

The Floating Point (FP) limits shown in Figure 2.6 are: Thread Level Parallelism (TLP), corresponding to the maximum performance of simple multi-threaded programs; TLP plus Instruction Level Parallelism (ILP), the maximum performance of multi-threaded programs which use hardware features like SIMD vectorization; and peak floating point performance, the maximum performance of threaded, vectorised programs with the right instruction mix to keep CPU functional units fully occupied.

The memory bandwidth limits shown in Figure 2.6 are: no prefetching, corresponding to random memory access patterns that cannot be predicted;

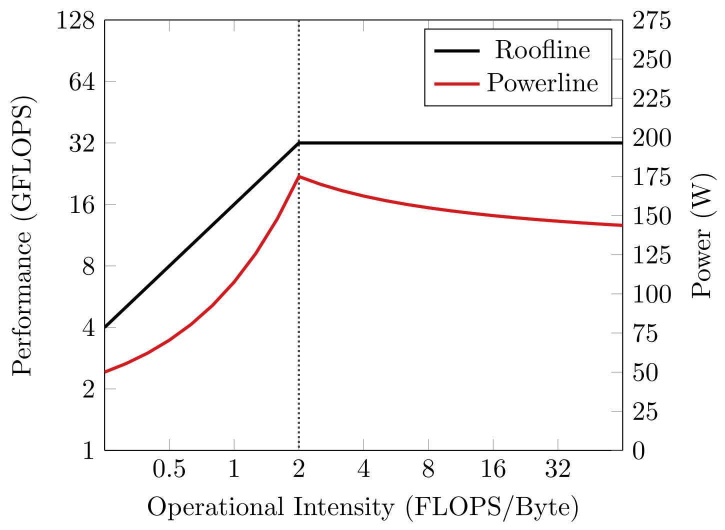


Figure 2.7: Powerline Model

hardware prefetching, whereby hardware is able to predict upcoming memory accesses and preload the data; and peak bandwidth, the maximum possible bandwidth attainable with perfect hardware prefetching and optimal memory layout and access patterns.

Roofline models are able to diagnose performance issues using easily obtainable information. Developers identify where their code appears on a Roofline diagram by measuring its operational intensity and floating point performance. This allows them to determine which performance limit their code is bounded by, and hence whether to look for runtime or memory optimisations.

In the example shown by Figure 2.6, improving the memory performance of codes with operational intensities above eight FLOPS per byte will not reduce their runtime. Even the lowest bandwidth limit is enough to keep a CPU fully supplied with data beyond this point. Conversely, memory bandwidth should be the sole optimisation target for codes with operational intensities under one FLOPS per byte. Between these limits the best course of action depends on the level of floating point performance observed.

Choi et al. extended the Roofline model to identify the algorithmic conditions necessary for trade-offs between runtime and energy [19]. Figure 2.7 shows the distinctive shape of their ‘Powerline’ model and how it compares to conventional Roofline analysis. In particular, it shows how power consumption peaks when operational intensity places equal demands on memory and floating point performance. This is because with both subsystems under equal load, neither one can become a bottleneck and force the other to enter an idle state waiting for more work. Idle subsystems draw less power, so power consumption drops off when either subsytem is forced to spend periods of time idle.

The Power Optimised Software Envelope (POSE) model developed in Chapter 5 is another example of energy-aware heuristic performance modelling.

2.6.2 Analytical Modelling

Analytical models distil the structure and behaviour of a program into a set of parameterised mathematical expressions. Performance predictions are then obtained by solving these expressions for the required input parameters. Analytical models are able to predict the behaviour of real systems in a short amount of time, making them particularly suitable for parameter studies.

Early analytical modelling approaches created bespoke models specific to individual machines and applications. These approaches fell out of favour because of the considerable time and expertise required to model complex systems accurately. Furthermore, the resulting models were not portable and had to be completely rebuilt for each new platform. Modern approaches provide generalised model skeletons which can be tailored to individual applications.

The Parallel Random Access Machine (PRAM) framework was one of the first modelling techniques to produce portable performance models. PRAM defines an idealised representation of SMP hardware consisting of n processors with perfectly synchronised clocks [40], as shown in Figure 2.8. Each processor has its own private memory and can access global shared memory through a common memory access unit. Processors perform one instruction each clock cycle, including potentially reading from or writing to shared memory.

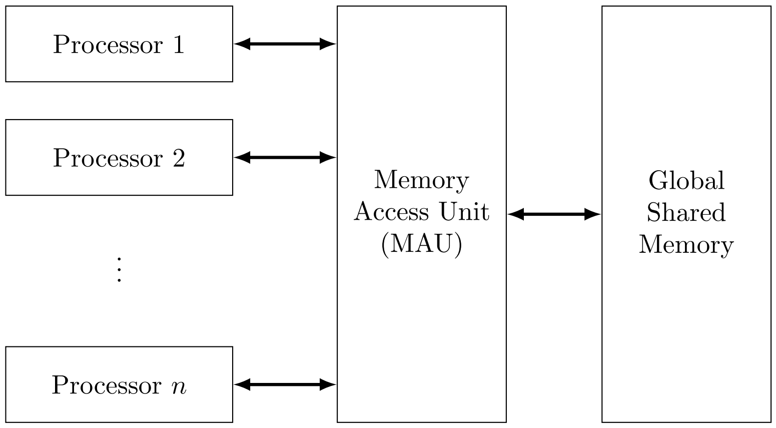


Figure 2.8: PRAM Abstract Machine Model

Conflict resolution policies define what happens when multiple processors access the same memory location simultaneously. There are four policies to choose from, namely: Exclusive Read Exclusive Write (EREW), Concurrent Read Exclusive Write (CREW), Exclusive Read Concurrent Write (ERCW), and Concurrent Read Concurrent Write (CRCW). Both ERCW and CRCW have sub-policies to determine which concurrent write access succeeds.

The PRAM model assumes that all processors are synchronised and communication between processors is free. These were reasonable assumptions when uniform memory access SMP machines were common in HPC. The scalability of these systems is limited by resource contention, however, and they have largely been replaced by NUMA and message-passing DMM approaches.

PRAM emphasised the importance of model portability, however its models are limited to SMP machines. The LogP model was devised to model parallel applications regardless of the computer architecture used [24].

LogP is named after its four system parameters: L, which models network latency; o, the overhead of sending and receiving messages; g, the minimum gap between messages, or equivalently the reciprocal of inter-node bandwidth; and P, the number of processors or nodes. LogGP extends the original model with G, a parameter which captures the higher bandwidth available for longer messages and bulk transfers on many systems [6].

Some analytical models use hardware performance counters to estimate system power consumption. Power usage and performance events are recorded for a selection of benchmark programs. Regression analysis is then used to derive power costs for each category of performance event. This approach has been used to develop power models for components like CPUs [14, 69], GPUs [62], and Xeon Phi coprocessors [111], as well as entire supercomputer systems [13].

Despite its popularity, this approach has significant limitations. Processors can only monitor a small number of performance counters simultaneously, so many events will be missed. Furthermore, processor events are not standardised between processors, limiting the portability of these models. Lively et al. demonstrated this fact and proposed code-specific power models as a solution [86]. In effect, they suggested intentionally over-fitting models to particular target applications and platforms.

2.6.3 Simulation

Analytical performance modelling involves constructing detailed models of application behaviour. Every application requires its own customised model, even with modern frameworks, and these models must be continually updated and revalidated in response to code changes. Simulators avoid these issues by taking applications themselves as input, either directly or in the form of profiler traces.

Simulators gather performance data by running some representation of the target application through a detailed model of a computer system. This shifts the burden of model construction and verification away from performance engineers and towards simulator designers. Once validated, a simulator can be used to model the performance of many different applications.

Simulators are categorised based on the granularity of their system models. Hardware simulators model the low-level operation of computer systems in as much detail as possible. Cycle-accurate simulators are able to mimic hardware down to the level of individual clock cycles. This amount of detail is useful when designing new hardware or assessing the impact of exotic architectures.

Discrete event simulators operate at a higher level of abstraction, modelling system behaviour as a sequence of distinct states. State transitions are triggered by application events like network communications or synchronisation barriers. Profilers gather event traces for a target application, which are then passed as input to the simulator. Discrete event simulators are useful for 'what if' investigations which assess the likely outcomes of different scenarios.

Simulations are the most detailed approach to performance modelling, but this detail comes at significant runtime cost. Event-based approaches require traces to be gathered by running the original application in full. Hardware approaches take even longer as they run every application instruction through simulated hardware. Simulators like Sandia's Structural Simulation Toolkit (SST) offer a combined approach, providing hardware simulation for key components and falling back to an event based approach where less detail is required [67].

Hardware simulation is often used to model power consumption. Wattch is a popular framework for analysing and optimising microprocessor architectures for reduced power consumption [17]. McPAT is a similar tool which replaces the linear scaling assumptions in Wattch with non-linear power models, making it suitable for the post-Dennard era [85].

SST supports system power simulation via its modular architecture [64]. Existing component-level power simulators like McPAT are used as back-ends to model the power consumption of individual processors, which SST then aggregates to provide a system-level overview.

2.7 Optimisation

Performance optimisation involves modifying applications to improve properties like runtime or energy consumption. *Algorithmic optimisations* lead to more efficient algorithms irrespective of the platform used. Once these optimisations are exhausted, any further improvements come from tuning applications to better exploit the underlying hardware.

Examples in the latter category include: Multi-threading, where multiple threads are run simultaneously on different CPU cores; cache blocking, where loop iterations are re-ordered to make more efficient use of the cache hierarchy; and vectorisation, where SIMD instructions are used to improve floating point throughput. Hand-optimising code using these techniques can produce large performance improvements, however the resulting applications are platform-specific and can be hard to maintain [32].

Performance portability is the idea that optimised code should remain as general as possible in order to perform well across different platforms. When following this approach, developers expose opportunities for parallelism within their code. It is then left for optimising compilers and libraries to map these opportunities on to specific hardware features [100].

One way to achieve performance portability is to use frameworks which abstract away details of the underlying platform. OP2 allows users to develop unstructured grid applications independently of the underlying hardware. Back ends then translate these high level implementations into low-level code which targets a specific platform [43, 44]. Other examples of this approach include Kokkos [32] and Charm++ [71].

Compiler directives offer a more direct approach in which developers annotate their source code with additional information and instructions. These directives expose opportunities for parallelism and allow compilers to use more aggressive optimisation strategies. OpenMP [26] provides directives to exploit TLP via the *fork-join* model, and ILP using SIMD instructions. OpenACC [119] uses directives to specify functions for offload to GPUs and other accelerators.

2.7.1 Energy-Aware Optimisation

Energy use can be reduced either by shortening runtime or decreasing power consumption. The runtime optimisations described above are therefore also capable of reducing energy consumption. Power optimisation is less developed, however some progress has been made.

Dynamic Voltage and Frequency Scaling (DVFS) and sleep states are two hardware features often exploited by power optimisations. Briefly, DVFS allows processors to run at different clock speeds and supply voltages, while sleep states allow processors to power down during periods of inactivity.

The work-span model described in Section 2.6 suggests several strategies for power optimisation. Nodes off the critical path can use DVFS to lower their clock speeds and reduce power draw [31]. Alternatively, they can temporarily increase their clock speeds to finish their work quickly before entering into sleep states [33]. Finally, DVFS-aware scheduling algorithms can be used to pack applications onto fewer nodes [84].

2.8 Summary

Performance engineering is a complex process during which developers rely on the tools and techniques listed above. Until recently, minimising runtime has been the main aim of performance engineering. Current tools share this runtime focus and must be updated to support the Multi-Objective Optimisation of both power and runtime as energy consumption becomes a limiting factor.

The next chapter examines hardware power consumption and the ways in which developers can influence it. Performance engineering tools will be required to help capitalise on the optimisation opportunities which arise from this ability to influence power consumption.

CHAPTER 3

Energy Efficiency in Computer Systems

Energy-aware performance engineers need to understand the factors which influence system power consumption. From a hardware perspective, the most important factors are the technologies used to fabricate digital circuits and the processor architectures which are built on top of them.

This chapter begins by charting the evolution of processor fabrication, from early valve-based systems up to the Complimentary Metal Oxide Semiconductor (CMOS) chips in use today. It then examines the relationship between digital circuit design and power consumption. This relationship is illustrated with a simple model of CMOS power draw. Various features of modern computer architectures are then introduced which performance engineers can leverage to improve the energy efficiency of their code. This chapter concludes by remarking on ongoing trends in energy efficient processor design.

3.1 Fabrication Technologies

Digital computers are based on circuits which use different voltage levels to represent discrete logic states. Binary computers use circuits with only two states, labelled *zero* and *one*. Conceptually, binary circuits take in patterns of zeros and ones as input, perform some calculations on them, then output new patterns which encode the result. This process requires the ability to control output voltages based on the values represented by input voltages.

Thermionic valves were the first practical technology which could be used to build digital computers. These devices consist of a heated cathode and an unheated anode sealed inside a vacuum tube. Heating a negatively charged cathode allows electrons to escape from its surface in a process known as thermionic emission. Once free, these negatively charged electrons are repelled by the cathode and attracted towards the more positively charged anode. This results in a stream of electrons flowing from the cathode to the anode. Current can flow along this stream, but not in the opposite 'upstream' direction. Valves which allow current to flow in one direction only are known as *diodes*.

Some valves also contain a third component called the *control grid*, which is situated between their anode and cathode elements. Electrons flow through the holes in this component when it is not electrically charged. When a negative charge is applied to the grid, however, it repels electrons and blocks current flow. This type of valve is called a *triode*, and is a forerunner to modern *transistors*.

Concerns about system power draw have existed since the dawn of computing, when early valve-based machines consumed as much energy as modern supercomputers. The Electronic Numerical Integrator and Computer (ENIAC) was the first electronic stored-program computer built in the US. It contained approximately 17,500 thermionic valves and weighed thirty tons [58]. When ENIAC was switched on in 1946 it had a peak power dissipation of 174 kW [89], a figure which would not look out of place in the current Top 500 list [3].

Thermionic valves were superseded in the 1950s and 60s by bipolar transistors packaged into ICs. ICs are etched onto semiconductor wafers in a process called *photolithography*. First, circular wafers are cut from a cylindrical ingot of monocrystalline silicon up to 30 cm in diameter. Multiple copies of a circuit are then etched onto the surface of these wafer before they are cut into individual pieces and packaged into chips.

Although they were slower, bipolar circuits consumed far less energy and were more reliable than equivalent valve-based designs. This led to the development of increasingly complex computers and a dramatic reduction in system power consumption. Over time, manufacturing improvements delivered ever smaller transistors, yielding rapid increases in both performance and power density. Ultimately this resulted in escalating power draw which threatened to halt the advance of computer technology [70].

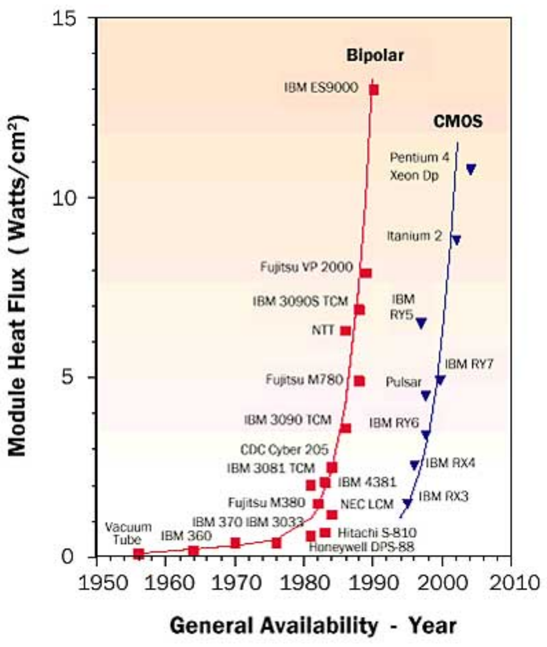


Figure 3.1: Power Density Trends, based on data from [22]

Bipolar transistors peaked in the early 1990s, before being replaced by the slower, more efficient CMOS technology in use today. Figure 3.1 shows how CMOS development is following the same trajectory, with escalating power draw again threatening to stall advances in High Performance Computing (HPC).

Slower, more energy efficient technologies replacing faster but more power hungry ones is a recurring theme in processor design. That said, at present there are no obvious replacements for CMOS as the dominant processor fabrication technology. Until a replacement can be identified, hardware designers are reliant on incremental improvements in CMOS fabrication and architectural innovations to deliver performance improvements. Examples of the former include the use of exotic high-κ materials [106] and non-planar "3D" transistors [60], while examples of the latter are presented below.

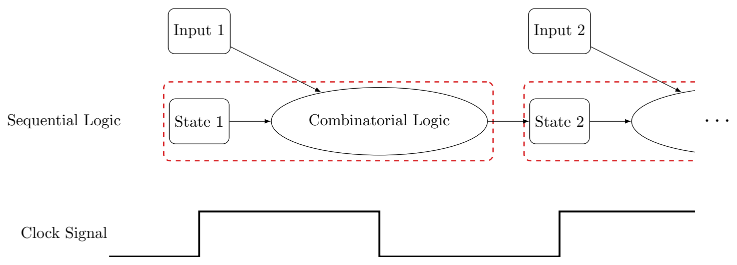


Figure 3.2: Synchronous Sequential Logic

3.2 CMOS Digital Logic

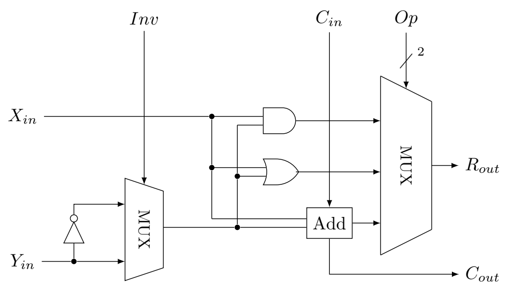
The remainder of this thesis deals with the energy used by modern computer hardware based on CMOS technology. CMOS processors are complex circuits which contain many distinct subsystems, each of which is built out of logic elements and, ultimately, transistors.

Processors include both *combinatorial* and *sequential* circuitry. Combinatorial circuits consist of many individual logic gates arranged in layers. Input cascades through each of these layers, changing along the way, before it finally emerges as output. Combinatorial logic is *stateless* because it contains no memory and its output is purely a function of its input.

The time it takes for a circuit to finish updating its output in response to new input is known as its *propagation delay.* The length of a propagation delay depends on how many logic gates are on a circuit's *critical path*, and how quickly the transistors which make up these gates are able to change state.

Sequential circuits augment combinatorial logic with *flip-flops* which are components capable of retaining state. This saved state is passed along with fresh inputs into blocks of combinatorial logic, enabling sequential logic to produce outputs based on many previous inputs.

The *synchronous* sequential circuits found in most processors update their state at discrete times in response to clock signals. The clock period used to drive these circuits must be longer than the propagation delay of any internal combinatorial logic; if this logic is not given enough time to reach a stable state



(a) One Bit ALU Circuit

Op	Inv	Operation
00	0	$A \wedge B$
00	1	$A \wedge \bar{B}$
01	0	$A \vee B$
01	1	$A \vee \bar{B}$
10	0	$A + B$
10	1	$A - B$

(b) ALU Control Signals

Figure 3.3: One Bit ALU Schematic

then indeterminate values will be stored in flip-flops, invalidating future results. A processor's clock frequency is determined by the longest propagation delay found within its various subsystems.

Pipelining is a design technique which breaks down large combinatorial logic circuits into smaller sequential stages separated by flip-flops. This technique is used to spread complex operations like floating point arithmetic over multiple clock cycles. While pipelined circuits take a similar amount of time to produce their final results, each individual stage has a smaller propagation delay and can therefore operate at higher clock frequencies. Pipelining can also increase throughput if several stages of the pipeline can be kept active simultaneously.

It is worth noting that although logic pipelining is related to instruction pipelining, they are distinct concepts. The latter includes speculative execution, instruction re-ordering and other optimisations on top of basic pipelining.

3.2.1 Arithmetic Logic Unit Design

Arithmetic Logic Units (ALUs) are a type of circuit which can be implemented using either sequential or combinatorial logic. The process of designing ALUs is used as a motivating example in the next chapter. This subsection provides all the necessary background information about their construction.

ALUs perform arithmetic and logical operations on binary data. Figure 3.3 shows the schematic for a single bit ALU which contains three logic gates; *AND*,

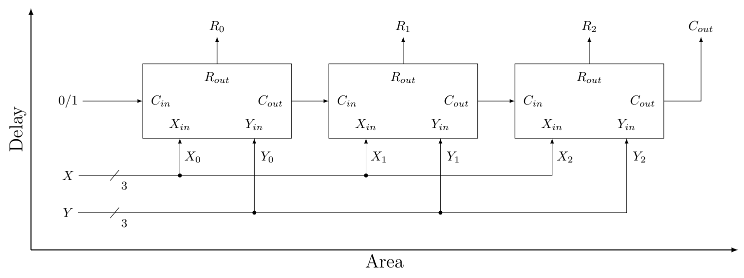


Figure 3.4: Combinatorial ALU

OR and *NOT*; and a one bit full adder. This circuit takes in a pair of input bits, X_{in} and Y_{in}, passes them to each of its logic elements, then returns one of their results as output. Control signals Op and Inv are sent to multiplexers to select which element is connected to the R_{out} output and whether Y_{in} should be negated, respectively. Output C_{out} indicates whether a carry overflow occurred during binary arithmetic.

Performing operations on individual bits is of limited usefulness. Practical ALUs handle multiple bits, with the exact number determined by processor word length. These multi-bit circuits can be constructed by combining single bit ALUs in several ways, each with its own set of design compromises.

Figure 3.4 shows a combinatorial design for a three-bit ALU in which three single bit units are chained together in parallel. Each unit processes one pair of input bits to produce one output bit, with carry bits rippling down the chain for arithmetic operations. Control signal inputs are omitted from this diagram; these are connected together and receive the same inputs.

Figure 3.5 shows an alternative, sequential design which reuses the same ALU three times, with carry outputs persisted in a flip-flop between each stage. This design exhibits longer delays and slower performance because it produces output one bit at a time. These disadvantages are offset by comparable reductions in power consumption and chip area requirements.

Although practical ALUs are more complex than those described above, the same design trade-offs still apply. For instance, most ALUs use carry lookahead

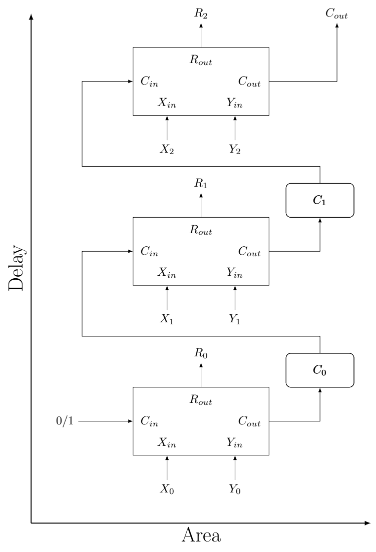


Figure 3.5: Sequential ALU

rather than ripple carry schemes, which is a further example of time/space trade-off. It is also possible to combine sequential and combinatorial logic in different ratios; a sixteen bit ALU may process all bits in parallel, or eight bits at a time in two stages, or four bits at a time in four stages, and so on.

3.3 CMOS Power Draw

Equation 3.1 describes the sources of power draw in CMOS chips, of which dynamic and leakage power are the most significant:

$$P_{tot} = P_{dyn} + P_{leak} + P_{other} \tag{3.1}$$

Dynamic power refers to the power consumed when transistors change state as a processor performs work, while *Leakage power* is consumed even when gates remain inactive. Other forms of power dissipation do exist, however their effects are comparatively minor [73].

$$P_{dyn} \propto CV^2Af \tag{3.2}$$

Equation 3.2 is a common approximation of dynamic power in which C denotes load capacitance, V the supply voltage, A the activity factor and f the clock frequency. Each of these factors is covered in more detail below.

Load capacitance (C) is a property of chip architectures which depends on the length of wires between on-chip structures and the degree of connectivity between their logic gates. Hardware engineers can minimise this property by optimising chip layout and favouring small, simple designs which can be packed more tightly onto chip real-estate.

Supply Voltage (V) is the voltage at which a processor operates. Supply voltage remains proportional to feature size under Dennard scaling, meaning this factor was ultimately responsible for the benefits seen during the "Free Lunch" period [52].

Activity Factor (A) is a scalar value between zero and one which represents the fraction of logic elements that change state each clock cycle. The exact value of A changes depending on a processor's workload.

Clock Frequency (f) is the number of clock cycles which occur in one second. Clock frequency and supply voltage vary in tandem, taking values from a set of *(frequency, voltage)* pairs known in the literature as Power States or *P-states.*

Leakage power exists because the insulating properties of silicon break down at very small scales. Quantum tunnelling and other effects allow some current to flow (or leak) even when gates remain inactive. Leakage accounts for an increasing proportion of power consumption as transistors continue to shrink.

$$P_{leak} = V \times I_{leak} \tag{3.3}$$

Equation 3.3 is a simplified expression for leakage power which exploits the fact that leakage current (I_{leak}) is not related to workload [76]. Leakage current depends on a number of factors which are outside the control of performance engineers. These include ambient temperature, the dielectric constant of CMOS transistors and the threshold voltage separating zero and one states.

CMOS power draw is ultimately limited by how quickly power can be delivered to a chip and how quickly the resulting heat can be removed from it. Temperature differentials cause material properties to change and lead to physical stresses which reduce component lifespan.

Thermal Design Power (TDP) is the maximum power draw that a processor can sustain for thermally significant periods while running software [101]. This figure is determined by the maximum rate at which heat can be generated and dissipated without causing damage to sensitive electronics. Real-world power consumption often approaches TDP limits because hardware designers have a strong incentive to maximise the performance of their processors.

3.4 Architectural Energy Efficiency Features

Moore's law led to rapid increases in the transistor budgets available to hardware designers. During the period of Dennard scaling, these extra transistors were used to add features to increasingly complex single core processors which were optimised for high clock speeds and single threaded performance. This is exemplified by Intel's Netburst architecture, which was in use between 2000 and 2006 and featured in their Pentium 4 processors [59].

A large proportion of Netburst's transistor budget went towards creating highly superscalar processors [78]. These processors featured extremely deep instruction pipelines up to 36 stages in length, along with extensive hardware to support the dynamic scheduling, speculative execution and instruction reordering required to keep these pipelines full. This approach became unsustainable after Dennard scaling ended in 2006, when thermal problems caused by high power consumption led to the abandonment of the Netburst architecture and the end of the Pentium product line [98].

Hardware designers have since developed a number architectural features to improve the energy efficiency of their processors in the post-Dennard era. Each of the features listed below seeks to minimise some subset of the parameters in the CMOS power equations listed above.

3.4.1 Multi-core Processors

The most notable change to processor design in recent years has been the introduction of Multi-core Central Processing Units (CPUs). Multi-core architectures replace the monolithic processors of the past with a collection of smaller, simpler interconnected cores. These cores operate independently while sharing access to common hardware like last level caches and main memory.

Smaller cores have fewer components and shorter wire lengths, both of which lead to reduced load capacitance and leakage current. Multiple cores also amplify the effects of other energy efficiency features listed below.

Multi-core processors prioritise throughput over single threaded performance. Performance engineers have to deal with the overhead of parallelising their codes in order to see the benefit of this architectural approach.

3.4.2 Clock Gating

Of all the subsystems in modern processors, the *clock tree* has the potential to be the most power hungry. Clock trees distribute the signal from a central clock across all areas of a processor, which inevitably means they have long wire lengths and high load capacitance. Furthermore, their activity factor is maximal by definition; circuits carrying the clock signal will change state with every clock cycle.

Clock gating reduces power consumption by disconnecting or *gating* those parts of the clock tree which are connected to idle logic. The activity factor of gated subtrees drops to zero, meaning they only incur leakage power costs.

Performance engineers can maximise the benefit of clock gating by batching similar operations together. In the case of multi-core processors, similar logic can also be pinned to particular cores. Both of these approaches result in longer idle periods for the effected subsystems, increasing the likelihood of clock gating.

3.4.3 Dynamic Voltage and Frequency Scaling

Equation 3.2 shows that dynamic power consumption grows quadratically with supply voltage. Small reductions in supply voltage therefore have the potential to deliver significant reductions in power consumption. Unfortunately, the switching speed of transistors also decreases when they operate at lower voltages. This increases the propagation delay of CMOS logic, which can lead to timing errors if this delay exceeds the clock period.

Dynamic Voltage and Frequency Scaling (DVFS) gets around this issue by scaling supply voltage and clock frequency in tandem. Lower supply voltages are paired with slower clock speeds in order to give CMOS logic enough time to finish operating. These matched supply voltage and clock frequency pairs are called *P-States*. DVFS allows processors to choose from a set of predefined P-States based on their current workload.

DVFS has a cubic relationship with power consumption because clock frequency is also a parameter in Equation 3.2. Its relationship with energy is less obvious because reduced power consumption can be offset by longer runtimes. DVFS is most effective when performance does not depend on clock speed; a processor may enter lower power states while it waits for data, for example.

3.4.4 Heterogeneous Computing

Heterogeneous computing takes two main forms. The first, most common form of heterogeneity is the inclusion of accelerators or other special purpose hardware within compute nodes. These accelerators augment the capabilities of general purpose CPUs by allowing them to offload specific tasks.

A second form of heterogeneity involves building special-purpose compute cores directly into processors. ARM's "big.LITTLE" concept is one example of this kind, in which smaller, more energy efficient cores are twinned with larger, more performant ones [97]. Work migrates between these cores as required to meet performance and energy efficiency targets. Advanced energy-aware scheduling techniques are necessary to take advantage of this type of heterogeneous architecture [123].

The Green500 list of energy efficient supercomputers is dominated by heterogeneous architectures [114]. All but one of the top thirty machines in the November 2016 list make extensive use of accelerators, while the remaining machine is based on a custom heterogeneous processor design [2].

3.5 Energy Efficiency Trends

This section charts key trends in processor energy efficiency and design. Many vendors target the HPC market, and each of them offers a range of products to target different market segments, operating points and use cases. This makes it difficult to perform fair comparisons between different hardware generations. It is more instructive to base such comparisons on specific HPC systems.

Pleiades, a machine operated by the United States National Aeronautics and Space Administration (NASA), is a particularly good candidate for such comparisons for a number of reasons. First, it has seen extensive use in many different scientific investigations. This shows that it is representative of production systems and not simply a "stunt machine" designed to score highly in benchmark tests at the expense of real workloads.

Secondly, a program of incremental upgrades has maintained Pleiades' position as one of the world's fastest supercomputers over nearly a decade of operation. Pleiades first appeared in third place in the November 2008 Top500 list with a Linpack performance of 4.87×10^{14} Floating Point Operations per Second (FLOPS) and a power consumption of 2090 kW. As of November 2016 it occupies thirteenth place, with performance figures of 5.95×10^{15} FLOPS at 4407 kW [3]. This corresponds to more than twelve times the original floating point performance and approximately twice the power draw.

Thirdly, Pleiades' long service life and incremental evolution provides an unbroken record of CPU advances over many years. Until 2016, Intel CPUs

Year	Codename	Model	Cores	Frequency (GHz)	TDP (W)	Lithography (nm)
2008	Harpertown	E5472	4	3.00	80	45
2009	Nehalem	X5570	4	2.93	95	45
2010	Westmere	X5670	6	2.93	95	32
2012	Sandy Bridge	E5-2670	8	2.60	115	32
2014	Ivy Bridge	E5-2680 v2	10	2.80	115	22
2015	Haswell	E5-2680 v3	12	2.50	120	22
2016	Broadwell	E5-2680 v4	14	2.40	120	14

Table 3.1: Pleiades CPU Upgrades

followed a "Tick-Tock" release model, with roughly 18 months between each phase. Under this model, each "Tick" represents a shrinking of CMOS feature sizes and each "Tock" represents the introduction of a new microarchitecture. Due to its annual update schedule, Pleiades has incorporated new hardware at every stage of this cycle since 2008.

Finally, Pleiades' initial design goals prioritised energy efficiency, and this focus has persisted throughout its many years of service [15]. Coupled with the annual updates, this has led to Pleiades becoming something of a showcase for energy efficient processor technologies.

Table 3.1 lists the year in which different processor models were first incorporated into Pleiades. Several broad trends can be identified in this table; core counts and TDP have been rising while feature size and clock frequency have decreased. Also, although per-processor power consumption has increased, per-core power consumption decreased over the same period.

In its initial configuration, Pleiades consisted of 100 racks of 64 dual-socket Harpertown nodes, totalling 12800 processors and 51200 cores. As of 2016, the machine consists of 161 racks containing a mixture of Broadwell, Haswell, Ivy Bridge and Sandy Bridge nodes, totalling 22944 processors and 246048 cores. A further three 32-node Sandy Bridge racks are equipped with accelerators; two racks with NVIDIA K40 GPUs and one with Intel Xeon Phi 5110P coprocessors.

Pleiades mirrors the trends seen across the wider Top500 list. Rising per-processor power consumption is being compounded by increasing numbers of processors. Even more striking is the exponential growth in core count fuelled by ever more cores in ever more processors.

3.6 Summary

Energy-aware performance engineering starts by understanding the factors which contribute to power consumption. Performance engineers must tune their codes to take advantage of these factors in order to improve energy efficiency. Current trends in computer hardware point towards a future of diverse hardware platforms and heterogeneous architectures. New tools and techniques are required to support energy-aware code optimisation on these new platforms.

The next chapter considers performance metrics which can be used to guide energy-aware software optimisation. Current metrics were developed by the hardware community to guide the process of designing energy efficient hardware. These metrics are shown to be inappropriate for software optimisation and suitable alternatives are proposed.

CHAPTER 4
Metrics for Energy-Aware Software Optimisation

Hardware engineers use delay product metrics as guides when designing novel processor architectures. In particular, the Energy Delay Product (EDP) family of metrics was created to promote the development of energy efficient processors. Some members of the performance engineering community have since co-opted these hardware metrics to guide energy-aware software optimisation.

This chapter begins by examining the rationale behind delay product metrics in order to explain why they make sense from a hardware design perspective. It then goes on to show that certain assumptions which underpin these metrics do not hold for software optimisation. A list of necessary criteria for software optimisation metrics is proposed which demonstrates the shortcomings of delay product formulations. These criteria are then used to create new metrics which are more suitable for energy-aware software optimisation. This chapter concludes with a demonstration of these metrics, studying codes taken from the Mantevo application suite.

4.1 Delay Product Metrics

Processor designs are subject to a number of physical and technical constraints. Propagation delay is a technical constraint which determines processor clock speed and therefore has a strong impact on performance. Physical constraints include limits to chip area, power draw or energy consumption. Hardware design involves striking a balance between potentially conflicting design constraints.

Processors consist of many interconnected subsystems operating in tandem. Ensuring that all of these subsystems can work together without violating any constraints is a major design challenge. This is made more difficult by the

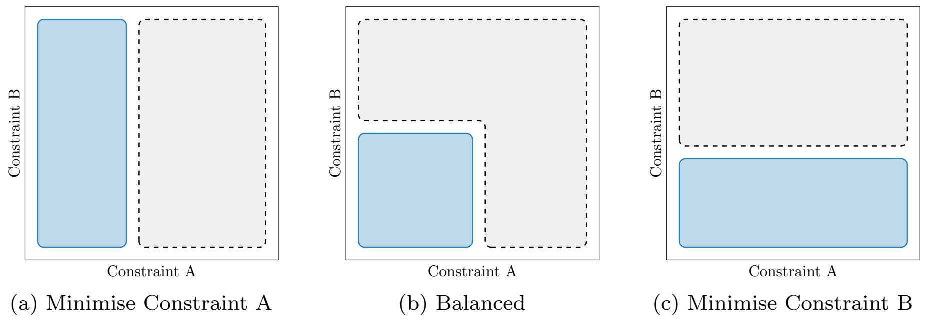


Figure 4.1: Design Trade-Off Constraint Diagram

fact that individual subsystems are usually designed in isolation first and then integrated later on. Hardware engineers must adhere to global constraints when working on individual subsystems, despite not knowing the overall processor design. In the absence of global knowledge, the best approach is to choose whichever design places fewest restrictions on the rest of the processor.

The multi-bit ALUs introduced in Section 3.2.1 are a good example of how design constraints can often be traded off against each other. Recall that the combinatorial circuit called for three single bit ALUs operating in parallel, while the sequential circuit reused the same single bit unit three times. The former design uses three times more circuit area and power, whereas the propagation delay of the latter is three times longer.

In cases where direct trade-offs are possible, the least restrictive design is the one which uses fewest resources overall. The dark blue areas in Figure 4.1 represent three alternative designs which make different trade-offs between two constraints. Integrating subsystems to create a finished design which satisfies global constraints can be thought of as trying to "pack" subsystems into one of these constraint diagrams. The least restrictive design is the one which leaves the most free space, shown in grey, available for other components.

Delay product metrics embody the possible trade-offs between propagation delay and other design constraints. Multiplying constraints together is equivalent to calculating how much space a subsystem occupies in a constraint diagram. The key benefit of delay product metrics is that they promote flexibility by maximising the scope for trade-offs later on in the design process.

Power Delay Product (PDP) and Area Delay Product (ADP) are commonly used metrics in processor design. Power constraints, in the form of Thermal Design Power (TDP) limits, are determined by the maximum rate at which heat can be removed from the processor. Area constraints are largely economic in origin; smaller designs have higher yields and can be more reliable.

PDP and ADP exhibit two properties which are hallmarks of a meaningful delay product formulation. First, both area and power can be exchanged for reduced delay in a relatively linear manner. Secondly, power and area are in some sense "orthogonal" to delay; different components can be active at the same time, while the same component can be active at different times. When one circuit stops drawing power then this power becomes available to other parts of the processor. Likewise, the same circuit area can be reused at different times, as seen in the sequential Arithmetic Logic Unit (ALU) design above.

The proliferation of battery powered computing devices led to energy use becoming a prominent constraint in processor design. Horowitz et al. proposed EDP, which uses a delay product formulation to measure energy efficiency [63]. They argued that if two circuits produce identical results at different speeds while consuming the same amount of energy, then the faster circuit has used its energy more efficiently. PDP does not conform to this definition because it does not penalise slower designs if energy consumption remains constant.

EDP departs from some of the intuitions behind other delay product metrics. One example is that Horowitz et al. reinterpret delay, treating it as synonymous with runtime. Also, while it may be possible to exchange increased energy consumption for higher performance, this relationship is not straightforward or linear. Most critically, energy and delay are not orthogonal for two reasons. First, energy is defined as the product of power and time, and time and delay are related. Secondly, energy is a consumable resource which can be used only once, by a single component. Energy cannot be reused, so the concept of "packing" components to fit within constraints does not apply.

4.2 Software Optimisation Metrics

This section provides formal definitions which underpin later discussions before outlining the properties a software optimisation metric should exhibit. It begins by formalising the notion of a code as a repeatable sequence of instructions which, when executed by a processor, incurs energy and runtime costs.

Definition 1. *All processors consume non-zero amounts of time and energy to run programs. The cost of a code θ is the pair $(E_\theta, t_\theta) \in \mathbb{R}_+ \times \mathbb{R}_+$ corresponding to the energy and runtime costs incurred by running it on a given platform.*

Definition 2. *Codes can be composed by concatenating their instruction sequences. The composition of codes θ and λ yields the following cost:*

$$\theta \circ \lambda = (E_\theta + E_\lambda, t_\theta + t_\lambda)$$

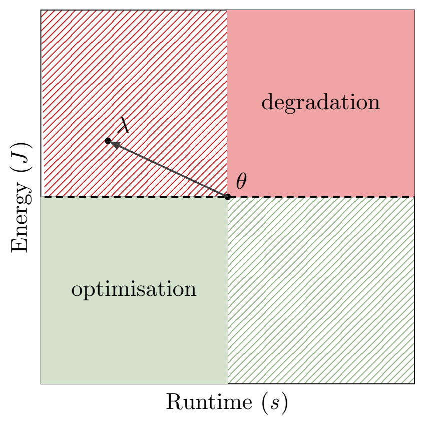
The goal of energy-aware software optimisation is to minimise the runtime and energy costs of a given application. Energy-aware optimisation metrics are functions of energy and time which capture the utility of a code.

Definition 3. *An energy-aware optimisation metric is an element-wise monotonic function M which combines energy and runtime costs into a scalar Figure of Merit (FoM):*

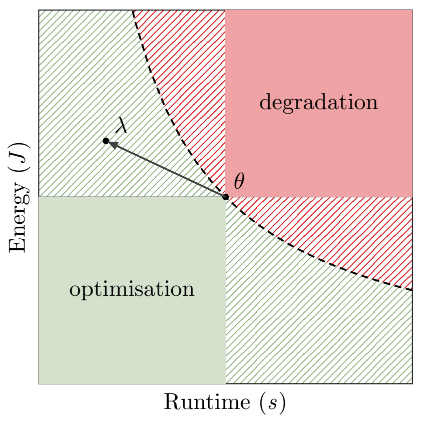
$$M\colon (E, t) \in \mathbb{R}_+ \times \mathbb{R}_+ \to \mathbb{R}_+$$

Element-wise monotonicity means that for all fixed $E_0, t_0 \in \mathbb{R}_+$, the functions $M(E_0, t)$ and $M(E, t_0)$ are monotonic. In other words, increasing one cost without a corresponding reduction in the other leads to a worse FoM.

Software optimisation can be modelled as a hill-climbing problem [54]. Starting from an initial code θ, performance engineers make incremental changes and measure their impact using a FoM metric. Changes which improve performance



(a) Single Objective Metric (Energy)



(b) Multi-Objective Metric (Et^1)

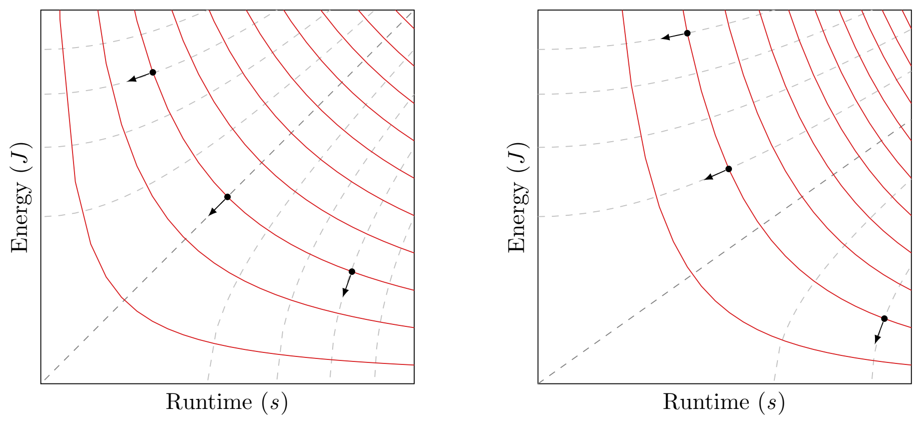
Figure 4.2: Metric Optimisation Regions

against this metric are kept while those which reduce it are discarded. Whether a given code change represents an optimisation depends on the metric chosen.

Definition 4. *For logically equivalent codes θ and λ, the transformation $\theta \to \lambda$ is an optimisation with respect to metric M iff $M(\lambda)$ strictly dominates $M(\theta)$.*

By Definition 3, all valid metrics identify code changes that reduce both energy and time costs as optimisations. Similarly, all code changes leading to strictly worse performance in both regards will be treated as performance degradations. The classification given by energy-aware optimisation metrics only differ in cases where some degree of energy-time trade-off is possible.

Figure 4.2 shows how valid metrics can disagree on whether the same code change $\theta \to \lambda$ is an optimisation. In these diagrams, the lighter green areas correspond to performance optimisations and darker red areas to performance degradations. They are separated by a dashed *Isometric line* connecting all points with FoM values equal to $M(\theta)$. Both metrics agree on code changes in the solid shaded regions where costs change in tandem. Energy-time trade-offs are represented by cross-hatched quadrants. The Multi-Objective Optimisation (MOO) metric in Figure 4.2b identifies $\theta \to \lambda$ as a valid energy-time trade-off, whereas Figure 4.2a shows it is not an energy optimisation.



(a) Et^1 (b) Et^2

Figure 4.3: Et^n Metric Fitness Landscapes

Energy-aware optimisation metrics assign a FoM to all (E, t) cost pairs. Returning to the hill-climbing analogy, optimisation metrics define a fitness landscape over the energy/time plane. Figure 4.3 shows how plots similar to Figure 4.2 can be used to visualise the fitness landscapes of different metrics.

The isometric lines coloured red in Figure 4.3 connect all points where the FoM is some multiple of a fixed value. Mathematically these lines represent level sets of the metric function; intuitively they are contours in the corresponding fitness landscape. The closeness of these lines corresponds to the gradient of the fitness landscape.

Isotopic lines run perpendicular to isometric lines, and correspond to the path of fastest decent (steepest gradient) within the fitness landscape. Mathematically, these lines are orthogonal trajectories of a metric function M. Conceptually, they show the direction in which a metric drives optimisation.

Having formally defined what an energy-aware optimisation metric is and how it can be visualised, attention now turns to how it should behave. The goal of an optimisation metric is to condense the utility of an application into a single, meaningful FoM. The following list enumerates the properties which an idealised optimisation metric should possess:

1. **Bounded:** A metric should bound regions of the optimisation space;
2. **Directed:** drive optimisation efforts in a sensible direction;
3. **Additive:** remain additive (linear) under code composition;
4. **Stable:** give a stable definition of optimisation under code composition;
5. **Tunable:** be tunable to different application domains; and
6. **Intuitive:** correspond to a tangible and intuitive property of the system.

These properties are explored further in the next section.

4.3 Et^n Evaluation

The previous section listed several desirable criteria for energy-aware optimisation metrics. In this section these criteria are used to evaluate the suitability of Et^n metrics for guiding software optimisation.

Bounded

The first criteria states that energy-aware optimisation metrics should bound regions of the optimisation space. In other words, a metric should place upper limits on how much energy or runtime can be consumed under a given FoM. This requirement is met if the isometric lines described by a metric intercept both the energy and runtime axes.

Figure 4.3 shows that Et^n isometric lines do not intercept either axis. In theory, codes can be modified to consume an arbitrarily large amount of either time or energy while still improving their overall performance. Bounded metrics do not consider such pathological cases to be valid optimisations. Another benefit of bounded metrics is that they limit the space in which to search for optimisations; something which Et^n cannot do.

Directed

The second criteria requires metrics to guide optimisation in sensible directions. Intuitively, performance engineers wish to speed up slow codes and reduce the power consumption of energy intensive ones. On the contrary, Et^n disproportionately rewards speeding up fast codes and saving energy in frugal ones. As energy consumption increases, Et^n gives higher priority to runtime optimisation and vice versa. This fault was encountered by Hsu et al. when they noted that Et^n metrics are unfairly biased towards massive parallelism in High Performance Computing (HPC) systems [65].

The first two criteria are linked. It is necessary (but not sufficient) for a metric to be bounded in order for it to guide optimisation in a sensible direction. The isometric lines of an unbounded metric never touch either axis, meaning the corresponding isotopic lines must intersect the axes at right angles. As the energy or time cost of a code approaches zero, the path of fastest decent therefore tends exclusively towards further reductions in this already close-to-zero cost.

Additive

The third criteria states that FoM metrics should be additive under code composition. Performance engineers focus their attention on expensive procedures within a code. This involves profiling the code to identify areas causing poor performance, based on the assumption that the cost of a code is the sum of the costs of its constituent parts. While true for simple metrics like energy and time, this is not generally the case for compound metrics.

Definition 5. *A metric is additive iff for code segments θ and λ:*

$$M(\theta \circ \lambda) = M(\theta) + M(\lambda)$$

Metric functions must be linear in terms of both time and energy in order to fulfil this requirement. This is not the case for Et^n, for which the cost of a code tends to be much greater than the costs of its constituent parts. Profilers cannot be relied upon to identify targets for Et^n optimisation. Furthermore, this additional *non-local* cost depends on total application runtime and energy consumption. An Et^n FoM is therefore meaningless outside the context of a single fixed application.

Stable

The fourth criteria requires metrics to provide a stable definition for optimisation. If the same code change alters the cost of two applications by the same amount, and it is an optimisation with respect to metric M for one of the codes, then it should count as an optimisation for both of them.

Definition 6. *A metric is stable iff for equivalent code segments λ and λ':*

$$M(\lambda') < M(\lambda) \implies M(\theta \circ \lambda') < M(\theta \circ \lambda)$$

It is worth noting that linear metrics automatically fulfil this requirement. Linear metrics are inherently stable, however stable non-linear metrics also exist.

Et^n is an unstable metric as it does not provide a consistent definition of optimisation. Whether or not a code change counts as an optimisation under Et^n is *context sensitive.* Code changes can be counted as optimisations only when evaluated in the context of the full application. Targeted optimisation of particular subroutines is impossible, and all past optimisations must be re-evaluated every time a change is made to the application.

This problem with Et^n metrics is best illustrated with an example. Suppose an application contains a procedure which consumes 10 J over 10 s to produce some result. This corresponds to an Et^1 FoM of $10 \times 10 = 100$. A modification is made to the procedure, causing it to produce the same result in 11 J and 9 s. This is a valid optimisation because, although it increases energy consumption, it reduces Et^1 to $11 \times 9 = 99$.

After this procedure terminates, the application gives its user the option

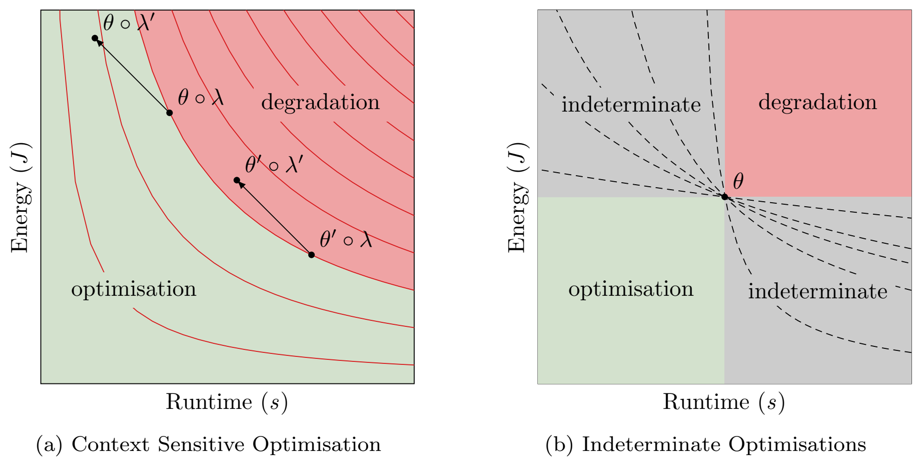


Figure 4.4: Et^n Optimisation Instability

to save the results at a cost of (5 J, 10 s). The un-optimised application could execute both tasks with total energy and runtime costs of $10 + 5 = 15$ J and $10 + 10 = 20$ s respectively, giving an overall EDP of $(10+5) \times (10+10) = 300$. The same sequence of actions in the 'optimised' application results in a higher (worse) EDP of $(11+5) \times (9+10) = 304$. Under Et^n metrics, saving the results of this procedure can somehow retroactively invalidate its optimisation.

Figure 4.4a shows how the same change applied to two codes with the same starting Et^n FoM may be considered either an optimisation or a performance degradation. Furthermore, Figure 4.4b shows how any energy-time trade-off can be made to appear as an optimisation or a performance degradation depending on the context. Different ratios of E_θ and t_θ can shift the optimisation/degradation boundary to any point within the indeterminate quadrants.

Mini-applications are powerful tools in scientific computing [56]. They package relevant features of large production applications into smaller, more manageable codes. Performance engineers use them as test beds to search for optimisations which can be ported back to the original application. Sometimes optimisations which work at small scale will fail to improve the production application, signalling a discrepancy between the mini and production applications. Using Et^n metrics, however, optimisations to the mini-application may not count as optimisations to the production code even when they yield identical cost changes in both cases. This is further evidence that Et^n metrics are incompatible with modern performance engineering techniques.

Tunable

The penultimate criteria is that it should be possible to tune a metric to reflect the energy and time constraints of different domains by means of an appropriate parameterization. The Et^n metric meets this criteria via its n parameter. This parameter sets the "exchange rate" at which small changes in runtime and energy can be traded against each other. This can be shown by equating the partial derivatives of Et^n as shown in Equation 4.1:

$$\begin{gathered}\frac{\partial}{\partial E}\left(Et^n\right) = t^n \qquad \text{and} \qquad \frac{\partial}{\partial t}\left(Et^n\right) = nEt^{n-1} \\ t^n \cdot \partial E = nEt^{n-1} \cdot \partial t \\ \frac{\partial E}{E} = n\frac{\partial t}{t} \end{gathered} \tag{4.1}$$

Intuitive

The final and most subjective criteria states that metrics should be intuitive. In practice, this means a metric should measure some tangible property of a system, ideally with values measured in meaningful units. Et^n does not meet this requirement.

The costs of an extra Joule or second are not fixed under Et^n; in fact, the cost of increasing each factor depends on the current magnitude of the other. This implies that a Joule consumed by a long running process somehow costs more than a Joule consumed by a short-lived one. Furthermore, real systems impose maximum and minimum rates of power consumption on a code, which are referred to in this work as P_{max} and P_{min}. Given that $P_{min} \cdot t < E < P_{max} \cdot t$, the growth rate of Et^n is $\Theta(t^{n+1})$. The FoM cost of an additional second or Joule grows polynomially, hindering comparison between different scales.

4.3.1 Justification of Et^n

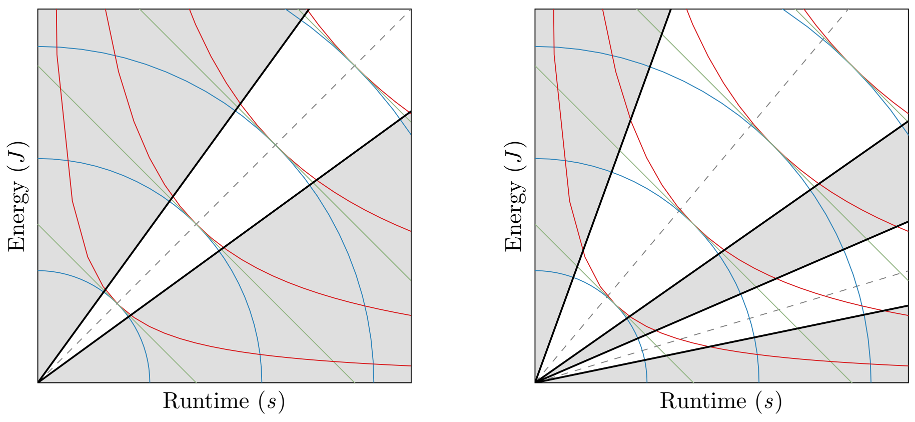
The continued use of Et^n metrics despite their flaws is a testament to the need for standardised energy-aware optimisation metrics. In the absence of better alternatives, software engineers rely on Et^n metrics because of their popularity and relative ease of use. Et^n metrics remain the de-facto standard technique for combining energy and runtime costs into a single FoM.

One factor which can mask the problems associated with Et^n metrics is the small range of power consumption figures exhibited when running HPC workloads at small scales. This range is limited by high base power consumption and marginal differences under load in modern hardware [49].

Figure 4.5a shows isometric lines for Et^1 as well as for both the novel metrics proposed below. It demonstrates how a small $[P_{min}, P_{max}]$ range (represented by the central unshaded area) limits the scope for divergence between different metrics. Although the numeric values of each metric may differ significantly, there is little scope for metrics to disagree as to which version of a code is optimal. This effect is more pronounced for smaller $[P_{min}, P_{max}]$ ranges. In the extreme case, when $P_{min} = P_{max}$, E_θ is a scalar multiple of t_θ and all energy-aware metrics become monotonically increasing functions of time.

The scarcity of power-instrumented hardware means that energy-aware optimisation is often attempted at the level of individual nodes. Although single nodes exhibit narrow $[P_{min}, P_{max}]$ ranges, multi-node and system-level power draw is much less constrained. As a consequence, different metrics may disagree on which is the most optimised version of a code at scale even when they all agree on a single node.

Figure 4.5b shows two performance envelopes with the larger having P_{min} and P_{max} values three times those of the smaller one. This models the effect of running the same code on a single node and over three nodes in parallel. Even at this small (3 node) scale the discrepancies between Et^n and other metrics become readily apparent. Similar discrepancies can also occur when running code across multiple architectures with different power characteristics, such as



(a) Narrow Range Equivalence (b) Discrepancies at Scale

Figure 4.5: Power-Limited Isometric Lines

GPUs and FPGAs.

4.4 Proposed Metrics

Two new FoM metrics for energy-aware software optimisation are proposed in this section. These metrics have slightly different properties and the choice of which to use is left to developers. That said, they both significantly outperform Et^n metrics according to the proposed assessment criteria.

The first new metric, Energy Delay Sum (EDS), is a weighted sum of energy and runtime costs. The second metric, Energy Delay Distance (EDD), measures the cost of an application in terms of Euclidean distance from an 'optimal' point at the energy/time origin where both costs are zero. Fitness landscapes for both metrics are shown in Figures 4.6a and 4.6b respectively.

4.4.1 Proposed Metric 1: Energy Delay Sum

Energy and compute time are limited resources which have costs associated with their consumption. The primary cost of energy consumption is the purchase price of electricity. Environmental impact and other concerns can also be factored in. Runtime also has a monetary cost – the purchase cost of a machine

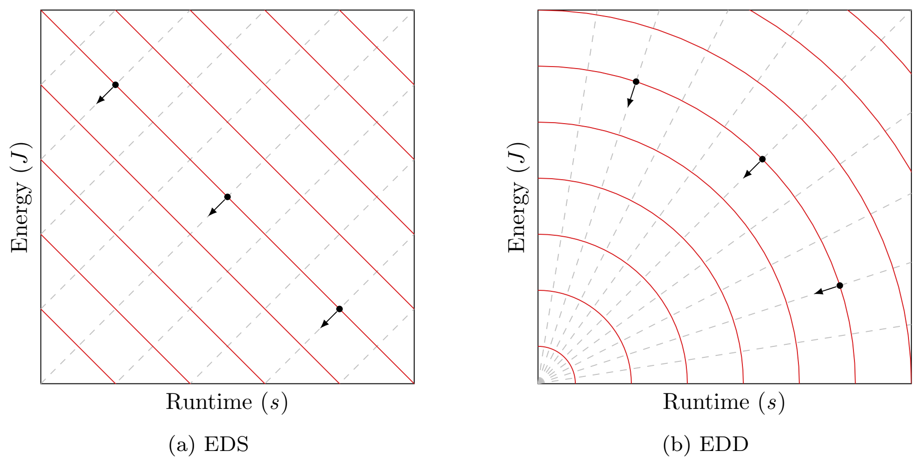


Figure 4.6: Proposed Metrics Fitness Landscapes

amortised over its limited lifespan. Energy and runtime costs are captured by the α and β parameters in Equation 4.2.

$$\begin{aligned} M(\theta) &= \alpha E_\theta + \beta t_\theta \\ &= (\alpha, \beta) \cdot (E_\theta, t_\theta) \end{aligned} \tag{4.2}$$

Bounded

This first criteria requires metrics to bound regions of the energy/time space. The isometric lines in Figure 4.6a intercept both axes, which is enough to satisfy this criteria. An EDS FoM therefore places upper limits on energy and runtime costs. The runtime contribution to a metric is maximised when energy is minimised and vice versa, allowing us to deduce cost limits under a given FoM:

$$\begin{aligned} M(\theta) &= \alpha \cdot E_{max} + \beta \cdot 0 \\ \therefore\ E_{max} &= \frac{M(\theta)}{\alpha} \\ M(\theta) &= \alpha \cdot 0 + \beta \cdot t_{max} \\ \therefore\ t_{max} &= \frac{M(\theta)}{\beta} \end{aligned}$$

Performance engineers need not evaluate code changes with energy costs greater than E_{max}, or runtime costs greater than t_{max}. This is in stark contrast to the Et^n case, where any given energy or runtime cost could be considered an optimisation under the right circumstances.

Directed

The second criteria requires metrics to guide optimisation in sensible directions. Fast, energy intensive codes are likely to require different optimisations to slow, energy efficient ones. As a linear function, EDS does not differentiate between these cases; the isotopic lines in Figure 4.6a all run in parallel. This metric still outperforms Et^n in this regard however as it does not introduce perverse optimisation incentives.

Additive and Stable

The third and fourth criteria require metrics to be linear functions of time and energy and to provide stable definitions of optimisation. The function $\alpha E + \beta t$ is linear in both parameters. Linear functions are automatically stable; meaning this metric fulfils both criteria, providing stable definitions for optimisation and allowing for meaningful code profiling.

Tunable

The penultimate criteria is that metrics should be tunable to different application domains. EDS allows energy and runtime costs to be specified via its α and β parameters. Unlike the exponential formulation of Et^n, it is immediately apparent how different values will alter the balance between energy and runtime.

A single scalar parameter would be enough to express any ratio of energy and time components. One property of this metric is that with appropriate tuning factors it can be used as a proxy for the monetary cost of running a code. This use-case is why two tuning parameters are used in this metric, so that it can provide notional value results.

Intuitive

The final criteria requires metrics to correspond to some meaningful property of the system. Given appropriate coefficients this metric can report results in terms of monetary cost. Monetary cost has meaningful units, allows for fair comparisons to be made between different platforms and architectures, and is useful during procurement.

Equation 4.2 provides a dot product formulation of the EDS metric which suggests a second geometric interpretation. Dot products correspond to the projection of one vector onto another – in this case of (E_λ, t_λ) onto (α, β).

4.4.2 Proposed Metric 2: Energy Delay Distance

EDS measures code performance in terms of separable energy and time costs. This fulfils all but one of the assessment criteria; as a linear function it was not able to direct the optimisation of codes according to their starting costs. The EDD metric remedies this by defining the cost of a code as its distance from the optimum point in the fitness landscape – the origin:

$$M(\theta) = \sqrt{{E_\theta}^2 + (\beta t_\theta)^2}$$

EDD can also be expressed as the magnitude of a weighted cost vector:

$$M(\theta) = \|(E_\theta,\, \beta \cdot t_\theta)\|$$

Bounded

The isometric lines shown in Figure 4.6b follow semi-circular trajectories which intercept the axes. This is sufficient to satisfy the first criteria, meaning that EDD limits E_{max} and t_{max} for a given FoM. These limits can be derived as follows:

$$\begin{aligned} M(\theta) &= \sqrt{{E_{max}}^2 + \beta \cdot 0} \\ \therefore\ E_{max} &= M(\theta) \\ M(\theta) &= \sqrt{0 + \beta \cdot {t_{max}}^2} \\ \therefore\ t_{max} &= \frac{M(\theta)}{\beta} \end{aligned}$$

Directed

The isometric lines for this metric form concentric ellipse segments centred about the origin. As a result, the corresponding isotopic lines converge on the origin. Figure 4.6b makes it clear that as a result this metric prioritises optimisations which minimise whichever cost is greater.

Additive

The formula for EDD is non-linear, meaning the overall FoM of a code is not equivalent to the sum of its parts. This is an unavoidable consequence of being a directed metric, and means that EDD is not ideal for accurate code profiling. Unlike Et^n, however, the discrepancy between the sum of component FoMs and the overall code FoM for EDD is bounded. Because EDD is defined in terms of vector magnitude it obeys the triangle inequality. As energy and time costs are always positive, this gives:

$$\sqrt{M(\theta)^2 + M(\lambda)^2} < M(\theta \circ \lambda) \leq M(\theta) + M(\lambda)$$

Stable

EDD does not meet the stability criteria. Figure 4.7 shows a case where $M(\lambda') < M(\lambda)$, yet $M(\theta \circ \lambda') > M(\theta \circ \lambda)$. The runtime axis is scaled so that isometric lines remain concentric for all values of β. That said, EDD instability is bounded

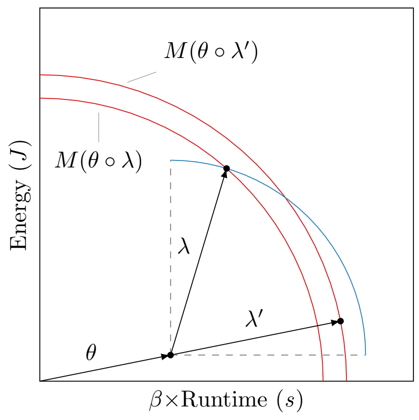

$M(\theta \circ \lambda')$
$M(\theta \circ \lambda)$
λ
λ'
θ
Energy (J)
$\beta\times$Runtime (s)

Figure 4.7: Energy Delay Distance Instability

by $M(\theta) + M(\lambda) - M(\theta \circ \lambda)$ as this metric obeys the following inequality:

$$M(\lambda') < M(\lambda) \implies M(\theta \circ \lambda') < M(\theta) + M(\lambda)$$

Tunable

This metric is tunable via the β parameter. A single parameter is sufficient to achieve any ratio of energy to runtime contribution.

Intuitive

This metric has a direct geometric interpretation as the Euclidean distance to the origin. It does not treat energy and runtime as separate and distinct costs; in reality they are inseparable. In general, reducing the runtime of a code will also reduce its energy consumption. EDD defines the cost of a code in terms of how far away it is from being optimal.

4.5 Case Study

This section investigates the energy-efficiency characteristics of codes in the Mantevo [56] mini-application benchmark suite. The results found show that the issues with Et^n become more evident at larger scales.

These experiments were carried out on the Taurus system at TU Dresden, which is equipped with High Density Energy Efficiency Monitoring (HDEEM) instrumentation [51]. Taurus is a heterogeneous cluster with several classes of node. This work was carried out on the largest of these classes, with each node featuring two 12-core Intel Xeon E5-2680 v3 CPUs and 64 GB of memory.

All codes were compiled using Intel C++ Compiler (ICC) version 15.0.3. Application parameters were based on default values, with problem sizes tuned where necessary to ensure reasonable run times on single nodes. Values for these parameters are given in Appendix C. Each application was run fifteen times on the same node to reduce the impact of random variations in runtime and energy.

Et^3 was used in these experiments because Laros et al. found that this strikes the right balance between runtime and energy for high performance computing [80]. This implies that a 1% reduction in runtime is approximately three times more valuable than the same reduction in energy consumption.

In order to facilitate fair comparison, EDS and EDD parameterisations are based on the same 3:1 ratio. Whereas the Et^n parameter operates in a relative fashion, however, EDS and EDD parameters are based on absolute costs of consumption. The power drawn by active Taurus nodes ranges between 207.68 W and 345.33 W [104], meaning the magnitude of energy costs will be around 300 times greater than that of runtime. runtime costs must be scaled by a factor of 300 before applying the same $3:1$ ratio in order to compensate for this effect.

The parameterisation used for EDS is obtained by multiplying the 300 scaling factor and the 3:1 ratio together, resulting in the parameters $\alpha = 1$ and $\beta = 3 \times 300 = 900$. The parameterisation of EDD is very similar, except that it uses a multiplier of $\sqrt{3}$ rather than 3 to account for the square root present in the definition of EDD. This results in a parameterisation of $\alpha = 1$ and $\beta = \sqrt{3} \times 300 \approx 519.615$.

In practice it would be better to adopt a more fine-grained parameterisation which reflects real-world costs incurred by HPC systems. That said, exact cost figures are seldom made available in the public domain.

Table 4.1: Single Node Code Costs

Code	Runtime (s)	Energy (J)	Et^3	EDS	EDD
TeaLeaf	323.8	99,810.3	3,388,487,549,302	391,230	195,629
PathFinder	337.1	71,943.9	2,755,943,021,015	375,334	189,361
CloverLeaf	214.3	57,861.2	569,447,839,399	250,731	125,489
CloverLeaf 3D	153.1	43,755.9	157,022,610,497	181,546	90,792
MiniMD	125.5	31,162.1	61,596,763,623	144,112	72,275
CoMD	105.6	24,837.8	29,248,586,337	119,878	60,231
MiniFE	36.7	8,465.6	418,461,914	41,496	20,864
HPCCG	36.5	8,059.5	391,910,314	40,910	20,607

Table 4.2: MiniMD Multi-Node Costs

Nodes	Runtime (s)	Energy (J)	Et^3	EDS	EDD
1	125.5	31,162.1	61,596,763,623	144,112	72,275
2	94.2	44,999.0	37,614,524,063	129,779	66,489
4	66.8	63,166.0	18,828,371,703	123,286	72,075
6	55.2	76,400.0	12,850,220,851	126,080	81,607
8	54.0	99,032.6	15,594,069,326	147,633	102,931
12	44.0	119,008.9	10,137,654,138	158,609	121,185
16	39.8	145,198.3	9,153,996,622	181,018	146,664
18	37.8	152,380.5	8,230,093,967	186,401	153,641
24	36.0	191,056.9	8,913,950,726	223,457	191,970
28	37.2	231,525.5	11,918,666,023	265,006	232,331
32	37.5	258,054.5	13,608,342,773	291,805	258,789
64	39.4	518,748.6	31,728,212,322	554,209	519,152
128	46.2	1,203,476.1	118,676,135,742	1,245,056	1,203,716

The first test carried out measured the runtime and energy consumption of various codes running on a single node. The results for this test are presented in Table 4.1.

The first thing to note is that Et^n results rapidly become unwieldy even for relatively short runtimes and low node counts. The runtime of HPCCG is around 11.4% that of TeaLeaf, and it also exhibits a slightly lower rate of power draw. This translates to a four orders of magnitude difference in their Et^n values. Adding a single second to the runtime of TeaLeaf would increase its Et^3 cost by over 80 times the total Et^3 value of HPCCG.

Another thing to note is that despite large variations in values, all metrics assign the same efficiency ordering to these codes. As previously mentioned, the limits of single-node power draw limit the scope for metrics to disagree.

The second test done was to measure the runtime and energy consumption of MiniMD running at scale. The results for this test are presented in Table 4.2.

These results show how biased Et^n metrics are in favour of massive parallelism. The efficiency of MiniMD according to Et^n improves as the node count increases to 18. It is only at the point when adding nodes delivers little or no reduction in runtime that this trend reverses.

EDS identifies 4 nodes as the optimal node count. This configuration delivers roughly twice the runtime performance of a single node at the cost of doubling the energy consumption. Adding nodes beyond this point results in energy costs increasing faster than runtime performance improves.

EDD identifies two nodes as the optimal node count. The fact this figure is lower corresponds to the intuition that parallelism introduces overhead. As the parallel overhead grows, so too does inefficiency as measured by this metric.

Et^3 gives the impression that below-linear speed-ups coupled with above-linear rises in energy consumption represent efficiency gains. Conversely, both EDS and EDD conform to a more conventional understanding of energy efficiency. They identify optimal configurations which can be justified intuitively.

4.6 Summary

This chapter argues that Et^n metrics are not appropriate for energy-aware software engineering. Alternative metrics are proposed which can be used to measure the cost of applications and guide their optimisation. Finally, the performance of these new metrics is compared against established techniques by studying codes taken from the Mantevo mini-application suite.

This chapter begins by explaining the rationale between delay product metrics. It then gives several reasons why Et^n metrics are unable to provide meaningful values for individual experiments, cannot be compared between experiments and do not support optimisation efforts. First, improving the Et^n FoM of a section of code can degrade overall performance. Secondly, Et^n metrics drive optimisation efforts in counterproductive directions, encouraging developers to speed up already fast code and seek energy efficiency gains in energy efficient codes. Finally, these metrics provide no meaningful definition of an optimisation. In total, Et^n was able to fulfil only one of the seven criteria for software optimisation metrics outlined in this chapter.

After identifying flaws in existing approaches, this chapter introduces EDS and EDD, two new metrics which outperform Et^n against the proposed assessment criteria. EDS is appropriate for measuring the cost of applications, while EDD is well suited to guiding application optimisation. Both new metrics fulfil the majority of the criteria for software optimisation metrics and EDS fulfils the maximum number possible.

This chapter finishes with a study into the energy-efficiency costs of several popular applications. This study shows how the flaws of Et^n metrics have managed to remain hidden in small-scale optimisation studies. It also demonstrates how these flaws will prevent Et^n metrics from being employed at scale. As a result, new metrics like EDS and EDD will be required to support performance engineers as interest in energy optimisation continues to grow.

CHAPTER 5
Power Optimised Software Envelope Model

This chapter introduces the Power Optimised Software Envelope (POSE) model. The energy efficiency of a code can be improved in one of two ways, either by shortening its runtime or by reducing its power consumption. POSE models quantify the potential benefits of each approach, allowing developers to focus their efforts on whichever offers the greatest rewards.

POSE models work by partitioning the energy/runtime plane into areas with different performance characteristics relative to an unoptimised code θ. Each of these areas corresponds to a specific optimisation outcome; either power optimisation, runtime optimisation or reduced code performance. The following insights are then derived from the relative size and positions of these areas:

1. The maximum possible energy savings from reduced power consumption;
2. The maximum possible improvement in a metric from power optimisation;
3. The minimum speed-up guaranteed to improve performance irrespective of power draw;
4. The maximum possible slow-down while still improving performance; and
5. The minimum speed-up guaranteed to outperform any power optimisation.

This chapter begins by explaining how POSE models are constructed for the Et^n family of metrics. The various insights provided by POSE are then described in detail. This process is then repeated for the Energy Delay Sum (EDS) and Energy Delay Distance (EDD) metrics from the previous chapter, showing that POSE is metric agnostic. Finally, POSE is demonstrated by performing an investigation into the energy-aware optimisation characteristics of codes taken from the Mantevo mini-application suite.

In contrast to the previous chapter, this work uses Et^2 rather than Et^3 to illustrate Et^n metrics. The reasons for this are twofold. First, Et^2 was used when the POSE model was first published [104] and this chapter adopts the same convention. Secondly, because Et^2 places less emphasis on runtime optimisation its POSE regions are larger, resulting in clearer diagrams. Despite this, Et^2 and Et^3 are both equally suitable for use in conjunction with POSE.

5.1 Model Construction

This section introduces the various bounds which make up a POSE model. It does so by presenting derivations of these bounds for the Et^n family of metrics, along with the coordinates at which they intersect. Et^n metrics are used to introduce POSE because, despite their flaws, they remain the de-facto standard metrics for energy-aware software optimisation.

Although this section refers to Et^n metrics, POSE is metric agnostic and can be used in conjunction with any optimisation metric which is a continuous function of runtime and energy costs. The only other prerequisite when using POSE is that runtime and energy consumption can be accurately measured or calculated for the target platform.

All of the definitions introduced by this section apply regardless of the metric chosen. They are used unmodified in a later section to produce equivalent POSE derivations for the EDS and EDD metrics. Appendix A summarises the POSE equation derivations for Et^n, EDS and EDD.

5.1.1 Feasible Performance Envelope

POSE models are built around the concept of a Feasible Performance Envelope (FPE). This envelope is the area between the P_{max} and P_{min} energy bounds shown in Figure 5.1. These are lines of gradient P_{max} and P_{min} respectively, values which correspond to the maximum and minimum rates of power draw possible during normal operation of the target platform. As such, the energy

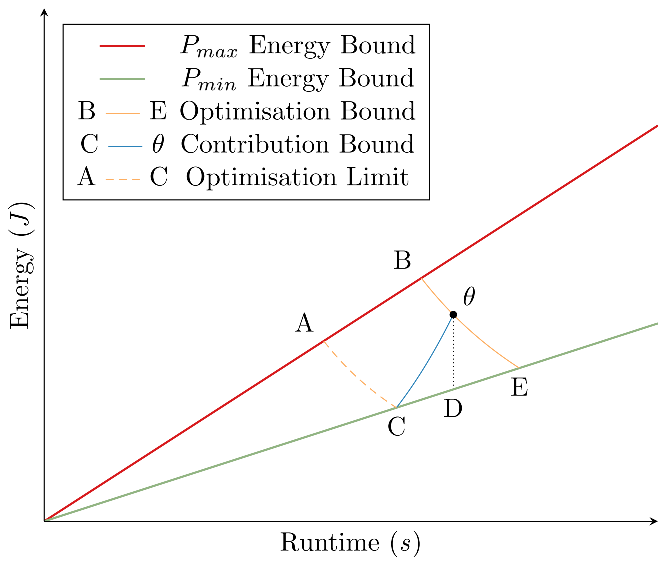


Figure 5.1: Et^2 Power Optimised Software Envelope

and runtime costs incurred by running any given code θ on this platform must be represented by a single point (E_θ, t_θ) somewhere inside this envelope.

The quantitative insights offered by POSE are calculated from the positions of the five vertices labelled A – E in Figure 5.1. Four of these vertices lie on an intersection between the FPE and one of the POSE bounds. The remaining vertex D lies directly below the initial code θ on the P_{min} energy bound at coordinates $(P_{min}\, t_\theta,\, t_\theta)$. This vertex corresponds to the largest possible pure power optimisation of θ, meaning an optimisation which reduces power consumption without any change to runtime.

5.1.2 Optimisation Bound

POSE considers the metric used to guide optimisation in order to constrain the search space for valid optimisations within the FPE.

Definition 7. *For logically equivalent codes θ and λ, the transformation $\theta \to \lambda$ is an optimisation with respect to a metric M iff $M(\lambda)$ dominates $M(\theta)$.*

The optimisation bound passes through θ, linking all points λ with the same metric value as the original code, such that $M(\lambda) = M(\theta)$. This bound is represented by the curve B — E in Figure 5.1.

Compared to θ, all points below the optimisation bound will have strictly better performance in terms of metric M, and all points above it will have strictly worse performance in terms of M. This follows from Definition 7, which is restated here from the previous chapter. In particular, any optimised versions of θ must appear below this bound in the direction of the origin.

The equation for the optimisation bound depends on the optimisation metric used. Deriving an equation for the optimisation bound involves finding an expression for the curve which links all points λ with the same metric value as θ. Figure 5.1 shows the optimisation bound for Et^2 while Equation 5.1 gives a general expression for the optimisation bound of any Et^n metric. The derivation of Equation 5.1 is as follows:

$$\begin{aligned} M(\lambda) &= M(\theta) \\ E_\lambda\, {t_\lambda}^n &= E_\theta\, {t_\theta}^n \\ E_\lambda &= E_\theta\, \frac{{t_\theta}^n}{{t_\lambda}^n} \\ E_\lambda &= E_\theta \left(\frac{t_\theta}{t_\lambda}\right)^n \end{aligned} \tag{5.1}$$

The intersections between the optimisation bound and the FPE determine the position of vertices B and E in Figure 5.1. Vertex B represents the fastest possible code within the FPE which shares the same metric value as θ. Any optimised version of θ with a runtime faster than B is guaranteed to outperform the original unoptimised code in terms of M. Similarly, vertex E represents the slowest possible code with the same metric value as θ. By definition, any optimised version of θ must run faster than E.

Vertex B lies on the intersection between the optimisation and P_{max} energy bounds. As such, its coordinates can be found by calculating the point at which Equation 5.1 for the optimisation bound intersects with the line $P_{max}\, t_\lambda$. This is done by equating the two expressions and re-arranging the result in terms of

t_λ to yield Equation 5.2 as follows:

$$\begin{aligned}
P_{max}\, t_\lambda &= E_\theta \left(\frac{t_\theta}{t_\lambda}\right)^n \\
P_{max}\, t_\lambda &= P_\theta\, t_\theta\, \frac{{t_\theta}^n}{{t_\lambda}^n} && (\text{As } E_\theta = P_\theta\, t_\theta) \\
P_{max}\, {t_\lambda}^{n+1} &= P_\theta\, {t_\theta}^{n+1} \\
{t_\lambda}^{n+1} &= {t_\theta}^{n+1}\, \frac{P_\theta}{P_{max}} \\
t_\lambda &= t_\theta \left(\frac{P_\theta}{P_{max}}\right)^{\frac{1}{n+1}} && (5.2)
\end{aligned}$$

The energy coordinate of vertex B is found by multiplying its runtime coordinate by P_{max}. Equation 5.3 lists the runtime and energy coordinates for vertex B:

$$\begin{aligned}
t_B &= t_\theta \left(\frac{P_\theta}{P_{max}}\right)^{\frac{1}{n+1}} \\
E_B &= P_{max} \cdot t_B
\end{aligned} \tag{5.3}$$

The derivation for the coordinates of vertex E is identical to Equation 5.2, except that P_{min} replaces P_{max} as E lies on the P_{min} energy bound. Equation 5.4 lists the runtime and energy coordinates for vertex E:

$$\begin{aligned}
t_E &= t_\theta \left(\frac{P_\theta}{P_{min}}\right)^{\frac{1}{n+1}} \\
E_E &= P_{min} \cdot t_E
\end{aligned} \tag{5.4}$$

5.1.3 Contribution Bound

All optimised versions of the initial, unoptimised code θ must appear inside the FPE in the region below the optimisation bound. The contribution bound further subdivides this region into runtime and power optimisations.

Performance engineers seek to use the most appropriate tools while searching for optimisations. Conventional time-based performance engineering techniques are more appropriate when searching for optimisations which result in large reductions in runtime, whereas energy-aware techniques are better suited to finding optimisations which primarily reduce power consumption. POSE uses the contribution bound to make this distinction.

Definition 8. *An optimisation $\theta \to \lambda$ with respect to metric M is considered to be a power optimisation iff the improvement in terms of M stems primarily from a reduction in power draw, such that $M(P_\lambda t_\theta, t_\theta)$ dominates $M(P_\theta t_\lambda, t_\lambda)$.*

Most optimisations will impact both runtime and power consumption to some degree. Definition 8 determines which of these impacts causes most improvement in terms of metric M. It does this by treating them as if they were two seperate optimisations; a pure power optimisation $(P_\theta t_\theta, t_\theta) \to (P_\lambda t_\theta, t_\theta)$, and a pure runtime optimisation $(P_\theta t_\theta, t_\theta) \to (P_\theta t_\lambda, t_\lambda)$, and then comparing them to see which is most beneficial. Power optimisations are those which derive most of their benefits from reduced power consumption rather than shorter runtimes, meaning that $M(P_\lambda t_\theta, t_\theta)$ dominates $M(P_\theta t_\lambda, t_\lambda)$.

Curve C — θ in Figure 5.1 links all points for which power and runtime factors contribute to M in the same ratio as the original code. By Definition 8, any power-optimised versions of θ must lie below this contribution bound.

The equation for the contribution bound also depends on the metric chosen. It obtained by letting $M(P_\lambda t_\theta, t_\theta) = M(P_\theta t_\lambda, t_\lambda)$, expanding the definition of M, re-arranging to make P_λ the subject, then finally multiplying by t_λ to provide a result in terms of energy. Figure 5.1 shows this bound for Et^2 while the general form for Et^n metrics is derived as follows:

$$\begin{aligned}
M(P_\lambda\, t_\theta, t_\theta) &= M(P_\theta\, t_\lambda, t_\lambda) \\
P_\lambda\, t_\theta \cdot {t_\theta}^n &= P_\theta\, t_\lambda \cdot {t_\lambda}^n \\
P_\lambda\, {t_\theta}^{n+1} &= P_\theta\, {t_\lambda}^{n+1} \\
P_\lambda &= P_\theta\, \frac{{t_\lambda}^{n+1}}{{t_\theta}^{n+1}} \\
P_\lambda &= P_\theta \left(\frac{t_\lambda}{t_\theta}\right)^{n+1} \\
E_\lambda &= P_\theta\, t_\lambda \left(\frac{t_\lambda}{t_\theta}\right)^{n+1} \qquad (5.5)
\end{aligned}$$

The intersection between the contribution and P_{min} energy bounds determines the position of vertex C in Figure 5.1. This vertex represents the fastest possible code which still meets the criteria to count as a power-optimised version of θ. Any optimisation which reduces runtime below that of C must have a larger impact on runtime than on power consumption, and as such would be considered a runtime optimisation.

Vertex C can also be interpreted as the best possible outcome for power optimisation. This is because, in addition to having the smallest runtime of any power optimisation, it also has the lowest possible power draw as it lies on the P_{min} energy bound. As such, it will have the best possible metric value of any point within the power optimised region.

The coordinates of vertex C can be found by calculating the point at which Equation 5.5 intersects with the line $P_{min}\, t_\lambda$. This is done by equating the two expressions, dividing throughout by common factors, then re-arranging the result in terms of t_λ to yield Equation 5.6 as follows:

$$\begin{aligned}
P_{min}\, t_\lambda &= P_\theta\, t_\lambda \left(\frac{t_\lambda}{t_\theta}\right)^{n+1} \\
P_{min} &= P_\theta \left(\frac{t_\lambda}{t_\theta}\right)^{n+1} \\
\frac{P_{min}}{P_\theta} &= \frac{{t_\lambda}^{n+1}}{{t_\theta}^{n+1}} \\
{t_\lambda}^{n+1} &= {t_\theta}^{n+1}\, \frac{P_{min}}{P_\theta} \\
t_\lambda &= t_\theta \left(\frac{P_{min}}{P_\theta}\right)^{\frac{1}{n+1}} \qquad (5.6)
\end{aligned}$$

The energy coordinate of vertex C is found by multiplying its runtime coordinate by P_{min}. Equation 5.7 lists the runtime and energy coordinates for vertex C:

$$\begin{aligned}
t_C &= t_\theta \left(\frac{P_{min}}{P_\theta}\right)^{\frac{1}{n+1}} \\
E_C &= P_{min} \cdot t_C
\end{aligned} \qquad (5.7)$$

5.1.4 Optimisation Limit

The bounds described so far delineate those regions of the energy/runtime plane in which runtime and power optimised versions of a given code can be found. The optimisation limit further partitions runtime optimisations into those which could potentially be outperformed by some hypothetical power optimisation and those which strictly dominate all possible power optimisations.

As its name suggests, the optimisation limit is closely related to the optimisation bound. They both link all points with the same metric value as a reference code, and as such are both defined by Equation 5.1. The only difference between them is that the optimisation limit connects all points with the same metric value as vertex C rather than the original code θ.

Given that vertex C represents the best possible outcome from power optimisation, all optimisations which lie below the optimisation limit must strictly dominate any possible power optimisation.

Vertex A lies on the intersection between the optimisation limit and the P_{max} energy bound in Figure 5.1. This vertex represents the fastest possible code with the same metric value as C, which in turn corresponds to the best possible outcome from power optimisation. As such, any optimisation which results in a faster code than A will outperform all possible power optimisations.

Because the optimisation bound and the optimisation limit are both based on Equation 5.1, the expressions for their coordinates are also similar. In particular, the coordinates of vertex A can be obtained using the same expressions derived for vertex B in Equation 5.3. The only difference is that C replaces θ as the reference point used, yielding Equation 5.8. The expression for t_C in Equation 5.7 is then substituted in to give Equation 5.9, an expression for the coordinate t_A in terms of θ, as follows:

$$t_A = t_C \left(\frac{P_C}{P_{max}}\right)^{\frac{1}{n+1}} \tag{5.8}$$

$$t_A = t_C \left(\frac{P_{min}}{P_{max}}\right)^{\frac{1}{n+1}} \qquad \text{(As } P_C = P_{min}\text{)}$$

$$t_A = t_\theta \left(\frac{P_{min}}{P_\theta}\right)^{\frac{1}{n+1}} \left(\frac{P_{min}}{P_{max}}\right)^{\frac{1}{n+1}} \qquad \text{(By Equation 5.7)}$$

$$t_A = t_\theta \left(\frac{{P_{min}}^2}{P_\theta\, P_{max}}\right)^{\frac{1}{n+1}} \tag{5.9}$$

The energy coordinate of vertex A is found by multiplying its runtime coordinate by P_{max}. Equation 5.10 lists the runtime and energy coordinates for vertex A:

$$\begin{aligned} t_A &= t_\theta \left(\frac{{P_{min}}^2}{P_\theta\, P_{max}}\right)^{\frac{1}{n+1}} \\ E_A &= P_{max} \cdot t_A \end{aligned} \tag{5.10}$$

5.2 POSE Insights

Figure 5.2 shows how POSE partitions the feasible performance envelope into four distinct regions, each with different performance characteristics.

Region 1 contains runtime optimisations which dominate the best case power

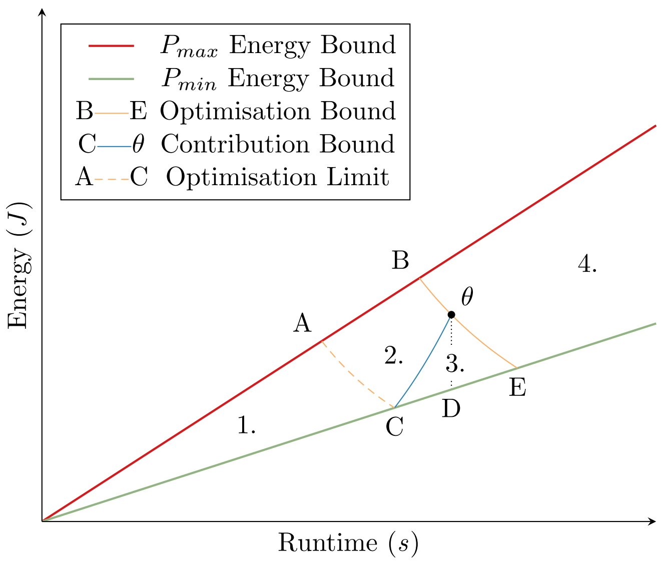


Figure 5.2: Et^2 Power Optimised Software Envelope Regions

optimisation in terms of a given metric M (*Strong Runtime Optimisation*). Region 2 contains runtime optimisations which dominate θ in terms of M, yet may be outperformed by some power optimised version of θ (*Weak Runtime Optimisation*). Region 3 contains optimisations for which improvements to M are primarily due to reduced power consumption (*Power Optimisation*). Finally, Region 4 corresponds to codes with performance strictly worse than that of θ (*Performance Degradation*).

The five vertices labelled A to E correspond to extreme outcomes of energy-aware optimisation. Comparing these outcomes to the initial performance of θ provides quantitative insights about the optimisation potential for this code. These insights fall into two broad categories which together help performance engineers decide if power optimisation is likely to prove worthwhile.

The first category relates to the potential benefits from power optimisation. The difference in energy between points θ and D places an upper bound on the amount of energy which can be saved by reducing power consumption. Similarly, the difference in value between $M(\theta)$ and $M(C)$ gives an upper bound for the improvement in a metric which can be delivered by power optimisation.

The second category relates to the scope a code has for power optimisation. The ratio t_θ/t_B represents the smallest speed-up which guarantees a code that outperforms θ with respect to M. The difference in runtime between points E and θ represents the maximum increase in runtime which could be traded off to achieve a slower yet more energy efficient code. Finally, t_θ/t_A is the smallest speed-up guaranteed to outperform any power optimised version of θ.

POSE results can be given in either relative or absolute forms by taking the ratio or the difference between values. For example, an optimisation guaranteed to outperform θ in terms of M must reduce runtime by at least $t_\theta - t_B$ seconds, or equivalently yield a relative speed-up of t_θ/t_B times. Expressions for POSE coordinates are all linear functions in terms of t_θ, meaning the ratios between them remain constant regardless of changes to runtime. This property means relative results can be used to predict large-scale optimisation characteristics from tests with shorter runtimes.

The results given by POSE are all bounds, and the true benefits of power optimisation will be more modest in practice. Even so, these values are useful as they allow performance engineers to make informed decisions about where best to focus their optimisation efforts.

One final thing to note is how metric tuning parameters affect POSE models. Figure 5.3 shows how POSE varies in response to different Et^n exponents ranging from Energy (Et^0) up to Energy Delay Cubed Product (Et^3). Higher values of n place more emphasis on runtime, resulting in less scope for energy-aware optimisation. POSE is able to reflect this change through its various insights and identify exactly how much the opportunity for energy-aware optimisation has been reduced by.

5.3 POSE Models for Novel Metrics

The previous sections introduced POSE in the context of the Et^n family of metrics. This section demonstrates that POSE is metric agnostic, and therefore

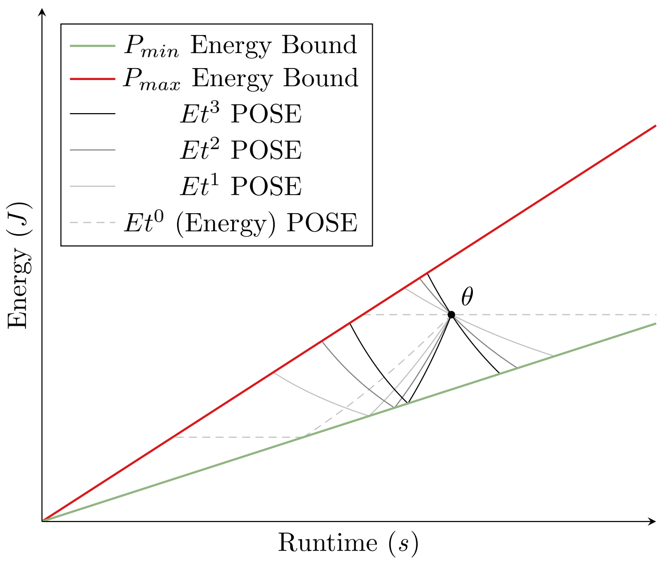


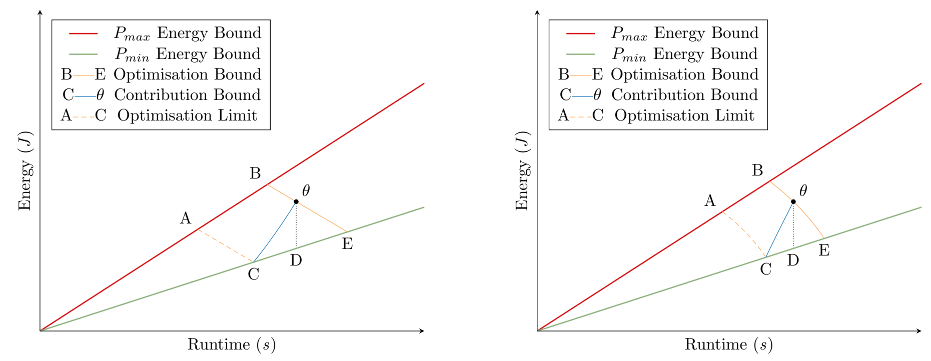
Figure 5.3: Et^n POSE Model Tunability

more generally applicable, by constructing models for the EDS and EDD metrics introduced in Chapter 4.

Many elements of POSE model construction are common between different metrics. The FPE remains the same as it is a property of the system under investigation. Similarly, vertex D does not move because it only depends on the FPE and the code being profiled, not the metric chosen. Furthermore, all POSE models consist of the same set of bounds and coordinates. As a result, the same insights are provided regardless of the optimisation metric used.

The only things that differ between metrics are the equations for the various POSE bounds and coordinates. These equations must be derived independently for each new metric used in conjunction with POSE. That said, these derivations follow similar patterns and as such are not a significant barrier.

Figure 5.4 displays POSE models for EDS and EDD drawn using the bounds derived below in this section.



(a) Energy Delay Sum (b) Energy Delay Distance

Figure 5.4: POSE Models for Novel Metrics

5.3.1 Energy Delay Sum POSE

The optimisation bound and optimisation limit are both determined by the same equation, as seen with Equation 5.1 for the Et^n metrics. Equation 5.11 links all points λ with the same EDS metric value as a reference code θ. It is noteworthy that Equation 5.11 is analgous to the point-slope form of the straight line equation, $y - y_1 = m(x - x_1)$, with $y_1 = E_\theta$, $x_1 = t_\theta$ and $m = -\beta/\alpha$.

$$\begin{aligned}
M(\lambda) &= M(\theta) \\
\alpha E_\lambda + \beta t_\lambda &= \alpha E_\theta + \beta t_\theta \\
\alpha E_\lambda &= \alpha E_\theta + \beta t_\theta - \beta t_\lambda \\
E_\lambda &= E_\theta + \frac{\beta t_\theta - \beta t_\lambda}{\alpha} \\
E_\lambda &= E_\theta + \frac{\beta}{\alpha}\left(t_\theta - t_\lambda\right) \qquad (5.11)
\end{aligned}$$

The process of finding POSE vertex coordinates is also the same for different metrics. Vertex B lies on the intersection between the optimisation and P_{max} energy bounds, at the point where Equation 5.11 crosses the line $P_{max}\, t_\lambda$. Its runtime coordinate is found by equating the two expressions then solving for t_λ

to give Equation 5.12 as follows:

$$\begin{aligned}
P_{max}\, t_\lambda &= E_\theta + \frac{\beta}{\alpha}\,(t_\theta - t_\lambda) \\
P_{max}\, t_\lambda &= P_\theta\, t_\theta + \frac{\beta}{\alpha}\, t_\theta - \frac{\beta}{\alpha}\, t_\lambda && (\text{As } E_\theta = P_\theta\, t_\theta) \\
P_{max}\, t_\lambda + \frac{\beta}{\alpha}\, t_\lambda &= P_\theta\, t_\theta + \frac{\beta}{\alpha}\, t_\theta \\
t_\lambda \left(P_{max} + \frac{\beta}{\alpha}\right) &= t_\theta \left(P_\theta\ + \frac{\beta}{\alpha}\right) \\
t_\lambda &= t_\theta \frac{P_\theta\ + \frac{\beta}{\alpha}}{P_{max} + \frac{\beta}{\alpha}} && (5.12)
\end{aligned}$$

The energy coordinate of vertex B is found by multiplying its runtime coordinate by P_{max}. Equation 5.13 lists the runtime and energy coordinates for vertex B under EDS:

$$\begin{aligned}
t_B &= t_\theta \frac{P_\theta\ + \frac{\beta}{\alpha}}{P_{max} + \frac{\beta}{\alpha}} \\
E_B &= P_{max} \cdot t_B
\end{aligned} \qquad (5.13)$$

As before, the derivation for the coordinates of vertex E is identical to Equation 5.12, except that P_{min} replaces P_{max} as E lies on the P_{min} energy bound. Equation 5.14 lists the runtime and energy coordinates for vertex E:

$$\begin{aligned}
t_E &= t_\theta \frac{P_\theta\ + \frac{\beta}{\alpha}}{P_{min} + \frac{\beta}{\alpha}} \\
E_E &= P_{min} \cdot t_E
\end{aligned} \qquad (5.14)$$

The contribution bound links all points λ where power and runtime contribute to M in the same ratio as the original code. Equation 5.15 shows the derivation of this bound for the EDS metric:

$$\begin{aligned}
M(P_\lambda t_\theta, t_\theta) &= M(P_\theta t_\lambda, t_\lambda)\\
\alpha P_\lambda t_\theta + \beta t_\theta &= \alpha P_\theta t_\lambda + \beta t_\lambda\\
t_\theta\,(\alpha P_\lambda + \beta) &= t_\lambda\,(\alpha P_\theta + \beta)\\
\alpha P_\lambda + \beta &= \frac{t_\lambda}{t_\theta}\,(\alpha P_\theta + \beta)\\
\alpha P_\lambda &= \frac{t_\lambda}{t_\theta}\,(\alpha P_\theta + \beta) - \beta\\
P_\lambda &= \frac{t_\lambda}{t_\theta}\left(P_\theta + \frac{\beta}{\alpha}\right) - \frac{\beta}{\alpha}\\
E_\lambda &= \frac{{t_\lambda}^2}{t_\theta}\left(P_\theta + \frac{\beta}{\alpha}\right) - t_\lambda\,\frac{\beta}{\alpha} \qquad (5.15)
\end{aligned}$$

Vertex C lies on the intersection between Equation 5.15 and the P_{min} energy bound given by $P_{min}\,t_\lambda$. The runtime coordinate of C is found by equating the two expressions and then re-arranging the result in terms of t_λ to yield Equation 5.16 as follows:

$$\begin{aligned}
P_{min}\,t_\lambda &= \frac{{t_\lambda}^2}{t_\theta}\left(P_\theta + \frac{\beta}{\alpha}\right) - t_\lambda\,\frac{\beta}{\alpha}\\
P_{min} &= \frac{t_\lambda}{t_\theta}\left(P_\theta + \frac{\beta}{\alpha}\right) - \frac{\beta}{\alpha}\\
P_{min} + \frac{\beta}{\alpha} &= \frac{t_\lambda}{t_\theta}\left(P_\theta + \frac{\beta}{\alpha}\right)\\
\frac{t_\lambda}{t_\theta} &= \frac{P_{min} + \frac{\beta}{\alpha}}{P_\theta + \frac{\beta}{\alpha}}\\
t_\lambda &= t_\theta\frac{P_{min} + \frac{\beta}{\alpha}}{P_\theta + \frac{\beta}{\alpha}} \qquad (5.16)
\end{aligned}$$

The energy coordinate of vertex C is found by multiplying its runtime coordinate by P_{min}. Equation 5.17 lists the runtime and energy coordinates for vertex C under EDS:

$$\begin{aligned}
t_C &= t_\theta\frac{P_{min} + \frac{\beta}{\alpha}}{P_\theta + \frac{\beta}{\alpha}}\\
E_C &= P_{max}\cdot t_C
\end{aligned} \qquad (5.17)$$

Both the contribution bound and contribution limit are defined by the same equation, but with a different reference point. As such, the coordinates for vertex A are given by the same equation as vertex B, namely Equation 5.13, except with C replacing θ to yield Equation 5.18. The expression for t_C in Equation 5.17 is then substituted in and the result rearranged to give Equation 5.19, an expression for the coordinate T_A in terms of θ, as follows:

$$t_A = t_C \frac{P_C + \frac{\beta}{\alpha}}{P_{max} + \frac{\beta}{\alpha}} \tag{5.18}$$

$$t_A = t_C \frac{P_{min} + \frac{\beta}{\alpha}}{P_{max} + \frac{\beta}{\alpha}} \qquad \text{(As } P_C = P_{min}\text{)}$$

$$t_A = t_\theta \frac{P_{min} + \frac{\beta}{\alpha}}{P_\theta + \frac{\beta}{\alpha}} \cdot \frac{P_{min} + \frac{\beta}{\alpha}}{P_{max} + \frac{\beta}{\alpha}} \qquad \text{(By Equation 5.17)}$$

$$t_A = t_\theta \, \frac{\left(P_{min} + \frac{\beta}{\alpha}\right)^2}{\left(P_\theta + \frac{\beta}{\alpha}\right)\left(P_{max} + \frac{\beta}{\alpha}\right)} \tag{5.19}$$

The energy coordinate of vertex A is found by multiplying its runtime coordinate by P_{max}. Equation 5.20 lists the runtime and energy coordinates for vertex A:

$$\begin{aligned} t_A &= t_\theta \, \frac{\left(P_{min} + \frac{\beta}{\alpha}\right)^2}{\left(P_\theta + \frac{\beta}{\alpha}\right)\left(P_{max} + \frac{\beta}{\alpha}\right)} \\ E_A &= P_{max} \cdot t_A \end{aligned} \tag{5.20}$$

5.3.2 Energy Delay Distance POSE

The optimisation bound and optimisation limit are both determined by Equation 5.21, which links all points λ with the same EDD metric value as a reference code θ:

$$\begin{aligned}
M(\lambda) &= M(\theta) \\
\sqrt{(\alpha E_\lambda)^2 + (\beta t_\lambda)^2} &= \sqrt{(\alpha E_\theta)^2 + (\beta t_\theta)^2} \\
(\alpha E_\lambda)^2 + (\beta t_\lambda)^2 &= (\alpha E_\theta)^2 + (\beta t_\theta)^2 \\
(\alpha E_\lambda)^2 &= (\alpha E_\theta)^2 + (\beta t_\theta)^2 - (\beta t_\lambda)^2 \\
{E_\lambda}^2 &= {E_\theta}^2 + \left(\frac{\beta}{\alpha}\right)^2 \left({t_\theta}^2 - {t_\lambda}^2\right) \\
E_\lambda &= \sqrt{{E_\theta}^2 + \left(\frac{\beta}{\alpha}\right)^2 \left({t_\theta}^2 - {t_\lambda}^2\right)} \qquad (5.21)
\end{aligned}$$

Vertex B lies on the intersection between the optimisation and P_{max} energy bounds, at the point where Equation 5.21 crosses the line $P_{max}\, t_\lambda$. Its runtime coordinate is found by equating the two expressions then solving for t_λ to give Equation 5.22 as follows. Note that only positive square roots are considered as runtime and energy costs cannot be negative.

$$\begin{aligned}
P_{max}\, t_\lambda &= \sqrt{{E_\theta}^2 + \left(\frac{\beta}{\alpha}\right)^2 \left({t_\theta}^2 - {t_\lambda}^2\right)} \\
(P_{max}\, t_\lambda)^2 &= (P_\theta\, t_\theta)^2 + \left(\frac{\beta}{\alpha}\right)^2 \left({t_\theta}^2 - {t_\lambda}^2\right) \qquad (\text{As } E_\theta = P_\theta\, t_\theta) \\
(P_{max}\, t_\lambda)^2 + \left(\frac{\beta}{\alpha}\right)^2 {t_\lambda}^2 &= (P_\theta\, t_\theta)^2 + \left(\frac{\beta}{\alpha}\right)^2 {t_\theta}^2 \\
{t_\lambda}^2 \left({P_{max}}^2 + \left(\frac{\beta}{\alpha}\right)^2\right) &= {t_\theta}^2 \left({P_\theta}^2 + \left(\frac{\beta}{\alpha}\right)^2\right) \\
{t_\lambda}^2 &= {t_\theta}^2 \frac{{P_\theta}^2 + \left(\frac{\beta}{\alpha}\right)^2}{{P_{max}}^2 + \left(\frac{\beta}{\alpha}\right)^2} \\
t_\lambda &= t_\theta \cdot \sqrt{\frac{{P_\theta}^2 + \left(\frac{\beta}{\alpha}\right)^2}{{P_{max}}^2 + \left(\frac{\beta}{\alpha}\right)^2}} \qquad (5.22)
\end{aligned}$$

The energy coordinate of vertex B is found by multiplying its runtime coordinate by P_{max}. Equation 5.23 lists the runtime and energy coordinates for vertex B

under EDD:

$$\begin{aligned} t_B &= t_\theta \cdot \sqrt{\frac{{P_\theta}^2 + \left(\frac{\beta}{\alpha}\right)^2}{{P_{max}}^2 + \left(\frac{\beta}{\alpha}\right)^2}} \\ E_B &= P_{max} \cdot t_B \end{aligned} \tag{5.23}$$

As always, the derivation for the coordinates of vertex E is identical to Equation 5.22, except that P_{min} replaces P_{max} as E lies on the P_{min} energy bound. Equation 5.24 lists the runtime and energy coordinates for vertex E:

$$\begin{aligned} t_E &= t_\theta \cdot \sqrt{\frac{{P_\theta}^2 + \left(\frac{\beta}{\alpha}\right)^2}{{P_{min}}^2 + \left(\frac{\beta}{\alpha}\right)^2}} \\ E_E &= P_{min} \cdot t_E \end{aligned} \tag{5.24}$$

The contribution bound links all points λ where power and runtime contribute to M in the same ratio as the original code. Equation 5.25 shows the derivation of this bound for the EDD metric:

$$\begin{aligned} M(P_\lambda t_\theta, t_\theta) &= M(P_\theta t_\lambda, t_\lambda) \\ \sqrt{(\alpha P_\lambda\, t_\theta)^2 + (\beta t_\theta)^2} &= \sqrt{(\alpha P_\theta\, t_\lambda)^2 + (\beta t_\lambda)^2} \\ (\alpha P_\lambda\, t_\theta)^2 + (\beta t_\theta)^2 &= (\alpha P_\theta\, t_\lambda)^2 + (\beta t_\lambda)^2 \\ (\alpha P_\lambda\, t_\theta)^2 &= (\alpha P_\theta\, t_\lambda)^2 + (\beta t_\lambda)^2 - (\beta t_\theta)^2 \\ {P_\lambda}^2 &= \left(P_\theta \frac{t_\lambda}{t_\theta}\right)^2 + \left(\frac{\beta\, t_\lambda}{\alpha\, t_\theta}\right)^2 - \left(\frac{\beta}{\alpha}\right)^2 \\ P_\lambda &= \sqrt{\left(P_\theta \frac{t_\lambda}{t_\theta}\right)^2 + \left(\frac{\beta\, t_\lambda}{\alpha\, t_\theta}\right)^2 - \left(\frac{\beta}{\alpha}\right)^2} \\ E_\lambda &= t_\lambda \cdot \sqrt{\left(P_\theta \frac{t_\lambda}{t_\theta}\right)^2 + \left(\frac{\beta\, t_\lambda}{\alpha\, t_\theta}\right)^2 - \left(\frac{\beta}{\alpha}\right)^2} \end{aligned} \tag{5.25}$$

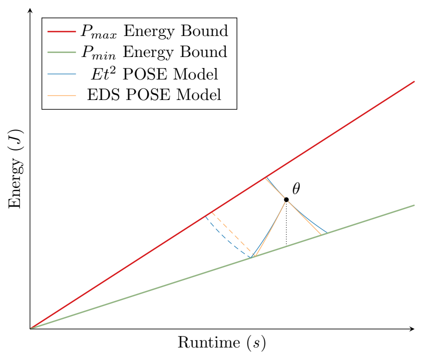
Vertex C lies on the intersection between Equation 5.25 and the P_{min} energy bound given by $P_{min}\, t_\lambda$. The runtime coordinate of C is found by equating the two expressions and then re-arranging the result in terms of t_λ to yield Equation 5.26 as follows:

$$\begin{aligned}
P_{min}\, t_\lambda &= t_\lambda \cdot \sqrt{\left(P_\theta \frac{t_\lambda}{t_\theta}\right)^2 + \left(\frac{\beta\, t_\lambda}{\alpha\, t_\theta}\right)^2 - \left(\frac{\beta}{\alpha}\right)^2} \\
P_{min} &= \sqrt{\left(P_\theta \frac{t_\lambda}{t_\theta}\right)^2 + \left(\frac{\beta\, t_\lambda}{\alpha\, t_\theta}\right)^2 - \left(\frac{\beta}{\alpha}\right)^2} \\
{P_{min}}^2 &= \left(P_\theta \frac{t_\lambda}{t_\theta}\right)^2 + \left(\frac{\beta\, t_\lambda}{\alpha\, t_\theta}\right)^2 - \left(\frac{\beta}{\alpha}\right)^2 \\
{P_{min}}^2 + \left(\frac{\beta}{\alpha}\right)^2 &= \frac{{t_\lambda}^2}{{t_\theta}^2}\left({P_\theta}^2 + \left(\frac{\beta}{\alpha}\right)^2\right) \\
{t_\lambda}^2 &= {t_\theta}^2 \, \frac{{P_{min}}^2 + \left(\frac{\beta}{\alpha}\right)^2}{{P_\theta}^2 + \left(\frac{\beta}{\alpha}\right)^2} \\
t_\lambda &= t_\theta \cdot \sqrt{\frac{{P_{min}}^2 + \left(\frac{\beta}{\alpha}\right)^2}{{P_\theta}^2 + \left(\frac{\beta}{\alpha}\right)^2}} \qquad (5.26)
\end{aligned}$$

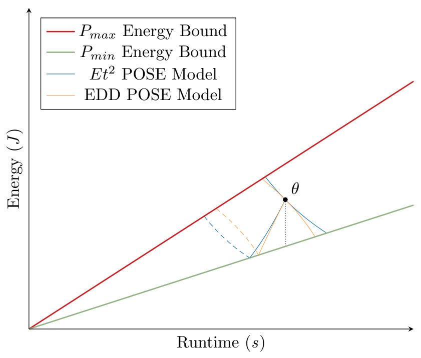
The energy coordinate of vertex C is found by multiplying its runtime coordinate by P_{min}. Equation 5.27 lists the runtime and energy coordinates for vertex C under EDD:

$$\begin{aligned}
t_C &= t_\theta \cdot \sqrt{\frac{{P_{min}}^2 + \left(\frac{\beta}{\alpha}\right)^2}{{P_\theta}^2 + \left(\frac{\beta}{\alpha}\right)^2}} \qquad (5.27) \\
E_C &= P_{min} \cdot t_C
\end{aligned}$$

Finally, the coordinate for vertex A is obtained by replacing C for θ in Equation 5.23 to yield Equation 5.28. The expression for t_C in Equation 5.27 is then substituted in and the result rearranged to give Equation 5.29, an expression



(a) EDS



(b) EDD

Figure 5.5: Comparison of POSE Models for Different Metrics

for the coordinate t_A in terms of θ, as follows:

$$t_A = t_C \cdot \sqrt{\frac{{P_C}^2 + \left(\frac{\beta}{\alpha}\right)^2}{{P_{max}}^2 + \left(\frac{\beta}{\alpha}\right)^2}} \tag{5.28}$$

$$t_A = t_\theta \cdot \sqrt{\frac{{P_{min}}^2 + \left(\frac{\beta}{\alpha}\right)^2}{{P_\theta}^2 + \left(\frac{\beta}{\alpha}\right)^2}} \cdot \sqrt{\frac{{P_{min}}^2 + \left(\frac{\beta}{\alpha}\right)^2}{{P_{max}}^2 + \left(\frac{\beta}{\alpha}\right)^2}}$$

$$t_A = t_\theta \cdot \frac{{P_{min}}^2 + \left(\frac{\beta}{\alpha}\right)^2}{\sqrt{{P_\theta}^2 + \left(\frac{\beta}{\alpha}\right)^2} \cdot \sqrt{{P_{max}}^2 + \left(\frac{\beta}{\alpha}\right)^2}} \tag{5.29}$$

The energy coordinate of vertex A is found by multiplying its runtime coordinate by P_{max}. Equation 5.30 lists the runtime and energy coordinates for vertex A:

$$\begin{aligned} t_A &= t_\theta \cdot \frac{{P_{min}}^2 + \left(\frac{\beta}{\alpha}\right)^2}{\sqrt{{P_\theta}^2 + \left(\frac{\beta}{\alpha}\right)^2} \cdot \sqrt{{P_{max}}^2 + \left(\frac{\beta}{\alpha}\right)^2}} \\ E_A &= P_{max} \cdot t_A \end{aligned} \tag{5.30}$$

Appendix A summarises all of the POSE equations derived in this chapter.

Figures 5.5a and 5.5b show how POSE models for the EDD and EDS metrics compare to the same model built for Et^2. The parameterisations used for these diagrams were based on those from the previous chapter in that values for the α and β coefficients were chosen to reflect the relative costs of an Et^n metric, in this case Et^2. The only difference from the previous chapter is that an Et^n exponent of two was used, meaning that a $2:1$ ratio was applied instead of the $3:1$ ratio used in Chapter 4.

Figure 5.5 also highlights an important difference between Et^n and the metrics introduced in Chapter 4. The parameterisations used for the EDS and EDD POSE models in this diagram were chosen to mirror the relative energy/time costs of Et^2. As a result, the gradients of their optimisation bounds at point θ are the same as for Et^2. Even so, the optimisation bound for Et^2 diverges from the other metrics, moving further away from the origin and suggesting a larger scope for energy-aware optimisation.

This divergence happens because Et^n metrics produce perverse optimisation incentives. As discussed in Chapter 4, Et^n places more emphasis on energy optimisations for efficient codes and on runtime optimisations for fast codes. Any small optimisation which improves energy efficiency will increase the apparent benefits of further energy optimisations, leading to the concave curvature of the optimisation bounds for Et^n metrics.

Avoiding perverse optimisation incentives was a key design principle for both EDS and EDD. They do not over-emphasize energy optimisation for efficient codes or runtime optimisations for fast ones. As a result, POSE models built for these metrics will show less opportunity for energy-aware optimisation than equivalent models built for Et^n metrics if eqivalent parameterisations are used.

5.4 POSE Investigation

This section uses POSE to investigate the energy-aware optimisation characteristics of codes from the Mantevo [56] mini-application benchmark suite. Experiments were carried out on the Taurus system operated by TU Dresden. Results were gathered using the High Density Energy Efficiency Monitoring (HDEEM) instrumentation infrastructure present on Taurus [51].

Taurus is a heterogeneous cluster with several different classes of node. Work was carried out on the largest of these classes, with each node featuring dual twelve core Intel Xeon E5-2680 v3 Central Processing Units (CPUs) and 64 GB of memory. This choice of platform and power measurement technique was motivated solely by availability as POSE places no restrictions on either.

5.4.1 Feasible Performance Envelope

The first step when applying POSE is to construct a feasible performance envelope. Hardware manufacturers usually publish power dissipation figures for their systems, however these are estimates which may not be observed in practice.

Thermal Design Power (TDP) figures are widely available but they specify the upper limits of safe operation; raising power draw above these limits can cause damage to CPU circuitry. TDP is an upper limit which real-world power consumption may not be able to match, especially if any safety margins were built in to the system's design to improve reliability.

POSE works best when the power bounds are as tight as possible. In the absence of concrete P_{min} and P_{max} figures for available hardware, the decision was taken to determine P_{min} and P_{max} empirically.

This work follows the convention of specifying power benchmarks using (S, A, C) tuples, with P-state S, activity factor A and active core count C. These three components determine the power consumption of Complimentary Metal Oxide Semiconductor (CMOS) circuitry as explained by Section 3.3. Briefly, the P-state is a *(frequency, voltage)* pair selected by Dynamic Voltage and Frequency Scaling (DVFS) logic in order to achieve some performance goal, while the activity factor is the average fraction of logic elements which change state on each clock cycle.

Benchmarks for P_{min} and P_{max} should reflect the full range of values that the elements of a (S, A, C) tuple could take for a given code θ. This notion is formalised by Equation 5.31.

$$
\begin{aligned}
P_{max} &= (S_{max}, A_{max}, C_{max} \mid \theta) \\
P_{min} &= (S_{min}, A_{min}, C_{min} \mid \theta)
\end{aligned}
\tag{5.31}
$$

The values of S, A, and C depend on the code to be optimised and the nature of the optimisations being considered. POSE models for inherently serial codes should be constructed using single threaded benchmarks, for example, so that $C_{min} = C_{max} = 1$.

Taurus supports the `libcpufreq` library, which allows its users to override DVFS controls and manually set the desired P-state S. The number of active cores C was controlled by specifying the number of threads used by the benchmarking routines and pinning each one to its own core to prevent thread migration.

Specially chosen benchmark codes were used to reach activity factors A_{min} and A_{max}. Although it is defined as a scalar between zero and one, the range of values which activity factor can take is more limited in practice. The range of values which A can take for some fixed S and C can be defined as $[\alpha,\ \beta]$ where $0 < \alpha < \beta < 1$.

A custom assembly micro-benchmark was developed for $A = \alpha$ which executes a single `jmp` instruction each clock cycle. This code prevents instruction pipelining, does not perform any calculations or memory accesses and keeps control logic to a minimum.

Non-trivial codes perform more work per unit time than this minimal $A = \alpha$ benchmark. Additional work means more transistors changing state per cycle, and hence a higher activity factor. The only exception occurs when applications are blocked for long periods, allowing the processor to enter an idle state. This can be addressed by adding delays to the benchmark.

FIRESTARTER [50] was used as the benchmark for activity factor β. This tool is designed to trigger peak power consumption on x86_64 based servers. It consists of hand optimised assembly routines which raise the activity factor

Table 5.1: Single Node Feasible Performance Envelope Parameters

	Blade Power (W)		CPU Power (W)	
Frequency (GHz)	P_{max}	P_{min}	P_{max}	P_{min}
2.5	345.33	207.68	116.44	65.79
2.4	343.24	182.74	116.30	55.97
2.3	341.50	175.40	116.23	51.56
2.2	340.42	169.52	116.13	49.35
2.1	334.84	163.58	113.81	47.84
2.0	325.44	159.20	109.13	44.96
1.9	309.33	153.91	102.95	42.25
1.8	290.65	151.56	95.02	42.24
1.7	278.77	138.96	88.40	36.85
1.6	266.83	136.97	83.36	35.24
1.5	260.57	135.44	76.76	34.65
1.4	256.35	133.61	73.14	34.03
1.3	254.44	132.31	65.20	33.04
1.2	251.31	128.93	61.22	30.80

above the level achievable with high level languages. Prime95 and Linpack were also evaluated as β benchmarks.

The benchmark parameter space was small enough to measure power draw for every (S, A, C) configuration. Benchmarking runs lasted for 400 seconds, allowing sufficient time for power readings to stabilise. Table 5.1 shows the results for a single fully occupied node ($C = 24$). This table identifies P-states by their frequency component.

Having built a Feasible Performance Envelope for Taurus, the next step in this investigation was to capture energy and runtime figures for real applications. The Mantevo application suite was chosen because it covers a broad range of scientific computing workloads.

All codes were compiled with the Intel C++ Compiler (ICC) version 15.0.3. Application parameters were based on default values, with problem sizes tuned where necessary to ensure reasonable run times on single nodes. Values for these parameters are given in Appendix C. Each application was run fifteen times on the same node to reduce the impact of random variations in runtime and energy.

5.4.2 POSE Models for Code Optimisation

The first experiment carried out investigates single node code performance. Table 5.2 lists the mean energy and runtime costs incurred by running codes

Table 5.2: Code Metrics for $S = 2.5\,\text{GHz}$, $C = 24$

Code	Runtime (s)	Energy (J)	Et^2	EDS	EDD
TeaLeaf	323.8	99,810.3	10,464,754,630	391,230	195,629
PathFinder	337.1	71,943.9	8,175,446,517	375,334	189,361
CloverLeaf	214.3	57,861.2	2,657,246,101	250,731	125,489
CloverLeaf 3D	153.1	43,755.9	1,025,621,231	181,546	90,792
MiniMD	125.5	31,162.1	490,810,866	144,112	72,275
CoMD	105.6	24,837.8	276,975,249	119,878	60,231
MiniFE	36.7	8,465.6	11,402,232	41,496	20,864
HPCCG	36.5	8,059.5	10,737,269	40,910	20,607

from the Mantevo suite. Each of these applications was observed to keep all 24 cores active when run on a single node, and to spend the vast majority of its runtime operating at the highest available P-State. The decision was taken to consider optimisations which did not reduce parallelism (C = 24) or decrease processor throughput (S = 2.5 GHz). This corresponds to the feasible performance envelope for Taurus nodes given by Equation 5.32.

$$\begin{aligned} P_{max} &= (2.5\,\text{GHz}, \beta, 24) = 345.33\,\text{W} \\ P_{min} &= (2.5\,\text{GHz}, \alpha, 24) = 207.68\,\text{W} \end{aligned} \tag{5.32}$$

The remainder of this section focusses on TeaLeaf, MiniMD and PathFinder. These codes cover the full range of mini-application power consumption. POSE models for TeaLeaf and PathFinder are reproduced graphically in Figures 5.6a and 5.6b respectively, and model summaries are presented in Table 5.3. POSE results for the remaining codes can be found in Appendix B.
These results show that TeaLeaf is the code most amenable to power optimisation in terms of both scope and benefit. PathFinder has very little to gain from such optimisation, as illustrated by the difference in scale between Figure 5.6a and Figure 5.6b. All other Mantevo applications fall somewhere between these two extremes.

Table 5.4 shows POSE model summaries for MiniMD built for both the EDS and EDD metrics introduced in Chapter 4. These results highlight the fact that these new metrics are less prone to over-emphasize the opportunities for energy-aware optimisation for already efficient codes.

Table 5.3: Et^2 POSE Model Summaries

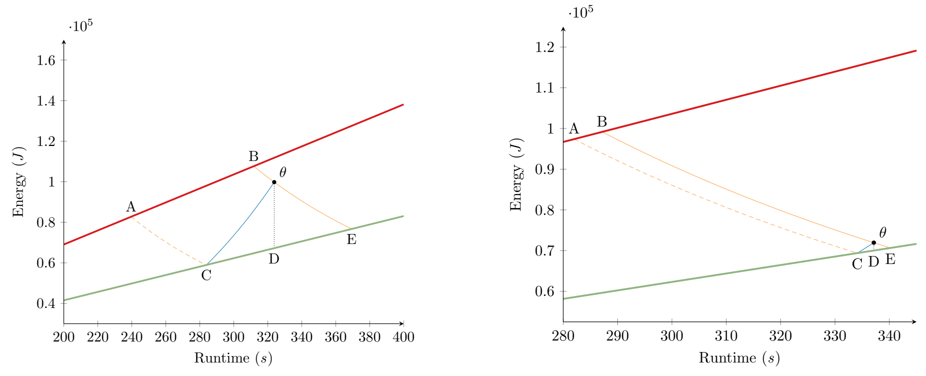
TeaLeaf	
Maximum Energy Saved by Reduced Power Consumption	32 560 J; 1.48×
Maximum Improvement in Et^2 from Power Optimisation	2.2×
Worst Case Slowdown as a result of Power Optimisation	45.55 s; 1.14×
Minimum Speed-up Guaranteed to Outperform θ	12.04 s; 1.04×
Speed-up Required to Dominate Power Optimisation	84.2 s; 1.35×
MiniMD	
Maximum Energy Saved by Reduced Power Consumption	4913 J; 1.16×
Maximum Improvement in Et^2 from Power Optimisation	1.35×
Worst Case Slowdown as a result of Power Optimisation	7.50 s; 1.05×
Minimum Speed-up Guaranteed to Outperform θ	16.58 s; 1.13×
Speed-up Required to Dominate Power Optimisation	28.93 s; 1.25×
PathFinder	
Maximum Energy Saved by Reduced Power Consumption	1928 J; 1.03×
Maximum Improvement in Et^2 from Power Optimisation	1.06×
Worst Case Slowdown as a result of Power Optimisation	3.07 s; 1.01×
Minimum Speed-up Guaranteed to Outperform θ	49.98 s; 1.17×
Speed-up Required to Dominate Power Optimisation	55.13 s; 1.20×

Table 5.4: MiniMD POSE Models for Novel Metrics

EDS	
Maximum Energy Saved by Reduced Power Consumption	4913 J; 1.16×
Maximum Improvement in E^1t^2 from Power Optimisation	1.06×
Worst Case Slowdown as a result of Power Optimisation	4.44 s; 1.03×
Minimum Speed-up Guaranteed to Outperform θ	12.29 s; 1.09×
Speed-up Required to Dominate Power Optimisation	20.09 s; 1.16×
EDD	
Maximum Energy Saved by Reduced Power Consumption	4913 J; 1.16×
Maximum Improvement in E^1t^2 from Power Optimisation	1.05×
Worst Case Slowdown as a result of Power Optimisation	3.48 s; 1.02×
Minimum Speed-up Guaranteed to Outperform θ	12.03 s; 1.09×
Speed-up Required to Dominate Power Optimisation	18.19 s; 1.14×

5.4.3 POSE Models for Frequency Scaling

The relationship between P-state and energy consumption is non-linear and workload dependent [83]. It has been shown that application-aware DVFS can save energy by selecting the optimal P-state schedule for a given code [20]. This implies that the reverse also holds; code changes may affect the optimal P-state



(a) TeaLeaf (b) PathFinder

Figure 5.6: Et^2 POSE Comparison of TeaLeaf and PathFinder

assignment. The second experiment was carried out to demonstrate how POSE can be used to reason about this class of optimisation.

The 2.5 GHz P-state was again used to gather the P_{max} baseline because this is the P-State which Taurus defaults to when running TeaLeaf or MiniMD. Because this experiment considers changes to P-State, the lowest power draw P-State of 1.2 GHz was chosen as the P_{min} benchmark. Equation 5.33 gives the corresponding feasible performance envelope.

$$\begin{aligned} P_{max} &= (2.5\,\text{GHz}, \beta, 24) = 345.33\,\text{W} \\ P_{min} &= (1.2\,\text{GHz}, \alpha, 24) = 128.96\,\text{W} \end{aligned} \tag{5.33}$$

The runtime and energy consumption of TeaLeaf and MiniMD were measured for each P-state supported by the Haswell CPUs on Taurus. Figure 5.7 illustrates how these two codes respond differently to changes in CPU frequency. TeaLeaf is tightly bound by memory bandwidth, and Figure 5.7a shows how switching to lower P-States can reduce Et^2 for TeaLeaf.

Figure 5.7b shows that MiniMD performance is sensitive to CPU frequency, with lower frequencies leading to longer runtimes. Despite initial reductions in energy consumption, the lowest Et^2 value for MiniMD occurs at 2.5 GHz, meaning race-to-halt is the optimal DVFS strategy for this code.

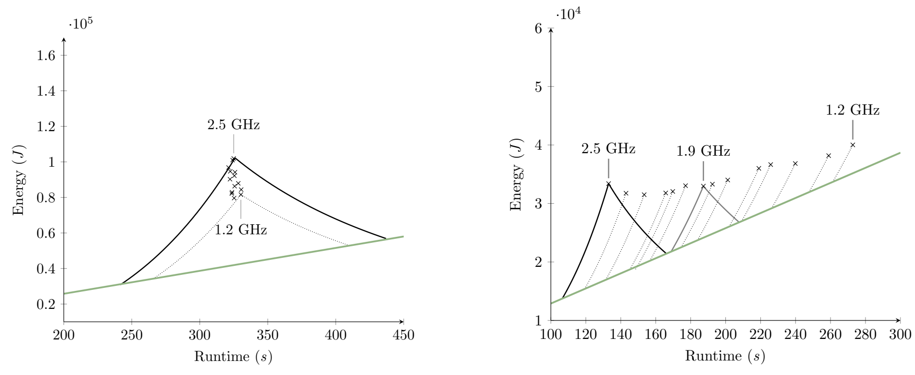


Figure 5.7: Et^2 POSE for P-State Optimisation of TeaLeaf and MiniMD

While useful, this simple analysis fails to consider co-optimisation of activity factor and P-state for MiniMD. It is possible that different software optimisations may be required to achieve optimal performance in different P-states. The flexibility of POSE allows us to model this scenario by considering optimisations which can impact P-state as well as activity factor.

If two P-states have overlapping POSE models then it may be possible for a power optimised version of the code running at the lower frequency P-state to outperform the original code running at the higher frequency. Conversely, if their POSE models do not overlap then no amount of power optimisation will be able to match the benefits of simply switching to the higher performance P-state. This analysis allows dominated P-States to be excluded from the search for power optimisations.

For MiniMD, Figure 5.7b shows that the first non-overlapping POSE model occurs at 1.9 GHz. This means that all power optimised versions of MiniMD operating at or below 1.9 GHz will have strictly worse performance than the original, unoptimised code running at 2.5 GHz. Power optimisation is therefore only worth pursuing at frequencies between 2.5 GHz and 2.0 GHz.

Dynamic Concurrency Throttling has also been proposed as a means to reduce energy consumption [25]. POSE could be used to model such optimisations

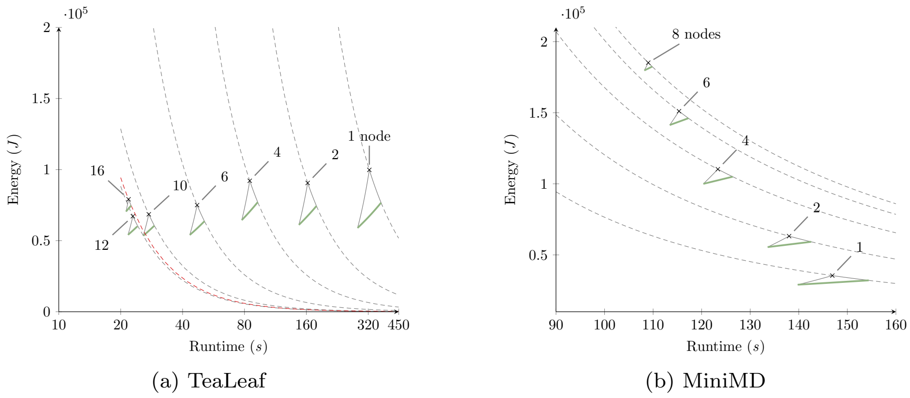


Figure 5.8: Et^2 POSE for Multi-Node Runs of TeaLeaf and MiniMD

in a similar manner to the P-state investigation; the only difference being the parameterisation of the feasible performance envelope ($C_{min} = 1$).

5.4.4 POSE Models for Distributed Codes

Distributing programs across multiple nodes is a common strategy for improving the throughput of scientific applications. Figure 5.8 shows the results of strong scaling studies for the TeaLeaf and MiniMD mini-applications. In both cases distributing the same problem size over progressively more nodes reduces time to completion as well as the power draw of individual nodes.

The dashed optimisation bounds drawn in Figure 5.8 connect all points which share the same Et^2 values as the scale runs. Figure 5.8a shows that increasing the number of nodes for TeaLeaf improves Et^2 up to between 12 and 16 nodes for the example problem size. Beyond this point the increase in power draw from adding extra nodes dominates improvements in runtime.

Figure 5.8b shows that increasing the number of nodes for MiniMD leads to strictly greater energy consumption. Furthermore, the increase in energy consumption outpaces improvements in runtime, meaning Et^2 performance diminishes as nodes are added.

A key observation for both codes is that the baseline rate of power consumption increases linearly as extra nodes are added, while application power draw per node shrinks. Consequently, the opportunity for power optimisation diminishes as the scale at which a code is run increases.

5.5 Summary

This chapter presents POSE, a mathematical and visual modelling tool which captures the trade off between software power consumption and runtime. POSE provides insights regarding the scope a code has for power optimisation as well as the level of improvement which can be expected. These insights help developers to determine whether power or runtime optimisation is the best approach for improving the efficiency of a code.

POSE works by partitioning the energy/runtime plane into areas corresponding to runtime and power optimised versions of an initial code with respect to an optimisation metric. This chapter provides derivations of the POSE boundaries for Energy Delay Product (Et^n) metrics and outlines the various insights these models provide.

POSE was then demonstrated by modelling the power consumption of codes taken from the Mantevo mini-application suite running on Taurus. The results gathered are expected to be of interest to performance engineers and serve to demonstrate the practical utility of POSE.

The first experiment showed that PathFinder offers the least scope for power optimisation, with Et^2 improvements limited to $1.06\times$. Runtime optimisation is therefore the only realistic approach to improving the performance of this code. TeaLeaf has the most scope for power optimisation, with potential improvements in the same metric of up to $2.2\times$. Power optimisation is worth considering for this code.

The second experiment was carried out to demonstrate how POSE can be used to reason about application specific and P-state optimisations. This investigation showed that no power optimised version of MiniMD operating at

P-states below 2.0 GHz can match the Et^2 performance of the original unoptimised code running at the default 2.5 GHz P-state. TeaLeaf was also found to be extremely insensitive to CPU frequency, meaning that application-aware DVFS may deliver significant energy savings for this code.

CHAPTER 6
System Summary POSE

Ordinary Power Optimised Software Envelope (POSE) models quantify the scope which exists for the energy-aware optimisation of a specific code running on a given system. This chapter introduces System Summary POSE, an extension of POSE that allows developers to reason about system-wide power optimisation characteristics without reference to any particular code.

Ordinary POSE models use system P_{max} and P_{min} energy bounds together with the energy and runtime costs incurred when running a code to calculate the scope that code has for power and runtime optimisation. System Summary POSE is a meta-heuristic which determines the range of results conventional POSE models could produce for a given system. This "bound-of-bounds" approach allows developers to understand the scope a system has for energy-aware software optimisation independent of the code being run.

This chapter begins by introducing System Summary POSE in the context of the Et^n family of metrics. It then shows how they, like ordinary POSE models, can also be used in conjunction with the novel metrics introduced in Chapter 4. Finally, System Summary POSE is used to comment on the power optimisation characteristics of the Taurus supercomputer.

6.1 System Summary POSE Derivation

System Summary POSE examines how the insights provided by POSE models vary in response to changes in the initial code θ. Increasing the power consumption of a code while keeping its metric value fixed leads to a corresponding increase in the scope for power optimisation. Figure 6.1 illustrates how such a change would be reflected in the output of a conventional POSE model.

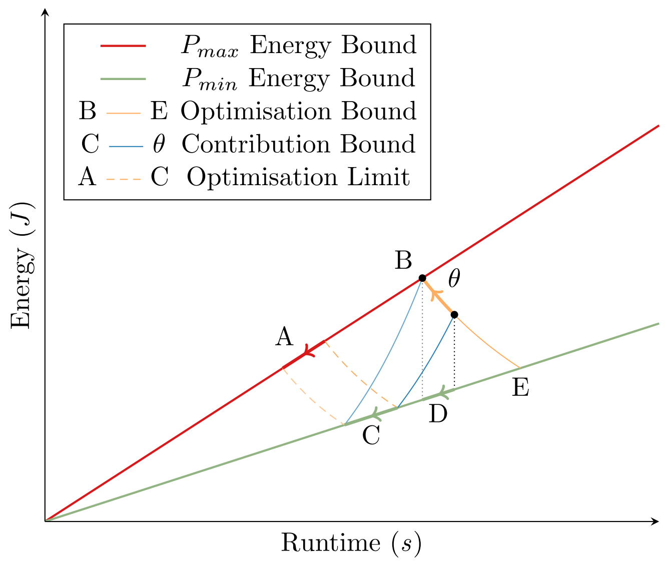


Figure 6.1: Et^2 System Summary POSE Intuition

System Summary POSE determines which point along the contribution bound $B - E$ maximises the value of each of the five key insights provided by POSE models. This maximum value then serves as an upper limit on the values which the corresponding insight could take for real codes running on the target system.

In practice, all POSE insights assume their maximum values at either vertex B or vertex E because these points correspond to extremes of power consumption. As such, another interpretation of System Summary POSE is as a pair of ordinary POSE models for the P_{min} and P_{max} energy benchmarks.

Ordinary POSE models require four input parameters; the P_{min} and P_{max} values which define a feasible performance envelope and the energy and runtime costs for a specific code. A key feature of the relative forms of POSE insights is that their runtime terms always cancel. Furthermore, the power draws at vertices B and E are by definition P_{min} and P_{max} respectively. As a result, System Summary POSE is able to derive system-wide power optimisation limits from just two unknowns, namely the values for P_{min} and P_{max}.

The first relative POSE insight, E_θ/E_D, places an upper limit on the amount of energy which can be saved by reducing power consumption. Figure 6.1 makes it clear that this value is maximised when $\theta = B$ and therefore $P_\theta = P_{max}$. Intuitively, the code with the most to gain from energy optimisation is the one which exhibits the highest rate of power consumption. Substituting in $P_\theta = P_{max}$ into the definition of the first insight yields the following expression for system-wide energy savings:

$$\begin{aligned}
\arg\max_\theta \; \frac{E_\theta}{E_D} &= B \\
\frac{E_B}{E_D} &= \frac{P_{max} \cdot t_\theta}{P_{min} \cdot t_\theta} \\
&= \frac{P_{max}}{P_{min}}
\end{aligned} \tag{6.1}$$

The second relative POSE insight, $M(\theta)/M(C)$, limits the maximum improvement in a metric which can be attributed to power optimisation. This value depends on the metric used, however for any valid metric (a monotonically increasing function of time and energy) this value is again maximised when θ is at point B. Substituting in $P_\theta = P_{max}$ and $P_C = P_{min}$ yields the following system-wide bound which holds for all Et^n metrics:

$$\begin{aligned}
\arg\max_\theta \; \frac{M(\theta)}{M(C)} &= B \\
\frac{M(B)}{M(C)} &= \frac{E_\theta \, {t_\theta}^n}{E_C \, {t_C}^n} \\
&= \frac{P_{max} \, {t_\theta}^{n+1}}{P_{min} \, {t_C}^{n+1}}
\end{aligned}$$

Equation 5.7 from the previous chapter is then used to express t_C in terms of t_θ, yielding:

$$\begin{aligned}
\frac{M(B)}{M(C)} &= \frac{{P_{max}}^2 \, {t_\theta}^{n+1}}{{P_{min}}^2 \, {t_\theta}^{n+1}} \\
&= \left(\frac{P_{max}}{P_{min}}\right)^2
\end{aligned} \tag{6.2}$$

The third relative POSE insight, t_θ/t_B, represents the smallest speed-up which guarantees a code that outperforms θ with respect to M. Uniquely, this value is maximised when θ runs at minimum power, and is therefore located at point E. This is because any speed-up at all would guarantee an improvement in terms of M for codes with maximum power consumption P_{max}. The derivation of this system-wide bound for Et^n metrics is as follows:

$$\begin{aligned}
\arg\max_{\theta} \ \frac{t_\theta}{t_B} &= E \\
\frac{t_E}{t_B} &= \frac{t_\theta \left(\frac{P_\theta}{P_{min}}\right)^{\frac{1}{n+1}}}{t_\theta \left(\frac{P_\theta}{P_{max}}\right)^{\frac{1}{n+1}}} && \text{(By Equations 5.3 and 5.4)} \\
&= \left(\frac{P_{max}}{P_{min}}\right)^{\frac{1}{n+1}} && (6.3)
\end{aligned}$$

The fourth relative POSE insight, t_E/t_θ, represents the maximum slowdown which could be traded off to achieve a slower yet more energy efficient code. This insight is maximised at vertex B because this point has the most scope for power optimisation. As a result, this system-wide bound takes on the same value as Equation 6.3:

$$\begin{aligned}
\arg\max_{\theta} \ \frac{t_E}{t_\theta} &= B \\
\frac{t_E}{t_B} &= \left(\frac{P_{max}}{P_{min}}\right)^{\frac{1}{n+1}} && (6.4)
\end{aligned}$$

The final relative POSE insight, t_θ/t_A, represents the smallest speed-up guaranteed to outperform any power optimised version of θ. This insight is once again maximised at vertex B because this point has the most scope for power optimisations and as such larger runtime optimisations are required in order to guarantee they outperform all possible power optimsations. The derivation of this system-wide bound for Et^n metrics is as follows:

$$
\begin{aligned}
\underset{\theta}{\arg\max}\ \frac{t_\theta}{t_A} &= B \\
\frac{t_B}{t_A} &= \frac{t_\theta \left(\frac{P_\theta}{P_{max}}\right)^{\frac{1}{n+1}}}{t_\theta \left(\frac{{P_{min}}^2}{P_\theta\, P_{max}}\right)^{\frac{1}{n+1}}} \\
\frac{t_B}{t_A} &= \frac{1}{\left(\frac{{P_{min}}^2}{{P_{max}}^2}\right)^{\frac{1}{n+1}}} && \text{(As } P_\theta = P_{max}\text{)} \\
&= \left(\frac{P_{max}}{P_{min}}\right)^{\frac{2}{n+1}} && (6.5)
\end{aligned}
$$

Equations 6.1 – 6.5 highlight a number of interesting properties. Equation 6.1 does not depend on the metric used, as it deals exclusively with energy savings and does not consider runtime. Equation 6.2 shows that the runtime exponent n does not influence the degree to which power optimisation can improve an Et^n metric. The fact that Equations 6.3 and 6.4 are identical shows that the maximum slowdown from power optimisation is the same as the smallest speed-up which is guaranteed to improve performance in terms of M. Most significantly, all of these equations only depend on P_{max} and P_{min}. As a result, System Summary POSE analysis can be carried out on any system for which these parameters are known.

6.2 System Summary POSE for Novel Metrics

The previous sections introduced System Summary POSE in the context of the Et^n family of metrics. This section provides similar derivations which allow System Summary POSE to be used in conjunction with the Energy Delay Sum (EDS) and Energy Delay Distance (EDD) metrics introduced in Chapter 4.

The first relative POSE insight does not depend on the metric used. Therefore, the first system-wide bound given by Equation 6.1 also applies to both EDS and EDD. The following subsections provide derivations for the remaining four system-wide bounds for use in conjunction with the EDS and EDD metrics.

6.2.1 Energy Delay Sum System Summary POSE

The second relative POSE insight, $M(\theta)/M(C)$, is maximised when $\theta = B$, meaning that $P_\theta = P_{max}$. The derivation of its maximum value for EDS is as follows:

$$\begin{aligned}
\frac{M(\theta)}{M(C)} &= \frac{\alpha E_\theta + \beta t_\theta}{\alpha E_C + \beta t_C} \\
&= \frac{t_\theta}{t_C} \cdot \frac{P_\theta + \frac{\beta}{\alpha}}{P_{min} + \frac{\beta}{\alpha}} \\
&= \frac{t_\theta}{t_C} \cdot \frac{P_{max} + \frac{\beta}{\alpha}}{P_{min} + \frac{\beta}{\alpha}} && \text{(As } P_\theta = P_{max}\text{)} \\
\frac{t_\theta}{t_C} &= \frac{P_\theta + \frac{\beta}{\alpha}}{P_{min} + \frac{\beta}{\alpha}} && \text{(By Equation 5.17)} \\
\therefore \quad \frac{M(\theta)}{M(C)} &= \left(\frac{P_{max} + \frac{\beta}{\alpha}}{P_{min} + \frac{\beta}{\alpha}}\right)^2 && (6.6)
\end{aligned}$$

The third relative POSE insight, t_θ/t_B, is maximised when $\theta = E$, meaning that $P_\theta = P_{min}$. The derivation of its maximum value for EDS is as follows:

$$\begin{aligned}
\frac{t_\theta}{t_B} &= \frac{t_\theta}{t_\theta \frac{P_\theta + \frac{\beta}{\alpha}}{P_{max} + \frac{\beta}{\alpha}}} && \text{(By Equation 5.13)} \\
&= \frac{t_\theta}{t_\theta \frac{P_{min} + \frac{\beta}{\alpha}}{P_{max} + \frac{\beta}{\alpha}}} && \text{(As } P_\theta = P_{min}\text{)} \\
&= \frac{P_{max} + \frac{\beta}{\alpha}}{P_{min} + \frac{\beta}{\alpha}} && (6.7)
\end{aligned}$$

The fourth relative POSE insight, t_E/t_θ, is maximised when $\theta = B$, meaning that its maximum value for EDS is also given by Equation 6.7.

The final relative POSE insight, t_θ/t_A, is also maximised when $\theta = B$ and therefore $P_\theta = P_{max}$. The derivation of its maximum value for EDS is as follows:

$$\begin{aligned}
\frac{t_\theta}{t_A} &= \frac{t_\theta}{t_\theta \frac{\left(P_{min} + \frac{\beta}{\alpha}\right)^2}{\left(P_\theta + \frac{\beta}{\alpha}\right)\left(P_{max} + \frac{\beta}{\alpha}\right)}} && \text{(By Equation 5.20)} \\
&= \frac{1}{\frac{\left(P_{min} + \frac{\beta}{\alpha}\right)^2}{\left(P_{max} + \frac{\beta}{\alpha}\right)^2}} && (\text{As } P_\theta = P_{max}) \\
&= \left(\frac{P_{max} + \frac{\beta}{\alpha}}{P_{min} + \frac{\beta}{\alpha}}\right)^2 && (6.8)
\end{aligned}$$

6.2.2 Energy Delay Distance System Summary POSE

The second relative POSE insight, $M(\theta)/M(C)$, is maximised when $\theta = B$, meaning that $P_\theta = P_{max}$. The derivation of its maximum value for EDD is as follows:

$$\begin{aligned}
\frac{M(\theta)}{M(C)} &= \frac{\sqrt{(\alpha E_\theta)^2 + (\beta t_\theta)^2}}{\sqrt{(\alpha E_C)^2 + (\beta t_C)^2}} \\
&= \frac{\sqrt{{t_\theta}^2\left({P_\theta}^2 + \left(\frac{\beta}{\alpha}\right)^2\right)}}{\sqrt{{t_C}^2\left({P_{min}}^2 + \left(\frac{\beta}{\alpha}\right)^2\right)}} \\
&= \frac{t_\theta}{t_C} \cdot \sqrt{\frac{{P_{max}}^2 + \left(\frac{\beta}{\alpha}\right)^2}{{P_{min}}^2 + \left(\frac{\beta}{\alpha}\right)^2}} && (\text{As } P_\theta = P_{max}) \\
\frac{t_\theta}{t_C} &= \frac{t_\theta}{t_\theta \cdot \sqrt{\frac{{P_{min}}^2 + \left(\frac{\beta}{\alpha}\right)^2}{{P_\theta}^2 + \left(\frac{\beta}{\alpha}\right)^2}}} && \text{(By Equation 5.27)} \\
&= \sqrt{\frac{{P_{max}}^2 + \left(\frac{\beta}{\alpha}\right)^2}{{P_{min}}^2 + \left(\frac{\beta}{\alpha}\right)^2}} && (\text{As } P_\theta = P_{max}) \\
\therefore \quad \frac{M(\theta)}{M(C)} &= \frac{{P_{max}}^2 + \left(\frac{\beta}{\alpha}\right)^2}{{P_{min}}^2 + \left(\frac{\beta}{\alpha}\right)^2} && (6.9)
\end{aligned}$$

The third relative POSE insight, t_θ/t_B, is maximised when $\theta = E$, meaning that $P_\theta = P_{min}$. The derivation of its maximum value for EDD is as follows:

$$\begin{aligned}
\frac{t_\theta}{t_B} &= \frac{t_\theta}{t_\theta \cdot \sqrt{\frac{{P_\theta}^2+\left(\frac{\beta}{\alpha}\right)^2}{{P_{max}}^2+\left(\frac{\beta}{\alpha}\right)^2}}} && \text{(By Equation 5.23)}\\
&= \frac{1}{\sqrt{\frac{{P_{min}}^2+\left(\frac{\beta}{\alpha}\right)^2}{{P_{max}}^2+\left(\frac{\beta}{\alpha}\right)^2}}} && \text{(As } P_\theta = P_{min}\text{)}\\
&= \sqrt{\frac{{P_{max}}^2+\left(\frac{\beta}{\alpha}\right)^2}{{P_{min}}^2+\left(\frac{\beta}{\alpha}\right)^2}} && (6.10)
\end{aligned}$$

The fourth relative POSE insight, t_E/t_θ, is maximised when $\theta = B$, meaning that its maximum value for EDD is also given by Equation 6.10.

The final relative POSE insight, t_θ/t_A, is also maximised when $\theta = B$ and therefore $P_\theta = P_{max}$. The derivation of its maximum value for EDD is as follows:

$$\begin{aligned}
\frac{t_\theta}{t_A} &= \frac{t_\theta}{t_\theta \cdot \frac{{P_{min}}^2+\left(\frac{\beta}{\alpha}\right)^2}{\sqrt{{P_\theta}^2+\left(\frac{\beta}{\alpha}\right)^2}\cdot\sqrt{{P_{max}}^2+\left(\frac{\beta}{\alpha}\right)^2}}} && \text{(By Equation 5.30)}\\
&= \frac{1}{\left(\frac{{P_{min}}^2+\left(\frac{\beta}{\alpha}\right)^2}{{P_{max}}^2+\left(\frac{\beta}{\alpha}\right)^2}\right)} && \text{(As } P_\theta = P_{max}\text{)}\\
&= \frac{{P_{min}}^2+\left(\frac{\beta}{\alpha}\right)^2}{{P_{max}}^2+\left(\frac{\beta}{\alpha}\right)^2} && (6.11)
\end{aligned}$$

6.3 System Summary POSE Investigation

This section uses System Summary POSE to investigate the scope for power optimisation on the Taurus supercomputer. The feasible performance envelope for a Taurus node with 64 GB of memory is given by Equation 6.12, which is reproduced from the previous chapter.

$$\begin{aligned} P_{max} &= (2.5\,\text{GHz}, \beta, 24) = 345.33\,\text{W} \\ P_{min} &= (2.5\,\text{GHz}, \alpha, 24) = 207.68\,\text{W} \end{aligned} \tag{6.12}$$

Substituting these values into Equations 6.1 – 6.5 shows that power optimisation can deliver at most a 1.66× reduction in compute node energy consumption, and improve Et^2 by at most 2.76×. Furthermore, a 1.18× reduction in runtime is guaranteed to lead to a better Et^2 performance, and an increase of the same magnitude is guaranteed to reduce performance by the same metric. Finally, a 1.40× reduction in runtime is guaranteed to beat any power optimisation.

$$\begin{aligned} P_{max} &= (2.5\,\text{GHz}, \alpha, 24) = 116.44\,\text{W} \\ P_{min} &= (2.5\,\text{GHz}, \alpha, 24) = 65.79\,\text{W} \end{aligned} \tag{6.13}$$

The same analysis can be repeated for individual subsystems as well as entire nodes. Equation 6.13 gives the feasible performance envelopes for the Intel Xeon E5-2680 v3 Central Processing Units (CPUs) found in Taurus nodes. Inside this envelope, power optimisation can deliver at most a 1.77× reduction in CPU energy consumption, and improve Et^2 by at most 3.13×. Furthermore, a 1.21× reduction in runtime is guaranteed to lead to a better Et^2 performance, and an increase of the same magnitude is guaranteed to reduce performance by the same metric. Finally, a 1.46× reduction in runtime is guaranteed to beat any power optimisation in terms of Et^2.

CPU energy consumption accounts for a significant portion of the energy used by high performance systems [42]. It is therefore unsurprising that System Summary POSE yields similar values for Taurus nodes and the CPUs they contain. That said, being able to build POSE models for individual components is useful because the results can be transferred to other machines which contain the same hardware.

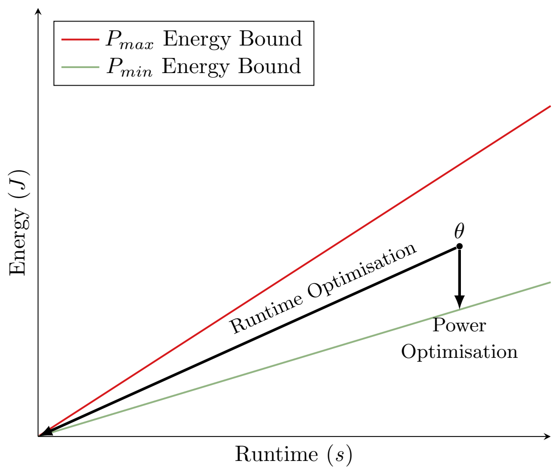


Figure 6.2: Optimisation Limits

6.4 Optimisation Study

System Summary POSE highlights a fundamental yet often overlooked distinction between runtime and power optimisation. While conventional optimisation aims to reduce runtime towards zero, power optimisation is constrained by the P_{min} power limit. Figure 6.2 illustrates this distinction. This section seeks to investigate how tight the System Summary bounds are in practice.

The lack of a runtime counterpart to the P_{min} limit in this diagram suggests that the scope for runtime optimisation will always exceed that for power optimisation. While this is often the case, it is not true in general because runtime optimisation also has limits; even the fastest codes require some non-zero amount of time to finish. The real distinction is that power limits are system-wide, while runtime limits are application specific.

System Summary POSE has shown that improvements from power optimisation are limited to around $2-3\times$ on Taurus nodes regardless of the code being optimised. Conversely, while runtime optimisation limits do exist, speed-ups of $100\times$ or more are not unheard of.

An experiment was carried out to illustrate the difference in scope for runtime and power optimisation. Stencil operations, which are a common pattern

Algorithm 1 Finite Difference Laplacian Diffusion Code

```
for t = 1 to timesteps do
    for k = 1 to kmax-1 do                                          ▷ Z axis
        for j = 1 to jmax-1 do                                      ▷ Y axis
            for i = 1 to imax-1 do                                  ▷ X axis
                $C^t_{i,j,k} = C^{t-1}_{i,j,k}$
                    $+ X \left( C^{t-1}_{i-1,j,k} - 2C^{t-1}_{i,j,k} + C^{t-1}_{i+1,j,k} \right)$
                    $+ Y \left( C^{t-1}_{i,j-1,k} - 2C^{t-1}_{i,j,k} + C^{t-1}_{i,j+1,k} \right)$
                    $+ Z \left( C^{t-1}_{i,j,k-1} - 2C^{t-1}_{i,j,k} + C^{t-1}_{i,j,k+1} \right)$
            end for
        end for
    end for
    UPDATEBOUNDARIES($C^t$)
end for
```

in numerical simulations, form the basis of this study. Algorithm 1 describes a nine-point stencil code which implements a finite difference scheme to solve the diffusion equation.

A reference version of the diffusion algorithm in Algorithm 1 was implemented in the C programming language. Reflective boundary conditions were used as they help with code validation. Conservation laws dictate that, while the distribution may change, the total amount of a conserved quantity remains constant in isolated systems. This property acts as a useful sanity check when developing diffusion solvers.

OpenMP was then used to parallelise the outer spatial (Z axis) loop to create a baseline parallel version. All codes were compiled using Intel C++ Compiler (ICC) version 15.0.3. with relevant optimisations enabled, including automatic loop vectorisation. This level of parallelism reflects a typical starting point for performance engineering as further improvements require more invasive code changes supported by specialist tools and experience.

Several optimisations were applied to the baseline parallel application in order to chart the effect these optimisations had on runtime and power consumption. Results were gathered on Taurus, following the same procedure used in previous experiments. The problem size was configured as a 3D grid of $800 \times 800 \times 800$ cells, simulated over 600 timesteps.

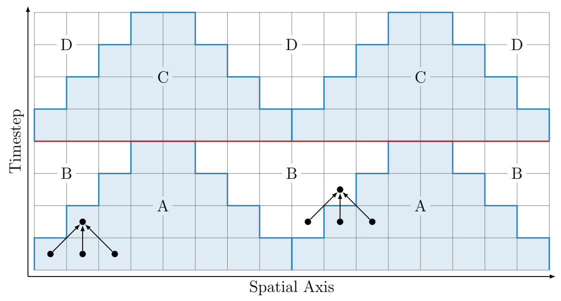


Figure 6.3: 1D Trapezoidal Decomposition

The first successful optimisation was achieved by switching from using the OpenMP library to Cilk Plus. Cilk Plus is fully integrated into the ICC compiler, allowing it to avoid the overhead associated with using external libraries.

The second optimisation was achieved by applying loop tiling. Loop tiling partitions the simulation's spatial domain into contiguous blocks in order to improve data locality and caching. It does this by transforming the three nested spatial loops into three outer and three inner loops. The outer loops step through blocks, while the inner loops iterate over the cells in each block.

The third optimisation was a switch back to OpenMP. While Cilk Plus has lower overhead, OpenMP offers more control over loop iteration scheduling, especially for nested loops. This increased control led to better overall performance for the loop tiled version of the sample code.

The fourth optimisation involved implementing a cache-oblivious space-time decomposition scheme for stencil codes adapted from one proposed by McCool et al. [91]. This scheme recursively partitions the simulation along its three spatial and one temporal dimensions. The resulting trapezoidal regions are then distributed across different threads to be processed in parallel.

Figure 6.3 shows the decomposition scheme applied in one spatial and one temporal dimension. Trapezoids are processed in alphabetical order, meaning different areas of the simulation domain advance at different times. This scheme works because each cell only depends on its immediate neighbourhood from the previous timestep. As the diagram shows, cells within this neighbourhood

Optimisation	Description	Runtime	Power	Energy
-	Reference	675.3	140.9	95,149.3
1	OMP Parallel	339.9	244.7	83,174.4
2	Cilk Parallel	208.9	255.5	53,379.9
3	Cilk Loop Tiling	162.4	247.0	40,109.3
4	OMP Loop Tiling	156.0	251.9	39,297.1
5	OMP Trapezoid	83.6	265.7	22,202.1
6	Cilk Trapezoid	81.2	268.6	21,811.7

Table 6.1: Optimisation Impact

are either in the same trapezoid as the update cell or in a different trapezoid which has already been completed. This scheme maximises spatial locality while minimising inter-thread synchronisation and data dependencies.

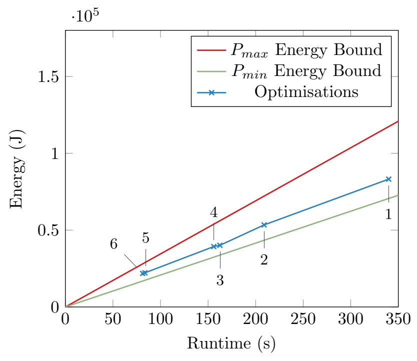
The fifth and final optimisation was a switch back to Cilk Plus. Cilk Plus uses advanced work sharing algorithms which allow it to distribute trapezoids across threads more efficiently than OpenMP.

Table 6.1lists the costs associated with each version of the application. Overall, optimisation achieved a 4.2× reduction in runtime and a 3.81× reduction in energy consumption. The difference between these figures is due to an increase in power consumption from 244.7 J to 268.8 J, confirming the intuition that runtime optimisation negatively impacts power consumption. These runtime optimisations delivered a 66.8× improvement in Et^2; far above the system limit of 2.76× improvement possible from power optimisation. Figure 6.4 illustrates the optimisation process in the runtime/energy and runtime/power domains.

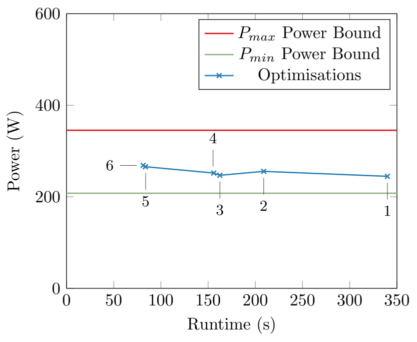
6.5 Summary

This chapter introduced System Summary POSE, a bound-of-bounds heuristic which places upper limits on the benefits which can be expected from power optimisation. This analysis works by calculating the range of results a conventional POSE model could potentially produce for a target system.

One of the results in this chapter showed that power optimisation could



(a) Energy Domain



(b) Power Domain

Figure 6.4: Laplacian Optimisation Progression

reduce the energy consumption of compute nodes by at most 2.76× on the target platform. Another important result was that a runtime optimisation of 1.18× or greater was guaranteed to outperform any possible power optimisation in terms of Et^2.

This section concludes with an optimisation study into a simple stencil code in order to provide some context for the System Summary POSE limits. A stencil code was chosen because this is a very common algorithmic pattern in numerical simulations, and also because its simplicity allows compilers to apply some optimisations automatically.

Despite taking every effort to help the compiler, hand-optimising the code still delivered significant benefits, reducing runtime by 4.2×. As this exceeds the 1.18× limit identified by System Summary POSE, it is possible to categorically state that the changes made outperform any possible power optimisations on the target platform. This result highlights the importance of having realistic expectations about the benefits of energy-aware code optimisation.

While runtime optimisation was the correct strategy for the stencil code chosen, this is not always going to be the case. Some codes are less amenable to runtime optimisation than others, especially if they are already highly optimised. System Summary POSE allows developers to gauge the potential benefits for power optimisation and, when combined with their experience, choose whether it is worth pursuing on a given platform.

CHAPTER 7
Conclusions and Future Work

Numerical simulations have become indispensable tools in many areas of science and engineering. Performance engineers optimise these simulations by tuning them to take advantage of specific hardware. Higher performance means more calculations can be carried out, which in turn allows domain experts to increase the size, complexity and resolution of their simulations.

Historically, runtime was the main factor used to define the performance of High Performance Computing (HPC) applications. More recently, unsustainable increases in power draw have led energy consumption to join runtime as a primary constraint in HPC. Performance engineers are facing a future in which they must minimise both runtime and energy consumption in tandem. Existing tools must be updated and new tools must be developed in order to support this emerging class of optimisation.

The field of energy-aware performance optimisation is still in its infancy, characterised by ad-hoc techniques and a lack of standardised metrics. The work in this thesis has attempted to address these issues and provide a stronger foundation for others to build on.

Chapter 4 proposed new Figure of Merit (FoM) metrics intended to guide energy-aware software optimisation. The metrics currently used for this purpose were developed by the hardware community based on assumptions which are invalid for software optimisation.

Chapter 4 began by outlining desirable criteria for software optimisation metrics. Current metrics fail on all but one of these criteria, leaving them unable to drive optimisation in sensible directions or support fair comparison between different implementations. Worse still, these metrics do not provide a meaningful definition for code optimisation.

Two new metrics were proposed which address the problems in existing approaches. The first new metric, Energy Delay Sum (EDS), is a weighted sum of runtime and energy costs. The second new metric, Energy Delay Distance (EDD), uses Euclidean distance to define the utility of a code. Both of these metrics outperform existing alternatives against all of the assessment criteria, and EDS manages to satisfy the maximum possible number of these criteria.

Chapter 4 concluded by comparing the real-world performance of EDS and EDD against established Et^n metrics. The results confirmed the common criticism that Et^n metrics are biased towards extreme parallelism. Conversely, both EDS and EDD produced results which were intuitively justifiable.

Chapter 5 presented the Power Optimised Software Envelope (POSE) model. POSE allows developers to compare the potential benefits of energy and runtime optimisation and determine which approach is most suitable for their code.

POSE models provide insights about the scope a code has for energy-aware optimisation. These insights are: the maximum amount of energy which can be saved by reducing power consumption; the maximum improvement in an energy efficiency metric achievable by energy-aware performance optimisation; the largest increase in runtime which could be traded off to achieve a slower yet more energy efficient code; the smallest speed-up guaranteed to improve code performance irrespective of power draw; and finally, the smallest speed-up guaranteed to outperform any power optimisation.

Chapter 5 demonstrated the POSE model by studying the optimisation characteristics of codes from the Mantevo mini-application suite. TeaLeaf was found to have the most scope for single node power optimisation, with potential improvements in the Et^2 metric of up to $2.2\times$, equivalent to a 54.6% reduction. Conversely, PathFinder had the least scope for power optimisation with improvements to the same metric limited to $1.06\times$ which is equivalent to a 5.29% reduction. POSE was also used to explore how the scope for power optimisation varies in response to changes in clock frequency and node count.

Chapter 6 built on the POSE model by deriving system-wide limits for the benefits of energy-aware software optimisation. This chapter introduced System Summary POSE, a meta-heuristic which operates by determining the range of results that conventional POSE models could produce for a given system. Performance engineers can use System Summary POSE to compare different platforms based on the scope they offer for power optimisation.

The results in Chapter 6 showed that, for a particular class of x86 node, power optimisation is limited to reducing node-level energy consumption by at most 1.66×. This corresponds to a maximum improvement in Et^2 of 2.76×. These results also showed that speed-ups from conventional optimisation of 1.4× or more are guaranteed to outperform all possible energy-aware optimisations on the target platform. Results like these are useful because they allow performance engineers to focus their efforts where they will yield the greatest return.

Work in this thesis has been shared and used in a variety of contexts. Both the POSE model and the novel metrics work has been presented to researchers and developers at the Science and Technology Facilities Council (STFC) Hartree Centre and ARM. The novel metrics work in Chapter 4 is also due to be presented at a forthcoming workshop at Lawrence Berkeley National Laboratory.

In at least one case POSE has lead to the early abandonment of work because the potential benefits from power optimisation were smaller than anticipated. It is also currently being engineered for inclusion into Allinea MAP [68], a well-known state of the art application analytics tool for HPC clusters and applications. The metrics work has also gained some attention from the open source compiler community.

7.1 Thesis Limitations

This thesis focussed on the energy consumed by compute nodes in the course of running simulations. Supercomputers also expend large amounts of energy on secondary functions, most notably cooling. A common rule of thumb is that each Watt used by compute resources translates into an additional Watt lost to power supply inefficiencies and a further Watt required for cooling [74]. The methods used in this thesis are suitable for whole-system power analysis, however power figures for secondary subsystems are hard to come by.

Another limitation of this thesis is that it only considers a narrow range of platforms based on Intel x86 processors. This is due to the scarcity of power-instrumented supercomputing hardware. Care has been taken to ensure that the techniques presented are as general as possible in order to mitigate this limitation. The methods described in this thesis do not depend on any particular hardware platform, and empirical results have been backed up by algebraic proofs where possible.

A further limitation concerns the parameterisations used for the EDS and EDD metrics. The values used in this thesis were chosen to match the most commonly used exponents for Et^n metrics in order to facilitate comparison between these results. That said, it is not clear that these are the best values in practice and more work is required to determine the most appropriate parameterisations.

7.2 Future Work

Energy-aware performance engineering is a nascent field with much work still to be done. Many platforms do not report energy figures as standard, or do so at temporal resolutions too low to be useful for performance engineering. These limitations must be addressed before energy-aware performance optimisation can become widespread.

Work is also needed to adapt existing performance engineering tools to take energy into account. Energy-aware profilers and performance models are required to identify suitable optimisation opportunities. New classes of optimisation will be required to take advantage of future developments in energy-efficient hardware architectures.

The metrics introduced in Chapter 4 are well suited to comparing codes running at different scales and on different architectures. The quantitative nature of POSE models also makes them particularly suitable for comparison studies. Ongoing work at Warwick uses EDS and EDD metrics in conjunction with POSE models to investigate the power optimisation characteristics of various codes running on several different accelerator-based technologies, including Field-Programmable Gate Arrays (FPGAs), general purpose Graphics Processing Units (GPUs), and new ARM platforms for HPC.

Producing POSE models for novel platforms will show how suitable each of these platforms are for power optimisation. This work is also expected to demonstrate how POSE models may be used to identify specific optimisations. This will involve developing feasible performance envelopes for individual subsystems as well as for different types of kernel. Doing so would allow POSE to highlight optimisation opportunities at a per-kernel and per-subsystem level and hence facilitate targeted optimisation.

7.3 Final Remarks

Energy-aware performance optimisation has received much attention in recent times. Despite this, the field is still in its infancy and many fundamental issues remain to be addressed. The work in this thesis has shown that current optimisation metrics are unable to provide a meaningful definition for code optimisation. Furthermore, POSE models show that there are hard limits on how much energy-aware optimisation can deliver, which is a significant departure from conventional optimisation. These issues, and others like them, must be addressed in order for this field to flourish.

Looking forward, novel architectures and hardware/software co-design offer compelling opportunities to improve the energy efficiency of HPC systems. The emergence of ARM and other non-traditional vendors of HPC hardware points to a period of increasing heterogeneity and architectural diversity. New developments promise to blur the lines between Central Processing Units (CPUs) and accelerators. Processors featuring on-die FPGA fabrics are in development, and ARM and other CPU cores are being integrated directly into accelerator hardware. This diversity is fuelled by energy-efficiency concerns, and in turn offers a great deal of potential for energy-aware software optimisation.

References

[1] Advanced Micro Devices. BIOS and Kernel Developers Guide (BKDG) for AMD Family 15h Models 00h-0Fh Processors., 2012.

[2] Green500 List, November 2016. Online: `https://www.top500.org/green500/lists/2016/11/` (Accessed 2017-05-28).

[3] Top500 List, November 2016. Online: `https://www.top500.org/lists/2016/11/` (Accessed 2017-05-28).

[4] L. Adhianto and B. Chapman. Performance Modeling of Communication and Computation in Hybrid MPI and OpenMP Applications. *Simulation Modelling Practice and Theory*, 15(4):481–491, 2007.

[5] E. Alba and P. Vidal. Systolic Optimization on GPU Platforms. *Computer Aided Systems Theory*, pages 375–383, 2012.

[6] A. Alexandrov, F. I. Mihai, E. S. Klaus, and S. Chris. LogGP: Incorporating Long Messages into the LogP Model for Parallel Computation. *Journal of Parallel and Distributed Computing (JPDC '97)*, 44:71–79, 1997. ISSN 0743-7315.

[7] G. S. Almasi and A. Gottlieb. Highly Parallel Computing. 1988.

[8] G. M. Amdahl. Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities. In *Proceedings of the 1967 Joint Computer Conference*, pages 483–485. ACM, New York, NY, 1967.

[9] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, and S. W. Williams. The Landscape of Parallel Computing Research: A View from Berkeley. Technical report, Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, 2006.

[10] D. Bedard, M. Y. Lim, R. Fowler, and A. Porterfield. Powermon: Fine-Grained and Integrated Power Monitoring for Commodity Computer Systems. In *Proceedings of the 2010 IEEE SoutheastCon (SoutheastCon '10)*, pages 479–484. IEEE, 2010.

[11] C. Bekas and A. Curioni. A New Energy Aware Performance Metric. *Computer Science-Research and Development*, 25(3-4):187–195, 2010.

[12] B. D. Bingham and M. R. Greenstreet. Computation with Energy-Time Trade-Offs: Models, Algorithms and Lower Bounds. In *IEEE International Symposium on Parallel and Distributed Processing with Applications*, pages 143–152, 2008.

[13] W. L. Bircher and L. K. John. Complete System Power Estimation: A Trickle-Down Approach Based on Performance Events. In *International Symposium on Performance Analysis of Systems and Software (ISPASS '07)*, pages 158–168. IEEE, 2007.

[14] W. L. Bircher, M. Valluri, J. Law, and L. K. John. Runtime Identification of Microprocessor Energy Saving Opportunities. In *Proceedings of the 2005 International Symposium on Low Power Electronics and Design (ISLPED '05)*, pages 275–280. IEEE, 2005.

[15] R. Biswas, W. Thigpen, R. Ciotti, P. Mehrotra, C. Henze, J. Parks, B. Biegel, and R. Hood. Pleiades: NASA's First Petascale Supercomputer. In *Contemporary High Performance Computing: From Petascale toward Exascale*, chapter 12, pages 309–338. Chapman and Hall, 2013.

[16] L. Brochard, R. Panda, D. DeSota, F. Thomas, and R. H. Bell. Power and Energy-Aware Processor Scheduling. In *ACM SIGSOFT Software Engineering Notes*, volume 36, pages 227–234. ACM, 2011.

[17] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A Framework for Architectural-level Power Analysis and Optimizations. In *Proceedings*

of the 27th Annual International Symposium on Computer Architecture (ISCA'00), pages 83–94. ACM, New York, NY, June 2000.

[18] M. Burtscher, I. Zecena, and Z. Zong. Measuring GPU Power with the K20 Built-In Sensor. In *Proceedings of Workshop on General Purpose Processing Using GPUs*, page 28. ACM, 2014.

[19] J. W. Choi, D. Bedard, R. Fowler, and R. Vuduc. A Roofline Model of Energy. In *Proceedings of the IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*, pages 661–672, May 2013.

[20] K. Choi, R. Soma, and M. Pedram. Dynamic Voltage and Frequency Scaling Based on Workload Decomposition. In *Proceedings of the International Symposium on Low Power Electronics and Design (IPSLED'04)*, pages 174–179. ACM, New York, NY, August 2004.

[21] G. Chrysos. Intel Xeon Phi Coprocessor (codename Knights Corner). In *Proceedings of the 20th IEEE Hot Chips Symposium (HCS '12)*, pages 1–31. IEEE, 2012.

[22] R. C. Chu, R. E. Simons, and G. M. Chrysler. Experimental Investigation of an Enhanced Thermosyphon Heat Loop for Cooling of a High Performance Electronics Module. In *IEEE Semiconductor Thermal Measurement and Management Symposium*, pages 1–9. IEEE, 1999.

[23] Z. Cui, Y. Zhu, Y. Bao, and M. Chen. A Fine-Grained Component-Level Power Measurement Method. In *International Green Computing Conference (IGCC '11)*, pages 1–6. IEEE, 2011.

[24] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a Realistic Model of Parallel Computation. In *Proceedings of the Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPOPP'93)*, pages 1–12. ACM, New York, NY, May 1993.

[25] M. Curtis-Maury, J. Dzierwa, C. D. Antonopoulos, and D. S. Nikolopoulos. Online Strategies for High-performance Power-aware Thread Execution on Emerging Multiprocessors. In *Proceedings of the 20th International Conference on Parallel and Distributed Processing (IPDPS'06)*, pages 298–298, IPDPS, Rhodes Island, Greece, April 2006. IEEE Computer Society, Washington, DC.

[26] L. Dagum and R. Menon. OpenMP: an Industry Standard API for Shared-Memory Programming. *IEEE Computational Science and Engineering*, 5 (1):46–55, 1998.

[27] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le. RAPL: Memory Power Estimation and Capping. In *Proceedings of the International Symposium on Low-Power Electronics and Design (ISLPED '10)*, pages 189–194. IEEE, August 2010.

[28] R. H. Dennard, F. H. Gaensslen, V. L. Rideout, E. Bassous, and A. R. LeBlanc. Design of Ion-Implanted MOSFET's with Very Small Physical Dimensions. *IEEE Journal of Solid-State Circuits*, 9(5):256–268, 1974.

[29] R. Dimond, S. Racaniere, and O. Pell. Accelerating large-scale HPC Applications using FPGAs. In *Proceedings of the 20th IEEE Symposium on Computer Arithmetic (ARITH '11)*, pages 191–192. IEEE, 2011.

[30] J. J. Dongarra, P. Luszczek, and A. Petitet. The LINPACK benchmark: Past, Present and Future. *Concurrency and Computation: Practice and Experience*, 15(9):803–820, 2003.

[31] J. Eastep, S. Sylvester, C. Cantalupo, F. Ardanaz, B. Geltz, A. Al-Rawi, F. Keceli, and K. Livingston. Global Extensible Open Power Manager: A Vehicle for HPC Community Collaboration Toward Co-Designed Energy Management Solutions. In *International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems*, 2016.

[32] H. C. Edwards, C. R. Trott, and D. Sunderland. Kokkos: Enabling Many-core Performance Portability through Polymorphic Memory Access Patterns. *Journal of Parallel and Distributed Computing*, 74(12):3202–3216, 2014.

[33] R. Efraim, R. Ginosar, C. Weiser, and A. Mendelson. Energy Aware Race To Halt: A Down to EARTH Approach for Platform Energy Management. *IEEE Computer Architecture Letters*, 13(1):25–28, 2014.

[34] E. N. Elnozahy, R. Bianchini, T. El-Ghazawi, A. Fox, F. Godfrey, A. Hoisie, K. McKinley, R. Melhem, J. Plank, P. Ranganathan, and J. Simons. System Resilience at Extreme Scale. *Defense Advanced Research Project Agency (DARPA) Tech. Rep*, 2008.

[35] J. Enos, C. Steffen, J. Fullop, M. Showerman, G. Shi, K. Esler, V. Kindratenko, J. E. Stone, and J. C. Phillips. Quantifying the Impact of GPUs on Performance and Energy Efficiency in HPC Clusters. In *Proceedings of the 2010 International Green Computing Conference (IGC '10)*, pages 317–324. IEEE, 2010.

[36] S. Eranian. Perfmon2: a Flexible Performance Monitoring Interface for Linux. In *Proceedings of the 2006 Ottawa Linux Symposium*, pages 269–288. Citeseer, 2006.

[37] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger. Dark Silicon and the End of Multicore Scaling. In *Proceedings of the 38th Annual International Symposium on Computer Architecture (ISCA '11)*, pages 365–376. IEEE, 2011.

[38] W. Feng and K. Cameron. The Green500 List: Encouraging Sustainable Supercomputing. *Computer*, 40(12), 2007.

[39] M. J. Flynn. Some Computer Organizations and their Effectiveness. *IEEE Transactions on Computers*, 100(9):948–960, 1972.

[40] S. Fortune and J. Wyllie. Parallelism in Random Access Machines. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, pages 114–118. ACM, 1978.

[41] V. W. Freeh, D. K. Lowenthal, F. Pan, N. Kappiah, R. Springer, B. L. Rountree, and M. E. Femal. Analyzing the Energy-Time trade-off in High-Performance Computing Applications. *IEEE Transactions on Parallel and Distributed Systems*, 18(6):835–848, 2007.

[42] R. Ge, X. Feng, S. Song, H. Chang, D. Li, and K. W. Cameron. PowerPack: Energy Profiling and Analysis of High-Performance Systems and Applications. *IEEE Transactions on Parallel and Distributed Systems*, 21 (5):658–671, May 2010.

[43] M. B. Giles, G. R. Mudalige, Z. Sharif, G. Markall, and P. Kelly. Performance Analysis of the OP2 Framework on Many-Core Architectures. *ACM SIGMETRICS Performance Evaluation Review*, 38(4):9–15, 2011.

[44] M. B. Giles, G. R. Mudalige, B. Spencer, C. Bertolli, and I. Reguly. Designing OP2 for GPU Architectures. *Journal of Parallel and Distributed Computing*, 73(11):1451–1460, 2013.

[45] R. Gonzales and M. Horowitz. Energy Dissipation in General Purpose Processors. *IEEE Journal of Solid State Circuits*, 31:1277–1284, September 1996.

[46] N. Goswami, R. Shankar, M. Joshi, and T. Li. Exploring GPGPU Workloads: Characterization Methodology, Analysis and Microarchitecture Evaluation Implications. In *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC '10)*, pages 1–10. IEEE, 2010.

[47] J. L. Gustafson. Reevaluating Amdahl's law. *Communications of the ACM*, 31(5):532–533, 1988.

[48] D. Hackenberg and M. K. Patterson. Evaluation of a new data center air-cooling architecture: The down-flow Plenum. In *Proceedings of the 2016 IEEE Conference on Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm '16)*, pages 395–403. IEEE, 2016.

[49] D. Hackenberg, T. Ilsche, R. Schöne, D. Molka, M. Schmidt, and W. E. Nagel. Power Measurement Techniques on Standard Compute Nodes: A Quantitative Comparison. *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 194–204, March 2013.

[50] D. Hackenberg, R. Oldenburg, D. Molka, and R. Schöne. Introducing FIRESTARTER: A Processor Stress Test Utility. In *International Green Computing Conference (IGCC '13)*, pages 1–9, June 2013.

[51] D. Hackenberg, T. Ilsche, J. Schuchart, R. Schöne, W. E. Nagel, M. Simon, and Y. Georgiou. HDEEM: High Definition Energy Efficiency Monitoring. In *Energy Efficient Supercomputing Workshop (E2SC), 2014*, pages 1–10, November 2014.

[52] W. Haensch, E. J. Nowak, R. H. Dennard, P. M. Solomon, A. Bryant, O. H. Dokumaci, A. Kumar, X. Wang, J. B. Johnson, and M. V. Fischetti. Silicon CMOS Devices Beyond Scaling. *IBM Journal of Research and Development*, 50(4.5):339–361, 2006.

[53] M. Harman and J. Clark. Metrics are Fitness Functions Too. In *Proceedings of the International Symposium on Software Metrics*, pages 58–69, September 2004.

[54] M. Harman and B. F. Jones. Search-Based Software Engineering. *Information and Software Technology*, 43(14):833 – 839, 2001. ISSN 0950-5849.

[55] A. Hart, H. Richardson, J. Doleschal, T. Ilsche, M. Bielert, and M. Kappel. User-Level Power Monitoring and Application Performance on Cray XC30

Supercomputers. *Proceedings of the 2014 Cray User Group (CUG '14)*, 2014.

[56] M. A. Heroux, D. W. Doerfler, P. S. Crozier, J. M. Willenbring, H. C. Edwards, A. Williams, M. Rajan, E. R. Keiter, H. K. Thornquist, and R. W. Numrich. Improving Performance via Mini-Applications. *SNL Tech. Rep SAND2009-5574*, 2009.

[57] T. Hey, S. Tansley, and K. Tolle. *The Fourth Paradigm: Data-Intensive Scientific Discovery.* Microsoft Research, 2009.

[58] T. Hey, A. JG Hey, and G. Pápay. *The Computing Universe: A Journey Through a Revolution.* Cambridge University Press, 2014.

[59] G. Hinton, D. Sager, M. Upton, D. Boggs, D. Carmean, A. Kyker, and P. Roussel. The Microarchitecture of the Pentium® 4 Processor. In *Intel Technology Journal.* Citeseer, 2001.

[60] D. Hisamoto, W. Lee, J. Kedzierski, H. Takeuchi, K. Asano, C. Kuo, E. Anderson, T. King, J. Bokor, and C. Hu. FinFET - A Self-Aligned Double-Gate MOSFET Scalable To 20nm. *IEEE Transactions on Electron Devices*, 47(12):2320–2325, 2000.

[61] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, and M. B. Srivastava. Power Optimization of Variable-Voltage Core-Based Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18 (12):1702–1714, 1999.

[62] S. Hong and H. Kim. An Integrated GPU Power and Performance Model. In *ACM SIGARCH Computer Architecture News*, volume 38, pages 280–289. ACM, 2010.

[63] M. Horowitz, T. Indermaur, and R. Gonzales. Low-Power Digital Design. In *Proceedings of the Symposium on Low Power Electronics.* IEEE Computer Society, Los Alamos, CA, 1994.

[64] M. Hsieh, A. Rodrigues, R. Riesen, K. Thompson, and W. Song. A Framework for Architecture-Level Power, Area, and Thermal Simulation and its Application to Network-On-Chip Design Exploration. *ACM SIGMETRICS Performance Evaluation Review*, 38(4):63–68, 2011.

[65] C. H. Hsu, W. C. Feng, and J. S. Archuleta. Towards Efficient Supercomputing: A Quest for the Right Metric. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium*, 2005.

[66] T. Jamil. RISC versus CISC: Why Less is More. *Ieee Potentials*, 14(3): 13–16, 1995.

[67] C. L. Janssen, H. Adalsteinsson, S. Cranford, J. P. Kenny, A. Pinar, D. A. Evensky, and J. Mayo. A Simulator for Large-Scale Parallel Computer Architectures. *Technology Integration Advancements in Distributed Systems and Computing*, 179, 2012.

[68] C. January, J. Byrd, X. Oró, and M. O'Connor. Allinea MAP: Adding Energy and OpenMP Profiling Without Increasing Overhead. In *Proceedings of the 8th International Workshop on Parallel Tools for High Performance Computing*, pages 25–35, October 2014.

[69] R. Joseph and M. Martonosi. Run-Time Power Estimation in High Performance Microprocessors. In *Proceedings of the 2001 International Symposium on Low Power Electronics and Design (ISLPED)*, pages 135–140. ACM, 2001.

[70] N. P. Jouppi, P. Boyle, and J. S. Fitch. Designing, Packaging, and Testing a 300-MHz, 115W ECL Microprocessor. *IEEE Micro*, 14(2):50–58, April 1994.

[71] L. V. Kale and S. Krishnan. CHARM++: A Portable Concurrent Object Oriented System Based on C++. In *ACM Sigplan Notices*, volume 28, pages 91–108. ACM, 1993.

[72] S. Kamil, J. Shalf, and E. Strohmaier. Power Efficiency in High Performance Computing. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1–8. IEEE, 2008.

[73] S. Kaxiras and M. Martonosi. *Computer Architecture Techniques for Power-Efficiency*. Morgan and Claypool Publishers, 1st edition, 2008.

[74] S. Kaxiras and M. Martonosi. *Computer Architecture Techniques for Power-Efficiency*. Morgan and Claypool Publishers, 1st edition, 2008.

[75] C. Kessler, U. Dastgeer, S. Thibault, R. Namyst, A. Richards, U. Dolinsky, S. Benkner, J. L. Träff, and S. Pllana. Programmability and Performance Portability Aspects of Heterogeneous Multi-/Manycore Systems. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1403–1408. EDA Consortium, 2012.

[76] N. S. Kim, T. Austin, D. Baauw, T. Mudge, K. Flautner, J. S. Hu, M. J. Irwin, M. Kandemir, and V. Narayanan. Leakage Current: Moore's Law Meets Static Power. *Computer*, 36:68–75, 2003.

[77] V. Kindratenko and P. Trancoso. Trends in High-Performance Computing. *Computing in Science & Engineering*, 13(3):92–95, 2011.

[78] D. Koufaty and D. T. Marr. Hyperthreading Technology in the Netburst Microarchitecture. *IEEE Micro*, 23(2):56–65, March 2003.

[79] K. D. Lange. Identifying Shades of Green: The SPECpower Benchmarks. *Computer*, 42(3):95–97, March 2009.

[80] J. H. Laros, K. Pedretti, S. M. Kelly, Wei Shu, K. Ferreira, J. Vandyke, and C. Vaughan. Energy Delay Product. In *Energy-Efficient High Performance Computing: Measurement and Tuning*, pages 51–55. Springer, 2013.

[81] J. H. Laros, P. Pokorny, and D. DeBonis. PowerInsight - A Commodity Power Measurement Capability. In *International Green Computing Conference (IGCC '13)*, pages 1–6. IEEE, 2013.

[82] G. Lawson, V. Sundriyal, M. Sosonkina, and Y. Shen. Modeling Performance and Energy for Applications Offloaded to Intel Xeon Phi. In *Proceedings of the 2nd International Workshop on Hardware-Software Co-Design for High Performance Computing*, page 7. ACM, 2015.

[83] E. Le Sueur and G. Heiser. Dynamic Voltage and Frequency Scaling: The Laws of Diminishing Returns. In *Proceedings of the 2010 International Conference on Power Aware Computing and Systems (HotPower'10)*, pages 1–8. USENIX Association, October 2010.

[84] W. Y. Lee. Energy-Saving DVFS Scheduling of Multiple Periodic Real-Time Tasks on Multi-Core Processors. In *Proceedings of the 2009 13th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications*, pages 216–223. IEEE Computer Society, 2009.

[85] S. Li, J. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-42)*, pages 469–480, December 2009.

[86] C. Lively, X. Wu, V. Taylor, S. Moore, H. Chang, C. Su, and K. Cameron. Power-aware Predictive Models of Hybrid (MPI/OpenMP) Scientific Applications on Multicore Systems. *Computer Science-Research and Development*, 27(4):245–253, 2012.

[87] C. Malone and C. Belady. Metrics to Characterize Data Center & IT Equipment Energy Use. In *Proceedings of the 2006 Digital Power Forum*, September 2006.

[88] A. J. Martin, M. Nyström, and P. I. Pénzes. *ET^2: A Metric for Time and Energy Efficiency of Computation.* In *Power Aware Computing*, pages 293–315. Springer, 2002.

[89] C. D. Martin. ENIAC: The Press Conference That Shook the World. *IEEE Technology and Society Magazine*, 14(4):3–10, 1995.

[90] J. D. McCalpin. Memory Bandwidth and Machine Balance in Current High Performance Computers. *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, pages 19–25, December 1995.

[91] M. D. McCool, A. D. Robison, and J. Reinders. *Structured Parallel Programming: Patterns for Efficient Computation*. Elsevier, 2012.

[92] F. J. Mesa-Martinez, M. Brown, J. Nayfach-Battilana, and J. Renau. Measuring Performance, Power, and Temperature from Real Processors. In *Proceedings of the 2007 workshop on Experimental computer science*, page 16. ACM, 2007.

[93] G. E. Moore. Cramming More Components onto Integrated Circuits. *Proceedings of the IEEE*, 86(1):82–85, 1998.

[94] S. V. Moore. A Comparison of Counting and Sampling Modes of Using Performance Monitoring Hardware. In Sloot, P. M. A. and Hoekstra, A. G. and Tan, K. C. J. and Dongarra, J. J., editor, *Proceedings of the 2002 International Conference on Computational Science*, pages 904–912. Springer Berlin Heidelberg, 2002.

[95] T. Mudge. Power: A First-Class Architectural Design Constraint. *Computer*, 34(4):52–58, April 2001.

[96] M. S. Müller, A. Knüpfer, M. Jurenz, M. Lieber, H. Brunst, H. Mix, and W. E. Nagel. Developing Scalable Applications with Vampir, Vam-

pirServer and VampirTrace. In *PARCO*, volume 15, pages 637–644. Citeseer, 2007.

[97] E. L. Padoin, L. L. Pilla, M. Castro, F. Z. Boito, P. O. Navaux, and J. Méhaut. Performance/energy trade-off in scientific computing: The case of arm big.little and intel sandy bridge. *IET Computers & Digital Techniques*, 9(1):27–35, 2014.

[98] D. A. Patterson and J. L. Hennessy. *Computer Organization and Design: The Hardware/Software Interface*. Newnes, 2013.

[99] M. K. Patterson, S. W. Poole, C. Hsu, D. Maxwell, W. Tschudi, H. Coles, D. J. Martinez, and N Bates. TUE, a New Energy-Efficiency Metric Applied at ORNLs Jaguar. In *Supercomputing*, pages 372–382. Springer, 2013.

[100] S. J. Pennycook, J. Sewall, and V. Lee. Implications of a Metric for Performance Portability. *Future Generation Computer Systems (In Press)*, 2017.

[101] L. Ponciano and F. Brasileiro. On the Impact of Energy-Saving Strategies in Opportunistic Grids. In *Proceedings of the 2010 IEEE/ACM International Conference on Grid Computing (ICGC '10)*, pages 282–289, Oct 2010.

[102] T. Rauber and G. Rünger. *Parallel Programming for Multicore and Cluster Systems*. Springer Science & Business Media, 2013.

[103] S. I. Roberts, S. A. Wright, S. A. Fahmy, and S. A. Jarvis. The Power-Optimised Software Envelope. *ACM Transactions on Architecture and Code Optimisation*, in preparation.

[104] S. I. Roberts, S. A. Wright, D. Lecomber, C. January, J. Byrd, X. Oró, and S. A. Jarvis. POSE: A Mathematical and Visual Modelling Tool to Guide Energy Aware Code Optimisation. In *Proceedings of the 6th*

International Green and Sustainable Computing Conference (IGSC '15), December 2015.

[105] S. I. Roberts, S. A. Wright, S. A. Fahmy, and S. A. Jarvis. Metrics for Energy-Aware Software Optimisation. *Lecture Notes in Computer Science (LNCS)*, 10266:413–430, June 2017.

[106] J. Robertson. High Dielectric Constant Gate Oxides for Metal Oxide Si Transistors. *Reports on Progress in Physics*, 69(2):327, 2005.

[107] I. Rodero, H. Viswanathan, E. K. Lee, M. Gamell, D. Pompili, and M. Parashar. Energy-Efficient Thermal-Aware Autonomic Management of Virtualized HPC Cloud Infrastructure. *Journal of Grid Computing*, 10 (3):447–473, 2012.

[108] R. Schöne, J. Treibig, M. F. Dolz, C. Guillen, C. Navarrete, M. Knobloch, and B. Rountree. Tools and Methods for Measuring and Tuning the Energy Efficiency of HPC Systems. *Scientific Programming*, 22(4):273–283, 2014.

[109] J. Shalf, S. Dosanjh, and J. Morrison. Exascale Computing Technology Challenges. In *High Performance Computing for Computational Science (VECPAR 2010)*, volume 6449 of *LNCS*, pages 1–25. Springer Berlin Heidelberg, June 2011.

[110] H. Shan, K. Antypas, and J. Shalf. Characterizing and Predicting the I/O Performance of HPC Applications Using a Parameterized Synthetic Benchmark. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, page 42. IEEE, 2008.

[111] Y. S. Shao and D. Brooks. Energy Characterization and Instruction-Level Energy Model of Intel's Xeon Phi Processor. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED '13)*, pages 389–394. IEEE, September 2013.

[112] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-Aware Microarchitecture. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, pages 2–13. IEEE, 2003.

[113] V. Srinivasan, D. Brooks, M. Gschwind, P. Bose, V. Zyuban, P. N. Strenski, and P. G. Emma. Optimizing Pipelines for Power and Performance. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pages 333–344, 2002.

[114] B. Subramaniam, W. Saunders, T. Scogland, and W. Feng. Trends in Energy-Efficient Computing: A Perspective from the Green500. In *2013 International Green Computing Conference (IGCC '13)*, pages 1–8. IEEE, 2013.

[115] H. Sutter. The Free Lunch is Over: A Fundamental Turn Toward Concurrency in Software. *Dr. Dobb's Journal*, 30(3):202–210, March 2005.

[116] M. M. Waldrop. The Chips are Down for Moore's Law. *Nature*, 530(7589): 144–147, 2016.

[117] V. M. Weaver. Advanced Hardware Profiling and Sampling (PEBS, IBS, etc.): Creating a New PAPI Sampling Interface. Technical report, Department of Electrical and Computer Engineering, University of Maine, 2016.

[118] V. M. Weaver, M. Johnson, K. Kasichayanula, J. Ralph, P. Luszczek, D. Terpstra, and S. Moore. Measuring Energy and Power with PAPI. In *International Conference on Parallel Processing Workshops (ICPPW '12)*, pages 262–268. IEEE, 2012.

[119] S. Wienke, P. Springer, C. Terboven, and D. Mey. OpenACC First Experiences with Real-World Applications. In *European Conference on Parallel Processing*, pages 859–870. Springer, 2012.

[120] B. Wilkinson and M. Allen. *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2004. ISBN 0131405632.

[121] S. Williams, A. Waterman, and D. Patterson. Roofline: An Insightful Visual Performance Model for Multicore Architectures. *Communications of the ACM*, 52(4):65–76, April 2009. ISSN 0001-0782.

[122] S. Yeo and H. Lee. Using Mathematical Modeling in Provisioning a Heterogeneous Cloud Computing Environment. *Computer*, 44(8):55–62, 2011.

[123] Y. Zhu and V. J. Reddi. High Performance and Energy Efficient Mobile Web Browsing on big.LITTLE Systems. In *2013 International Symposium on High Performance Computer Architecture (HPCA '13)*, pages 13–24. IEEE, 2013.

APPENDIX A

POSE Model Summary for Different Metrics

This appendix summarises the equations derived for the various Power Optimised Software Envelope (POSE) bounds and coordinates in Chapter 5.

A.1 Et^n POSE

$$E_\lambda = E_\theta \left(\frac{t_\theta}{t_\lambda}\right)^n \tag{Optimisation}$$

$$E_\lambda = P_\theta \, t_\lambda \left(\frac{t_\lambda}{t_\theta}\right)^{n+1} \tag{Contribution}$$

$$t_A = t_\theta \left(\frac{{P_{min}}^2}{P_\theta \, P_{max}}\right)^{\frac{1}{n+1}} \qquad E_A = P_{max} \cdot t_A \tag{A}$$

$$t_B = t_\theta \left(\frac{P_\theta}{P_{max}}\right)^{\frac{1}{n+1}} \qquad E_B = P_{max} \cdot t_B \tag{B}$$

$$t_C = t_\theta \left(\frac{P_{min}}{P_\theta}\right)^{\frac{1}{n+1}} \qquad E_C = P_{min} \cdot t_C \tag{C}$$

$$t_D = t_\theta \qquad E_D = P_{min} \cdot t_D \tag{D}$$

$$t_E = t_\theta \left(\frac{P_\theta}{P_{min}}\right)^{\frac{1}{n+1}} \qquad E_E = P_{min} \cdot t_E \tag{E}$$

A.2 Energy Delay Sum POSE

$$E_\lambda = E_\theta + \frac{\beta}{\alpha}\left(t_\theta - t_\lambda\right) \tag{Optimisation}$$

$$E_\lambda = \frac{{t_\lambda}^2}{t_\theta}\left(P_\theta + \frac{\beta}{\alpha}\right) - t_\lambda \, \frac{\beta}{\alpha} \tag{Contribution}$$

$$t_A = t_\theta \, \frac{\left(P_{min} + \frac{\beta}{\alpha}\right)^2}{\left(P_\theta + \frac{\beta}{\alpha}\right)\left(P_{max} + \frac{\beta}{\alpha}\right)} \qquad E_A = P_{max} \cdot t_A \tag{A}$$

$$t_B = t_\theta \frac{P_\theta + \frac{\beta}{\alpha}}{P_{max} + \frac{\beta}{\alpha}} \qquad E_B = P_{max} \cdot t_B \tag{B}$$

$$t_C = t_\theta \frac{P_{min} + \frac{\beta}{\alpha}}{P_\theta + \frac{\beta}{\alpha}} \qquad E_C = P_{min} \cdot t_C \tag{C}$$

$$t_D = t_\theta \qquad E_D = P_{min} \cdot t_D \tag{D}$$

$$t_E = t_\theta \frac{P_\theta + \frac{\beta}{\alpha}}{P_{min} + \frac{\beta}{\alpha}} \qquad E_E = P_{min} \cdot t_E \tag{E}$$

A.3 Energy Delay Distance POSE

$$E_\lambda = \sqrt{{E_\theta}^2 + \left(\frac{\beta}{\alpha}\right)^2 \left({t_\theta}^2 - {t_\lambda}^2\right)} \tag{Optimisation}$$

$$E_\lambda = t_\lambda \cdot \sqrt{\left(\frac{t_\lambda}{t_\theta}\right)^2 \left({P_\theta}^2 + \left(\frac{\beta}{\alpha}\right)^2\right) - \left(\frac{\beta}{\alpha}\right)^2} \tag{Contribution}$$

$$t_A = t_\theta \cdot \frac{{P_{min}}^2 + \left(\frac{\beta}{\alpha}\right)^2}{\sqrt{{P_\theta}^2 + \left(\frac{\beta}{\alpha}\right)^2} \cdot \sqrt{{P_{max}}^2 + \left(\frac{\beta}{\alpha}\right)^2}} \qquad E_A = P_{max} \cdot t_A \tag{A}$$

$$t_B = t_\theta \cdot \sqrt{\frac{{P_\theta}^2 + \left(\frac{\beta}{\alpha}\right)^2}{{P_{max}}^2 + \left(\frac{\beta}{\alpha}\right)^2}} \qquad E_B = P_{max} \cdot t_B \tag{B}$$

$$t_C = t_\theta \cdot \sqrt{\frac{{P_{min}}^2 + \left(\frac{\beta}{\alpha}\right)^2}{{P_\theta}^2 + \left(\frac{\beta}{\alpha}\right)^2}} \qquad E_C = P_{min} \cdot t_C \tag{C}$$

$$t_D = t_\theta \qquad E_D = P_{min} \cdot t_D \tag{D}$$

$$t_E = t_\theta \cdot \sqrt{\frac{{P_\theta}^2 + \left(\frac{\beta}{\alpha}\right)^2}{{P_{min}}^2 + \left(\frac{\beta}{\alpha}\right)^2}} \qquad E_E = P_{min} \cdot t_E \tag{E}$$

APPENDIX B
Mantevo Suite POSE Models

Table B.1 summarises the results of Et^2 POSE models for the remaining codes documented in Chapter 5. Tables B.2 and B.3 present similar results for the Energy Delay Sum (EDS) and Energy Delay Distance (EDD) metrics respectively. These results are based on the same metric parameterisations used in chapter 4. These tables of results were generated automatically using open source tools produced as part of this work.

Table B.1: Et^2 POSE Model Summaries for Remaining Codes

CloverLeaf 3D	
Maximum Energy Saved by Reduced Power Consumption	11 960 J; 1.38×
Maximum Improvement in E^1t^2 from Power Optimisation	1.89×
Worst Case Slowdown as a result of Power Optimisation	17.19 s; 1.11×
Minimum Speed-up Guaranteed to Outperform θ	9.36 s; 1.07×
Speed-up Required to Dominate Power Optimisation	36.92 s; 1.32×
CloverLeaf	
Maximum Energy Saved by Reduced Power Consumption	13 351 J; 1.30×
Maximum Improvement in E^1t^2 from Power Optimisation	1.69×
Worst Case Slowdown as a result of Power Optimisation	19.58 s; 1.09×
Minimum Speed-up Guaranteed to Outperform θ	16.89 s; 1.09×
Speed-up Required to Dominate Power Optimisation	48.56 s; 1.29×
CoMD	
Maximum Energy Saved by Reduced Power Consumption	2916 J; 1.13×
Maximum Improvement in E^1t^2 from Power Optimisation	1.28×
Worst Case Slowdown as a result of Power Optimisation	4.49 s; 1.04×
Minimum Speed-up Guaranteed to Outperform θ	12.67 s; 1.14×
Speed-up Required to Dominate Power Optimisation	20.09 s; 1.24×
MiniFE	
Maximum Energy Saved by Reduced Power Consumption	843 J; 1.11×
Maximum Improvement in E^1t^2 from Power Optimisation	1.23×
Worst Case Slowdown as a result of Power Optimisation	1.31 s; 1.04×
Minimum Speed-up Guaranteed to Outperform θ	4.62 s; 1.14×
Speed-up Required to Dominate Power Optimisation	6.79 s; 1.23×
HPCCG	
Maximum Energy Saved by Reduced Power Consumption	469 J; 1.06×
Maximum Improvement in E^1t^2 from Power Optimisation	1.13×
Worst Case Slowdown as a result of Power Optimisation	0.74 s; 1.02×
Minimum Speed-up Guaranteed to Outperform θ	5.08 s; 1.16×
Speed-up Required to Dominate Power Optimisation	6.31 s; 1.21×

Table B.2: EDS POSE Model Summaries ($\alpha = 1, \beta = 900$)

TeaLeaf	
Maximum Energy Saved by Reduced Power Consumption	32 560 J; 1.48×
Maximum Improvement in E^1t^2 from Power Optimisation	1.19×
Worst Case Slowdown as a result of Power Optimisation	28.40 s; 1.09×
Minimum Speed-up Guaranteed to Outperform θ	9.65 s; 1.03×
Speed-up Required to Dominate Power Optimisation	59.76 s; 1.23×
CloverLeaf 3D	
Maximum Energy Saved by Reduced Power Consumption	11 960 J; 1.38×
Maximum Improvement in E^1t^2 from Power Optimisation	1.15×
Worst Case Slowdown as a result of Power Optimisation	10.80 s; 1.07×
Minimum Speed-up Guaranteed to Outperform θ	7.32 s; 1.05×
Speed-up Required to Dominate Power Optimisation	25.89 s; 1.20×
CloverLeaf	
Maximum Energy Saved by Reduced Power Consumption	13 351 J; 1.30×
Maximum Improvement in E^1t^2 from Power Optimisation	1.12×
Worst Case Slowdown as a result of Power Optimisation	12.05 s; 1.06×
Minimum Speed-up Guaranteed to Outperform θ	12.97 s; 1.06×
Speed-up Required to Dominate Power Optimisation	33.84 s; 1.19×
CoMD	
Maximum Energy Saved by Reduced Power Consumption	2916 J; 1.13×
Maximum Improvement in E^1t^2 from Power Optimisation	1.05×
Worst Case Slowdown as a result of Power Optimisation	2.63 s; 1.02×
Minimum Speed-up Guaranteed to Outperform θ	9.33 s; 1.10×
Speed-up Required to Dominate Power Optimisation	13.95 s; 1.15×
MiniFE	
Maximum Energy Saved by Reduced Power Consumption	843 J; 1.11×
Maximum Improvement in E^1t^2 from Power Optimisation	1.04×
Worst Case Slowdown as a result of Power Optimisation	0.76 s; 1.02×
Minimum Speed-up Guaranteed to Outperform θ	3.38 s; 1.10×
Speed-up Required to Dominate Power Optimisation	4.72 s; 1.15×
HPCCG	
Maximum Energy Saved by Reduced Power Consumption	469 J; 1.06×
Maximum Improvement in E^1t^2 from Power Optimisation	1.02×
Worst Case Slowdown as a result of Power Optimisation	0.42 s; 1.01×
Minimum Speed-up Guaranteed to Outperform θ	3.66 s; 1.11×
Speed-up Required to Dominate Power Optimisation	4.41 s; 1.14×
PathFinder	
Maximum Energy Saved by Reduced Power Consumption	1928 J; 1.03×
Maximum Improvement in E^1t^2 from Power Optimisation	1.01×
Worst Case Slowdown as a result of Power Optimisation	1.74 s; 1.01×
Minimum Speed-up Guaranteed to Outperform θ	35.71 s; 1.12×
Speed-up Required to Dominate Power Optimisation	38.81 s; 1.13×

Table B.3: EDD POSE Model Summaries ($\alpha = 1, \beta = 519.615$)

TeaLeaf	
Maximum Energy Saved by Reduced Power Consumption	32 560 J; 1.48×
Maximum Improvement in E^1t^2 from Power Optimisation	1.17×
Worst Case Slowdown as a result of Power Optimisation	25.80 s; 1.08×
Minimum Speed-up Guaranteed to Outperform θ	10.248 s; 1.03×
Speed-up Required to Dominate Power Optimisation	54.81 s; 1.20×
CloverLeaf 3D	
Maximum Energy Saved by Reduced Power Consumption	11 960 J; 1.38×
Maximum Improvement in E^1t^2 from Power Optimisation	1.12×
Worst Case Slowdown as a result of Power Optimisation	9.15 s; 1.06×
Minimum Speed-up Guaranteed to Outperform θ	7.576 s; 1.05×
Speed-up Required to Dominate Power Optimisation	23.53 s; 1.18×
CloverLeaf	
Maximum Energy Saved by Reduced Power Consumption	13 351 J; 1.30×
Maximum Improvement in E^1t^2 from Power Optimisation	1.10×
Worst Case Slowdown as a result of Power Optimisation	9.96 s; 1.05×
Minimum Speed-up Guaranteed to Outperform θ	13.17 s; 1.07×
Speed-up Required to Dominate Power Optimisation	30.65 s; 1.17×
CoMD	
Maximum Energy Saved by Reduced Power Consumption	2916 J; 1.13×
Maximum Improvement in E^1t^2 from Power Optimisation	1.04×
Worst Case Slowdown as a result of Power Optimisation	2.04 s; 1.02×
Minimum Speed-up Guaranteed to Outperform θ	9.05 s; 1.09×
Speed-up Required to Dominate Power Optimisation	12.68 s; 1.14×
MiniFE	
Maximum Energy Saved by Reduced Power Consumption	843 J; 1.11×
Maximum Improvement in E^1t^2 from Power Optimisation	1.03×
Worst Case Slowdown as a result of Power Optimisation	0.59 s; 1.02×
Minimum Speed-up Guaranteed to Outperform θ	3.30 s; 1.10×
Speed-up Required to Dominate Power Optimisation	4.30 s; 1.13×
HPCCG	
Maximum Energy Saved by Reduced Power Consumption	469 J; 1.06×
Maximum Improvement in E^1t^2 from Power Optimisation	1.02×
Worst Case Slowdown as a result of Power Optimisation	0.32 s; 1.01×
Minimum Speed-up Guaranteed to Outperform θ	3.48 s; 1.11×
Speed-up Required to Dominate Power Optimisation	4.05 s; 1.12×
PathFinder	
Maximum Energy Saved by Reduced Power Consumption	1928 J; 1.03×
Maximum Improvement in E^1t^2 from Power Optimisation	1.01×
Worst Case Slowdown as a result of Power Optimisation	1.29 s; 1.01×
Minimum Speed-up Guaranteed to Outperform θ	33.60 s; 1.11×
Speed-up Required to Dominate Power Optimisation	35.91 s; 1.12×

APPENDIX C
Mantevo Benchmark Input Parameters

Table C.1 lists the parameters used to run applications from the Mantevo suite. Applications were run with the default configuration given by their documentation where available. In some cases parameters corresponding to problem size were altered to produce realistic run times and memory consumption.

Table C.1: Application Run Parameters

Application	Parameters
TeaLeaf	tea_bm16_short.in
CloverLeaf 3D	clover_bm.in
CloverLeaf	clover_bm2.in
MiniMD	-t 24 -n 15000
CoMD	-e -x 90 -y 90 -z 90
MiniFE	-nx 256 -ny 256 -nz 256
HPCCG	256 256 256
PathFinder	-x medium_test.adj_list