Energy modelling of multi-threaded, multi-core software for embedded systems

Steven P. Kerrison

A thesis submitted to the University of Bristol in accordance with the requirements of the degree Doctor of Philosophy in the Faculty of Engineering, Department of Computer Science, September 2015.

51,000 words.

Abstract

Efforts to reduce energy consumption are being made across all disciplines. ICT's contribution to global energy consumption and by-products such as CO2 emissions continues to grow, making it an increasingly significant area in which improvements must be made. This thesis focuses on software as a means to reducing energy consumption. It presents methods for profiling and modelling a multi-threaded, multi-core embedded processor at the instruction set level, establishing links between the software and the energy consumed by the underlying hardware.

A framework is presented that profiles the energy consumption characteristics of a multi-threaded processor core, associating energy consumption with the instruction set and parallelism present in a multi-threaded program. This profiling data is used to build a model of the processor that allows instruction set simulations to be used to estimate the energy that programs will consume, with an average of 2.67 % error.

The profiling and modelling is then raised to the multi-core level, examining a channel based message passing system formed of a network of embedded multi-threaded processors. Additional profiling is presented that determines network communication costs as well as giving consideration towards system level properties such as power supply efficiency. Then, this is used to form a system level energy model that can estimate consumption using simulations of multi-core programs. The system level model combines multiple instances of a core energy model with a network level communication cost model.

The broader implications of this work are explored in the context of other embedded and multi-core processor architectures, identifying opportunities for expanding or transferring the models. The models in this thesis are formed at the instruction set level, but have been demonstrated to be effective at higher-levels of abstraction than instruction set simulation, through their support of further work carried out externally.

This work is enabled by several pieces of development effort, including a profiling framework for taking power measurements of the devices under investigation, tools for programming, routing and debugging software on a multi-core hardware platform called Swallow, and enhancements to an instruction set simulator for the simulation of this multi-core system.

Through the work of this thesis, an embedded software developer for multi-threaded and multi-core systems is equipped with tools, techniques and new understanding that can help them in determining how their software consumes energy. This raises the status of energy efficiency in the software development cycle as we continue our efforts to minimise the energy impact of the world's embedded devices.

Acknowledgements

I owe the successful completion of this PhD thesis to a great many people, and I can directly thank but a few of them here. If I interacted with you during the course of my research, please know that I am eternally grateful for that. Thank you to my family for supporting and encouraging me throughout.

Thank you to my supervisor, Kerstin Eder, whose guidance helped me develop a compelling research topic and secure funding for my work, without which none of this would have been possible. Thank you to Simon Hollis and Jake Longo Galea for creating a rather interesting set of research problems for us to collectively solve in the Swallow platform. David May, your thought provoking discussions have been invaluable and inspiring. Many thanks to my external examiners, Alex Yakovlev and Peter Marwedel, as well as my internal coordinator José Luis Núñez-Yáñez.

My colleagues and companions in research deserve much gratitude for their input, collaboration, and of course their tolerance. Jamie Hanlon, Roger Owens, Neville Grech, Kyriakos Georgiou, Jeremy Morse and James Pallister, you and many others in the department made research an exciting experience. A special thank you to Dejanira Araiza Illan for your support, particularly during the write-up.

In the first year of my studies I was hosted by XMOS. This was an excellent place to form ideas, gain industrial insight and motivate my work. In particular, thanks to Henk Muller, John Ferguson, Matt Fyles and Richard Osborne for their expert advice and support.

My work was funded by a University of Bristol PhD Scholarship, and much of it became relevant to the ENTRA EU FP7 FET research project. I am grateful to these funding sources for making this work possible, and for creating an ecosystem in which to disseminate and further explore this work.

Author's declaration

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Research Degree Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, the work is the candidate's own work. Work done in collaboration with, or with the assistance of, others, is indicated as such. Any views expressed in the dissertation are those of the author.

Signed: _____

Date: _____

Contents

List of Figures

List of Tables

List of Code Listings

1. Introduction

The goal of saving energy is considered a contemporary challenge, motivated by several factors, but dominated by two: managing the world's consumption of resources and limiting the rate at which we produce harmful by-products of that consumption, such as carbon dioxide. In computing, however, it is not necessarily a contemporary challenge, nor do those two factors alone form the primary goals.

Energy has always governed the uses for and effectiveness of computers. Mechanical computers were large and slow, whilst the adoption of vacuum tubes offered higher performance. The transistor and its subsequent miniaturisation to nanometer scale allowed computers to increase in speed, reduce in size, and consume a small enough amount of energy to be pervasive devices in offices, homes and vehicles.

While the practicalities of energy consumption in computing have been a governing factor for nearly a century, the motivation to reduce processor energy continues, as the Internet of Things (IoT) — an ever-growing number of interconnected embedded devices — creeps into the technological lexicon. These devices must be small and consume tiny amounts of energy, often powered by minute batteries or via energy harvesting.

Using energy consumption data from studies into data centers, PCs, network hardware and other Information and Communication Technology (ICT) equipment, ICTs energy consumption was determined to be 8 % of global consumption in 2008 [Pic+08]. Therefore, progress towards both environmental and product-centric goals can be made by continuing to reduce the energy consumption of devices.

As we reach technological limits, new techniques must be created to allow progress. For decades we have relied, and continue to rely upon Moore's Law [Moo65] and trends related to it. The shrinking of transistors and improvements in process technology yield energy efficiency improvements, but now more aggressive energy saving techniques are devised and applied at higher levels, from circuitry to turn off temporarily unused silicon, up to software controlled sleep states. The advent of multi-core, which was necessary to avoid the practical limits of operating frequency and power, introduces new opportunities but also new challenges, particularly in the areas of task scheduling and effective programming models.

As the aggressiveness of energy saving techniques increases, the software that runs on top of the processor eventually becomes a point of interest. This software is ultimately responsible for the behaviour of the hardware — the hardware exists to perform the tasks defined in software by the authors of that software. The software, therefore, is largely responsible for a device's energy consumption. To re-state the argument from a bottom-up perspective, a device with many energy-saving features is inefficient if the software running on it prevents those features from being used, or fails to adequately exploit them.

An abundance of evidence towards this is present in mobile phones, where devices must be designed to be energy efficient. However, a large number of software *energy bugs* have been observed at all levels of the software stack [PHZ11]. These software problems amount to 35 % of the energy bugs surveyed. Typically, these bugs prevent the hardware from entering low power states. Energy bugs have several negative impacts, including poor reviews for buggy applications, reports of phones with poor battery life and even increased product returns.

In order to write energy efficient software, developers must understand the energy that their code will consume. To that end, this thesis proposes new techniques for addressing the imbalance between understanding of hardware energy consumption and how the software running upon that hardware affects it.

The focus of this work is on multi-threaded and multi-core processors in the embedded system space, where the processor contributes a significant proportion of system energy consumption. This is evident if we consider a particular device class: the mobile phone. The most significant energy consumption within these devices is a combination of back-light, display, radio, graphics

and processor [CH10]. If we consider that in a more deeply embedded system, such as one not interacting directly with humans, then the display, along with back-light and graphics processor, are no longer present. Thus, the processor's energy consumption becomes dominant. Further, in such systems, energy is often in scarce supply, either due to the delivery mechanism or storage method, for example a battery of limited capacity. It is desirable to maximize energy efficiency in order to reduce the complexity of providing sufficient energy to these devices. The goal of this work is to propose new methods for identifying how software consumes energy in such systems, supporting these proposals with experimental tools, energy models, along with testing and evaluation. These contributions can then be used as the basis for future work.

The research herein includes an in-depth study of a multi-threaded processor, assembled into multi-cores. The hardware's energy consumption, and its relationship to the software running upon it, is analysed at multiple levels, starting at the instruction set and progressing to a system level considering multiple networked cores. Through this analysis, this thesis is able to present an energy model for a multi-threaded embedded processor architecture and raise that modelling up to the multi-core level. It is shown that a combination of understanding the target hardware and writing software that fits the hardware well is essential for energy efficiency.

Software is selected to demonstrate behaviours typical of an embedded system, including multi-threaded and multi-core examples. This software is compiled and the executables are then energy modelled using simulation at the instruction set level. The presented core-level multi-threaded energy model delivers accuracy within 10 % of measured hardware energy consumption and 2.67 % on average, with a standard deviation of 4.40 percentage points. At the network level, absolute energy estimations diverge from the hardware. However, the energy implications of communicating tasks are made clear through the reporting and visualisation methods that are presented. Most importantly, the relative improvements (or otherwise) from changes to the software can be observed without the detailed hardware modelling used in processor design, and without needing to instrument the target hardware. This makes energy modelling more accessible to the software developer.

Finally, this work enables higher level analyses, such as static analysis, to be performed, by feeding the model data into them. Thus, this research aims to provide enlightenment to software engineers with an interest in the energy consumption of their embedded software, and to other researchers seeking new methods to provide and act upon this information through reporting and optimisation.

The rest of this introductory chapter formally defines the research questions posed in this work, summarises the contributions of this thesis along with related publications, outlines the structure of the document and states the terminology and conventions used within.

1.1. Research questions and thesis

At its core, this thesis seeks to further the state of the art in energy modelling of software. It does so by focusing at the embedded device level, observing emerging changes in how devices are constructed and used across ICT. The fundamental question that lies beneath this work can be posed from the perspective of an embedded systems software developer:

How much energy will the software that I am writing consume?

Without sufficient hardware knowledge, there is very little intuition when seeking the answer to this question. Yet, in embedded systems, energy consumption is critical to the safe and correct operation of a device. If this question can be answered, then the software developer can make educated decisions about what action to take, be it make changes to their software, modify the system hardware, or re-visit the specification.

This question is quite a broad one, which when asked by an embedded software developer, indicates a specific goal: to minimise energy consumption in order to provide optimal functionality of the embedded device, without breaking any of the constraints essential to its correct operation. This can be phrased as a more specific question:

Software that is a good fit to the underlying hardware is more energy efficient, but how can I achieve this?

Whilst abstraction allows a developer to avoid concerning themselves with the engineering beneath the level at which they want to work, understanding how higher-level implementations map down to low-level activity is fundamentally important, both in terms of performance and energy. Regardless of energy saving features in the hardware, a piece of software that neither directly exploits the best features of the hardware, nor passively allows the features to work, will lead to sub-optimal power [RJ97]. This is true historically and continues to be true today, and methods for allowing this mapping to take place must continue to be developed if energy consumption of software is to be better understood on contemporary hardware. Understanding this research question also provides insight into what software is not a good fit to a particular system.

This thesis contributes new answers to these research questions. The statements underpinning the work of this thesis are as follows:

Effective energy estimates for modern embedded software must consider multi-threaded, multi-core systems. Parallelism in hardware is now necessary as a means to deliver increases in performance. This requires multi-threading and multi-core hardware, and by extension software that maps onto this type of system.

Energy modelling at the instruction set level provides good insight into the physical behaviour of a system whilst preserving sufficient information about the software. To be useful to a software developer, an energy model must be expressible in a way that relates to both the software and the underlying hardware, exposing reasons for the behaviours that are seen.

Energy saving and energy modelling techniques are placed under greater constraints in the embedded space. In an embedded system with hard real-time constraints, software or hardware changes that may save energy cannot risk breaking those constraints. Similarly, the available hardware resources, such as performance counters, may make it difficult to collect data to aid energy modelling, either online or offline. This necessitates a modelling strategy that accounts for these limitations or is unaffected by them.

Multi-threaded and multi-core devices introduce new characteristics that must be considered in energy models. Embedded processors often have simpler pipelines than more general purpose counterparts, but the introduction of multi-threading and multi-core systems into the embedded space creates characteristics to consider. These characteristics can be unique to embedded systems, which address the need for more performance in different ways to larger processors, to enable them to satisfy the constraints placed on real-time systems. Further, the objective in such systems is to satisfy an energy budget that is often defined by a limited source of energy, such as a battery. This is in contrast to a high performance processor, which is more limited by heat dissipation and power delivery.

Energy models that do not rely on run-time data from the processor provide greater flexibility for multi-level analysis. Prior research has shown a variety of methods for estimating the energy consumption of software, some of which utilise real-time data from the processor. Such methods preclude higher level analysis, whereas this thesis presents methods that can be used across several levels of abstraction, from instruction set simulation up to abstract network level views.

Both absolute accuracy and relative indicators provide useful information to a developer. Where energy consumption constraints can be specified and are hard targets, an energy model must provide sufficient accuracy to give the developer confidence that they have or have not met that target. Using the performance of a range of prior research as a baseline, this accuracy threshold will be established as $\pm 10\,\%$. Without this confidence, the development cycle is lengthened

by the need to repeatedly deploy and test on real hardware, which may be significantly more inconvenient then running a simulation or other analysis. However, where an energy target is not absolute, or a higher level view and understanding are required, relative measures remain appropriate, for example to answer the question "which version of this software uses less energy?" Given the current lack of intuition towards software's contribution to energy consumption, this is still a valuable contribution to a developer's knowledge. What is important in such cases, however, is that a sufficiently wide view of the system is given, so that an apparent improvement in one area is not eclipsed by a side-effect created in another.

Movement of data costs energy, no matter the form that movement takes. The embedded processors studied in this work do not feature caches, nor do they use shared memory to communicate between threads. Thus, the significant energy consumption arising from cache misses and the memory hierarchy is not present. However, data must still be moved between threads via other means, and a synchronisation or other flow coordination effort between threads must take place. The cost of this must still be analysed and presented to the developer, in order to assist them in reducing energy. A network-level view of communicating threads presents a different paradigm for identifying how communication takes place and how improvements can be made, departing from the often complex behaviours of large memory hierarchies that can be difficult to reason about.

Energy models for different architectures can have elements in common. Parallelism is being provided in modern processor architecture in various ways, as challenges such as distributing data across or sharing data between cores seek to be addressed. Although this creates variety in how different processors behave and need to be programmed, an instruction set level energy model can include at least some transferable properties between different architectures. This serves to ensure the energy models can be developed for new architectures more rapidly.

From these statements, many questions can be raised that guide the research. The structure of this document follows these thesis statements closely, posing and investigating these questions progressively. An explanation of this document's structure is given in § 1.3.

1.2. Contributions

This thesis makes contributions to research in the areas of energy modelling of software, computer architecture and embedded systems. The main contributions and related publications are outlined in this section.

Energy modelling a novel embedded processor architecture

The XMOS XS1 processor architecture has a number of novel aspects to it, relating to software-defined real-time Input/Output (I/O), hardware thread scheduling, parallelism in embedded processors and multi-core networks of message-passing processors. This thesis furthers the understanding of these architectural features in relation to energy consumption at the software level, defining the particular influences that software has upon this hardware.

Contributing to the creation of a multi-core research platform

The Swallow project [Hol12] was created by Simon Hollis at the University of Bristol with the intention of building a real multi-core embedded system for demonstration and experimentation, where previously a significant amount of research was based purely upon modelled or theoretical systems. The Swallow platform forms an essential part of the research conducted in this thesis, specifically in studying and modelling multi-core communication costs.

The research conducted in this thesis has resulted in a number of significant contributions to the Swallow project, namely:

- Initial bring-up and testing of the Swallow hardware, post-manufacture.

- The introduction of wrapper scripts and pre-processing for the XMOS compiler tool-chain to provide support for the large number of processors, not previously handled by the compiler.
- Development and testing of the platform description files (XN files [XMO13a]), including mapping Joint Test Action Group (JTAG) device IDs to XMOS network node IDs and implementation of the deadlock-free dimension-order routing algorithm on Swallow's unique topology.
- Code to boot Swallow devices over their network links rather than JTAG, significantly reducing start-up time for large grids from over a minute to less than ten seconds.
- An Ethernet software stack to allow both Ethernet based Trivial File Transfer Protocol (TFTP) booting and communication with running applications.
- Communication libraries to provide more flexible channel communication than what is built into the XC language.
- A significant amount of hardware surgery involving a soldering iron, microscope and scalpel.

These contributes enabled the multi-core energy data that is presented in this thesis to be collected, and has assisted in the enablement of research by others using Swallow.

Energy modelling of a network of embedded processors

This thesis traverses various levels of system abstraction, from Instruction Set Architecture (ISA) up to system level. At the system level, a Multi-Threaded and Multi-Core (MTMC) is viewed as a network of interconnected components. These components can be independently energy modelled, as well as the interconnects between them.

The core level energy model is combined with this relatively abstract network level view and a multi-core simulation, to provide energy modelling of embedded software with a unique level of detail given to *where* the most significant quantities of energy are consumed. This serves to provide better information into how software consumes energy in modern embedded systems, so that *informed* decisions can be made to reduce that energy consumption, rather than through undirected experimentation.

Related publications

The following publications are, at the time of writing, work directly related to this thesis. For each publication, a brief description of the relationship to the thesis is given.

- Steve Kerrison and Kerstin Eder. "Energy modelling of software for a hardware multi-threaded embedded microprocessor". In: *Transactions on Embedded Computer Systems (TECS)* (2015) [KE15b]

  This journal paper describes the initial energy profiling phase and preliminary model that was produced for a sub-set of the XMOS XS1 ISA. This thesis contains that same work, described in more detail, and then built upon to produce a refined model for full ISA.

- Umer Liqat, Steven Kerrison, Serrano Alejandro, Kyriakos Giorgiou, Pedro Lopez-Garcia, Neville Grech, Manuel V. Hermenegildo, and Kerstin Eder. "Energy Consumption Analysis of Programs based on XMOS ISA-Level Models". In: *23rd International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR'13)*. Springer, Sept. 2015 [Liq+15]

  The model described in [KE15b] is used in this paper as the basis for providing energy consumption predictions through static analysis of the software. The author of this thesis contributed a description of the model to the paper, along with simulation based energy estimation results, for comparison with the static analysis method.

- Steve Kerrison and Kerstin Eder. "Measuring and modelling the energy consumption of multi-threaded, multi-core embedded software". In: *ICT Energy Letters* (July 2014), pp. 18–

19. URL: http://www.nanoenergyletters.com/files/nel/ICT-Energy_Letters_8.pdf [KE14]

This letter summarises work on further development of the model in [KE15b], along with preliminary results into the impact of multi-processor communication costs. The Swallow project, which is also described in this thesis, is an essential part of this work.

- Steve Kerrison and Kerstin Eder. "A software controlled voltage tuning system using multi-purpose ring oscillators". In: *arXiv* (2015). arXiv: 1503.05733. URL: https://arxiv.org/abs/1503.05733 [KE15a]

  The ring oscillators onboard the XMOS XS1-L are used in this work to calibrate an optimised safe (faultless) core voltage for a given operating frequency. Components of this work, particularly the background, are used in § 4.2.3.

- Simon J. Hollis and Steve Kerrison. "Overview of Swallow — A Scalable 480-core System for Investigating the Performance and Energy Efficiency of Many-core Applications and Operating Systems". In: *arXiv* (2015) [HK15]

  This overview of the Swallow system describes the salient parts of its construction, such as the routing, performance, and energy consumption. This thesis and the work surrounding it has contributed to the figures and information presented in the paper.

- Neville Grech, Kyriakos Georgiou, James Pallister, Steve Kerrison, Jeremy Morse, and Kerstin Eder. "Static analysis of energy consumption for LLVM IR programs". In: *Proceedings of the 18th International Workshop on Software and Compilers for Embedded Systems*. SCOPES '15. Sankt Goar, Germany: ACM, 2015. DOI: 10.1145/2764967.2764974 [Gre+15]

  The energy models from this thesis and [KE15b] are leveraged in this paper to perform static analysis at the LLVM IR level — the intermediate representation used in the LLVM compiler toolchain. This provides potentially richer program information than at the ISA level, preserving more control flow and other data, assisting the analysis process. For the XMOS XS1 model, this work was enabled by a mapping between the instructions used in the ISA level model and sequences of LLVM IR instructions. The author of this thesis contributed the XMOS ISA model data, as well as hardware and simulation based energy results. The static analysis and mappings between LLVM IR and ISA were contributed by the other authors of the paper.

1.3. Structure

This document is structured to follow the arguments that form the thesis described in § 1.1. Each of the thesis statements builds upon the research conducted in response to the points before it. To effectively communicate this research this work is divided into two main parts, each comprising several chapters.

Part I addresses prior work and essential background. Parallelism is explored in Chapter 2, drawing attention to the topic from both a software and hardware perspective. A variety of energy modelling methods are then detailed in Chapter 3, including discussion of the challenges that parallelism introduces to the energy modelling process.

Chapter 4 then draws upon the previous two chapters to address the properties of modern embedded systems that present further challenges to energy modelling of software. Part I is concluded with Chapter 5, which examines the XMOS XS1-L processor core and a system of these processors assembled into a grid style network; the Swallow project. The unique properties of the processor and Swallow are discussed, in relation to the topics presented in the previous chapters. This lays out the key challenges that guide the implementation decisions of this thesis.

Part II focuses on implementation, using the previously established background work, combined with new research, to address the statements made in § 1.1. It begins with two chapters that focus on a single XS1-L multi-threaded processor core. Chapter 6 presents methods for relating the energy consumption of the XS1-L to its ISA, through a newly developed profiling rig, comprising

both hardware and software. The profiling demonstrates a number of the properties of energy consumption that are unique to this particular multi-threaded embedded processor. Chapter 7 then uses this profiling data to construct an ISA level model that can be used at various levels of abstraction, starting with instruction set simulation. Several variations of the model are presented and evaluated in order to determine the best possible model accuracy.

The subsequent two chapters are structured in a similar fashion, presenting the profiling techniques and simulation tools used for the multi-core Swallow system in Chapter 8, then the model and evaluation in Chapter 9. This completes the contribution of this thesis towards a multi-threaded, multi-core, network-level energy model for an embedded real time processor.

A broader view is applied in Chapter 10, which looks beyond the XS1 processor to identify how the contributions made in this work could be applied to other architectures. Several architectures are surveyed, indicating where common characteristics may be present, and where novel features may require new research in order to further the state of the art in energy modelling of software.

Finally, the thesis is concluded in Chapter 11. The chapter contains a review of the contributions made, a summary of all evaluations made throughout the work, and a description of future work opportunities that have either been discovered during this research, or created as a result of it.

1.4. Terminology and conventions

A small summary of critical terminology and chosen conventions are described herein. Other terms are defined as necessary throughout the document. Acronyms are expanded upon the first instance of their use and also in the List of Acronyms (LoA).

Power and energy

In this thesis the terms power and energy are used frequently. These terms are often interchanged in literature, but in the context of this work it would not be appropriate. For clarity, therefore, their definitions are given.

Power, P, or *power dissipation*, is an instantaneous measure of a *rate of energy transfer*, or the rate at which work is done. It is quantified in Watts, or W. Energy, E, or *energy consumption*, is a measure of *total work done*. This is the amount of charge that traverses the potential difference present in a circuit. This process transforms the energy, mostly from electrical form into thermal form. The charge present in the system is not constant, nor necessarily are the potential differences. As a result, power changes continuously. Energy therefore is the integral of power during a period of time, per Eq. (1.1). It is typically expressed in Joules, or J.

$$E = \int_0^T P(t)dt \tag{1.1}$$

Applying both the concepts of energy and power, a system that sustains a constant power dissipation of 1 Watt for 1 second, will have transferred 1 Joule of energy.

Multi-threaded and multi-core

A number of the processors in this work require a distinction between *multi-core* and *multi-threaded* to be made. This culminates in the study of a system that has both of these properties. The term Multi-Threaded and Multi-Core (MTMC) is used to refer specifically to this type of system. For further clarification of the distinctions, parallelism's various forms in both software and hardware are detailed in Chapter 2.

Part I.

Background

Introduction

Part I of the thesis introduces the research and components that form the foundations of the contributions presented in Part II. There are three essential topics: parallelism, energy modelling and energy saving. These are each covered in turn, with the inclusion of the referenced research justified in relation to the goals of this thesis.

The final chapter in this part introduces the hardware platforms upon which the majority of this thesis bases its work. This chapter includes the work that was put into developing the Swallow system in a platform that was usable for the profiling, analysis and modelling presented in Part II.

2. Parallelism and concurrency in programs and processors

This chapter provides a review of the technology and concepts behind parallelism and concurrency in hardware and software. It starts with programming and multi-tasking concepts in § 2.1, then examining single-core parallelism in § 2.2 before reviewing multi-core technologies that are becoming increasingly prevalent in modern computing in § 2.3. Where appropriate, the literature is reviewed in the context of embedded systems, although a broader view is suitable for much of this chapter.

The distinction between parallelism and concurrency is important to the understanding of MTMC systems and how to express programs for them. *Concurrency* allows components to make progress independently of each other, such that in a given period of time, all of the components can have performed work. However, this can be achieved by sub-dividing the observed time period, allocating a division of that time to each component, so that at any given point in time, only one component is doing work. *Parallelism* provides simultaneous progression of components, therefore multiple activities can happen at the same time.

The notion of parallelism is present throughout the history of computing, with Flynn establishing a taxonomy of computer architectures that remains relevant today [Fly72]. From this taxonomy, both Single Instruction Multiple Data (SIMD) and Multiple Instruction Multiple Data (MIMD) require parallelism of some kind, both of which are relevant to this thesis. In addition, Single Instruction Single Data (SISD) implementations can also contain some degree of parallelism when sequences of SISD instructions are considered. These are all explored in this chapter.

In the software domain, programs may express solutions to problems in ways that are concurrent. These are activities that *can* take place at the same time, conceptually. The execution of these programs may be serialised and therefore not parallel, whilst still retaining the property of concurrency [AS83, p. 4].

Parallelism and concurrency exist across the hardware/software stack, from programming paradigms that aid the expression of concurrent problems [Pin98], techniques to parallelize sequential non-dependent operations, through to the necessity to parallelize hardware, brought about by technological limitations [Kah13].

2.1. Concurrent programs and tasks

There are numerous forms of concurrency exposed at the software and programming levels. Concurrency can allow multiple independent workloads to be processed simultaneously if support for parallelism is present in the system (§ 2.1.1). Alternatively, a single problem, when written appropriately, can be expressed as a concurrent program (§ 2.1.3). When communication or response to events is required, techniques to handle multiple events in a desirable order and with adequate responsiveness must be used (§ 2.1.4). All of these rely on multi-threading either in expressiveness or implementation. This section begins with an overview of multi-threading and several related terms.

2.1.1. Multi-threading

Multi-threading is common across all computing, from high-performance scientific computing, through general purpose and down to embedded computing. The basic principle is to express multiple activities that may take place concurrently. There is a distinction between a multi-threaded view of a system, and how those threads are actually executed by the underlying processor(s). Many of the implementation details at the hardware level are discussed in the subsequent sections of this chapter. However, in this section, the software and Operating System (OS) level are the focus.

Depending on the context of the system, threads might also be termed tasks or processes. The distinction between them, if one exists, may differ. For example, in the Linux kernel [Mcc02], which closely follows the Portable Operating System Interface (POSIX) threading standards, a *process* is an address space and set of resources dedicated to the execution of a program, while a *thread* is an independent path of execution within a process; there may be one or more threads in a process. A *task* is a basic unit of work in Linux. If a process is cloned and some resources shared between instances of that process, then a set of cooperating tasks is created.

Defining multi-threading

For the purposes of this thesis and in the context of embedded systems, where an OS or POSIX implementation may not always be used, the term *process* is avoided except where supporting literature uses it. Terms relating to *threads* and *tasks* are defined as follows:

Software thread A unit of sequential execution, which may form the entirety of a program, or may work alongside other threads to achieve a common goal. This provides concurrency.

Hardware thread A front-end to a processor retaining its own program counter and other registers, able to accommodate a software thread. The computational resources behind the front-end may be shared, allowing multiple threads on a single processor core. This provides parallelism.

Task A separation of units of work that may have constraints such as hard real-time deadlines. A set of tasks might be realised as a group of time-sliced software threads managed by a Real-Time Operating System (RTOS), or they might be allocated as separate hardware threads on a sufficiently capable system. In any case, some tasks may need to complete their activities within a given time period.

2.1.2. Parallel tasks

An embedded system may have multiple objectives to achieve, defining multiple tasks. For example, take an embedded real-time system which is responsible for controlling an industrial process. It may have multiple sensors to communicate with, each of which requires data processing, along with actuators that must be controlled based on the result of that processing. It may also need to provide interfaces for reconfiguring the parameters that direct the processing of sensor data or control of actuators. Several interfaces will be involved in such a system, possibly implementing a range of protocols, such as Inter-Integrated Circuit (I^2C), Ethernet and Controller Area Network (CAN).

Figure 2.1 depicts a USB audio application for an XMOS-based platform [XMO14a]. It comprises several inter-connected tasks. There are tasks for I/O over various interfaces, as well as audio processing. The I/O protocols each have timing requirements, defined by the standards and behaviour of the components that are using them. As such, the embedded system must be able to send and receive data to and from these interfaces within their specifications. Further, for the system to operate correctly, there may be additional timing constraints that need to be applied. For example, in the context of an audio application, delayed audio processing could result in undesirable latency, or audible glitches caused by lost samples.

In such a system, all of these tasks must be able to run with sufficient speed and frequency in order to meet the timing requirements. In some implementations, an RTOS may be used to help with allocation of resources to meet these requirements. There may still be a need for the system software developer to correctly define priorities.

In general purpose computing, the OS also has task scheduling responsibilities, although the majority of tasks, particularly those initiated by users, are not considered time critical or only have soft deadlines (a missed deadline is inconvenient rather than system-breaking). Different scheduling techniques are used depending on the OS used and how it is configured. For example Linux and Windows have different schedulers and scheduling options [BC05; Mic12].

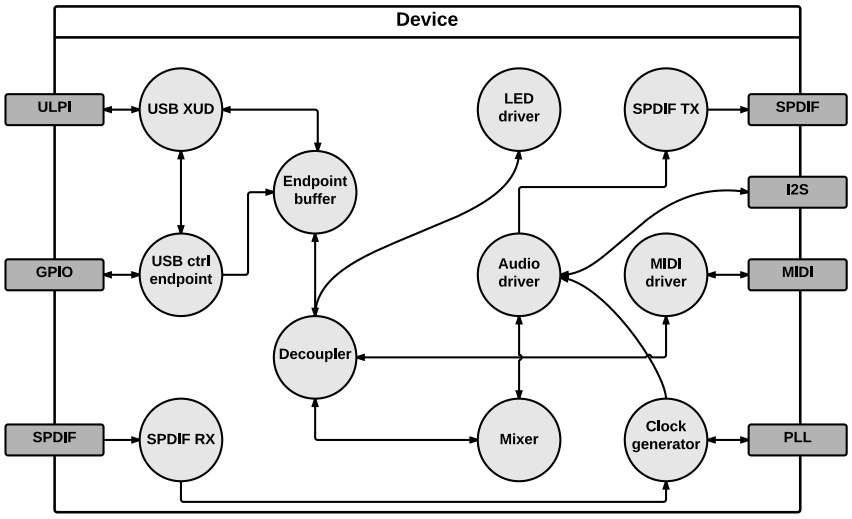


Figure 2.1: A multi-threaded task structure in a USB audio application.

2.1.3. Parallel programs

Certain types of programming problems, such as multi-stage processing, client-server and data parallelism, can be implemented in single programs that contain some level of parallelism. They can be distinguished from parallel tasks in that they cannot be separated from the other parts of the program and remain useful, or they are simply a replicated component. For example, an Ethernet interface task might be modularised to be used in multiple applications. A concurrent matrix multiplication algorithm, however, may replicate worker threads that each process a subset of the input data.

Software such as `pigz` [Adl10] allows data compression to be performed concurrently and is designed to exploit available parallelism in a system. The POSIX threading system is used for OS portability in `pigz`. The `sc_matrix` library, used in Chapter 7, expresses a number of vector and matrix operations concurrently, although it is targeted at bare-metal embedded programs rather than at devices running an OS and so exploits device specific parallelism features rather than a portable threading library.

Client-server arrangements can often exploit parallelism, in that a server may need to handle multiple clients simultaneously. The widely used Apache web server can use multiple worker threads or processes to serve a larger number of client connections simultaneously. The performance of such an implementation is both workload and configuration dependent, making it an area of interest to research in web technology [DKC08].

Many libraries and languages have been created to allow parallelism to be expressed in programs. Open Multi-Processing (OpenMP) [DM98] is a library that provides extensions to Fortran, C and C++ to enable shared-memory programming. The Message Passing Interface (MPI) standard [Sni98] provides methods for communicating between threads in parallel systems. Open Compute Language (OpenCL) [SGS10] provides a language and framework for leveraging parallelism in heterogeneous systems, allowing work to be allocated to different compute units, such as Central Processing Units (CPUs), Graphics Processing Units (GPUs) and Field Programmable Gate Arrays (FPGAs) [Cza+12]. There are many more languages, each expressing parallelism using different paradigms [Pin98], including Occam, MultiLisp, and Sire [Han14]. Message passing is a commonly used abstraction for parallel programing, two notable forms are formalised as Communicating Sequential Processes (CSP) [Hoa78] and the Actor model [Kow88]. The former uses communication channel ends with synchronisation, whilst the latter uses mailboxes at the receiver. The communication model of the XS1 architecture, described in Chapter 5, follows a CSP model of parallel processing. Other methods of communicating in process networks exist, either synchronous or asynchronous in nature [Mar11, pp. 21–118].

The choices for expressing parallelism in programs is rich and varied. To some extent, choices are driven by particular application areas. In the embedded space, a significant proportion of applications continue to be developed in C or its derivatives [Phi04, p. 151]. Although alternatives exist [Taf14], the inertia present from a significant amount of historic code, means that C is likely to remain a popular choice for the foreseeable future.

2.1.4. Event driven software

In event driven software, waiting on the availability of data, for example through I/O, is kept efficient by avoiding activities such as spin locks. Examples of this and alternative constructs are given later, in § 4.4. Event driven behaviours allow applications to wait without wasting CPU cycles, and for inputs to be queued for handling with minimal blocking. Event handling is an activity often handled by the OS, the software interfaces to which vary between OSs. Libraries such as `libevent` [Mat10] provide an abstraction layer on top of these various implementations. Languages that provide channel or other communication based abstractions must also rely on event implementations at a lower level in order to efficiently provide their data sharing model.

Software such as the `nginx` [Sys14] web server use events to handle the so-called *C10k problem*, where ten thousand client connections may need to be maintained simultaneously. Although the processing of this number of connections may not be fully parallel, the software architecture is able to accommodate this many open connections with low overhead.

In embedded computing, interrupts are frequently used to avoid polling of devices that may or may not be ready for some activity to be performed upon them (for example, a device buffer may be free to receive more data). Interrupts exist in both a hardware and software sense. A *hardware interrupt* uses an I/O signal to cause a context-switch in the processor that receives the signal. Typically, an Interrupt Service Routine (ISR) is entered, which deals with the cause of the interrupt, before returning to the previous context. Interrupts may be masked to avoid context switching in time-critical sections of software, and interrupts may also be nested or prioritised, so that multiple simultaneous interrupts, from numerous sources, can be handled appropriately.

If there is no computation to be performed, interrupts can be exploited for power saving. An idle processor can sit in a low power or sleep state until an interrupt triggers a wake-up into its fully active state. Thus, interrupts can be used not just for rapid context switching, but also power state transitioning. For example, many ARM devices feature *wait-for-interrupt* instructions that put the device into low power mode until an interrupt takes place. Similarly, the XMOS XS1-L can do this with both conditional and unconditional wait instructions.

A *software interrupt* uses a similar context-switch approach, but the activity is handled, and possibly initiated by an OS. For example, the OS may interrupt a running program to allow another to have processor time, thus achieving time-slicing multi-tasking. Alternatively, a program may cause a software interrupt in order to request a privileged activity from the OS, such as disk access.

Interrupts create scheduling challenges and potential context-switch overheads [Tsa07; TT09]. Certain multi-threaded architectures, such as XS1, also implement *events*. These are similar in behaviour to interrupts, except context is not preserved when an event takes place; the thread responding to the event simply jumps to a designated program location. This allows a thread to efficiently wait to respond to one of multiple possible events, whilst other threads continue to execute. The hardware architecture and distribution of work between threads then become the determining factors in responsiveness, rather than context switch and ISR overheads. The XS1 event handling implementation is discussed in more detail in Chapter 5.

2.1.5. Summary

This section has provided background on various parallel processing and concurrent programming paradigms, technologies and challenges. A number of these are relevant across computing while others are more specific to embedded systems or at the OS level.

In task concurrency, a program may comprise multiple threads, all working on independent tasks, with communication where necessary. Libraries such as POSIX threads, or a RTOS provide a means of defining these tasks.

Cycle	Stage 1	Stage 2	Stage 3	Stage 4	Stage 5
0	I0	—	—	—	—
1	I1	I0	—	—	—
2	I2	I1	I0	—	—
3	—	I2	I1	I0	—
4	I3	—	I2	I1	I0
5	I4	I3	—	I2	I1

Table 2.1: Example of a five stage processor pipeline including warm-up and stalling.

In a single concurrent program, inseparable tasks are composed together, with an expectation that each task can progress at any given time (save for synchronisation and communication). Programming languages such as Occam and Sire allow concurrency to be expressed in algorithms. This concurrency can then be used to realise parallelism on a suitably equipped hardware platform.

At the hardware and low level of software, interrupts and events allow asynchronous activities to be handled concurrently. There is motivation to minimise the overhead of handling these, both to ensure correct operation thus avoiding missed deadlines, and to provide good scaling, serving ongoing application challenges such as C10K.

The next two sections discuss how parallelism is made possible in hardware. Thus, the concurrency presented by the various techniques in this section have the potential to be exploited as parallelism by the underlying computer architecture.

2.2. Parallelism in a single core

At the hardware level, there are two main approaches to maximise performance. The first, is to increase the operating frequency of the processor, so that a newer, faster processor can deliver a higher throughput of work in a given unit of time. The second, is to do more work per clock cycle, such that a new processor with a higher Instructions Per Clock (IPC) can do more work in a given unit of time, at the same frequency.

The former has been maintained through lower threshold and operating voltages combined with increased transistor count at the same power density, per Dennard's scaling observations [DGY74]. However, at 130 nm feature size, this property has ceased to hold [Kuh09; Boh07]. This has resulted in a plateau in processor operating frequency since 2005 [Kah13]. To maintain competitiveness, processor manufacturers have sought and continue to seek methods for maximising IPC and throughput, creating parallelism of various forms in a single core.

If the objective of higher performance is substituted for lower energy, then a processor that can do more work per clock cycle than another can be run more slowly, and potentially at a lower voltage, whilst maintaining performance. In these conditions, it will most likely be the more energy efficient processor of the two.

This section examines methods of increasing IPC and throughput, all of which have an impact not just on performance, but on energy consumption and how such processors can be modelled for energy, as will be discussed in Chapters 3 and 4.

2.2.1. Pipelining

The majority of computer architectures break execution of instructions down into multiple stages, forming a pipeline. A sequence of instructions can be processed in this pipeline, progressing between stages on each clock cycle. For an N stage pipeline, up to N instructions can be executed simultaneously. The execution latency of an individual instruction is not improved by pipelining. However, the next instruction can begin to be processed before the current one is completed, thus increasing throughput. In addition to this, smaller stages typically allow a higher operating frequency for the processor, by shortening the critical path.

An example of instruction occupancy of a pipeline is given in Table 2.1. In ideal circumstances, the pipeline is always full. At the start of execution, the pipeline must *warm up*. Further, instructions may need to wait for an earlier instruction to complete before proceeding. For example, two

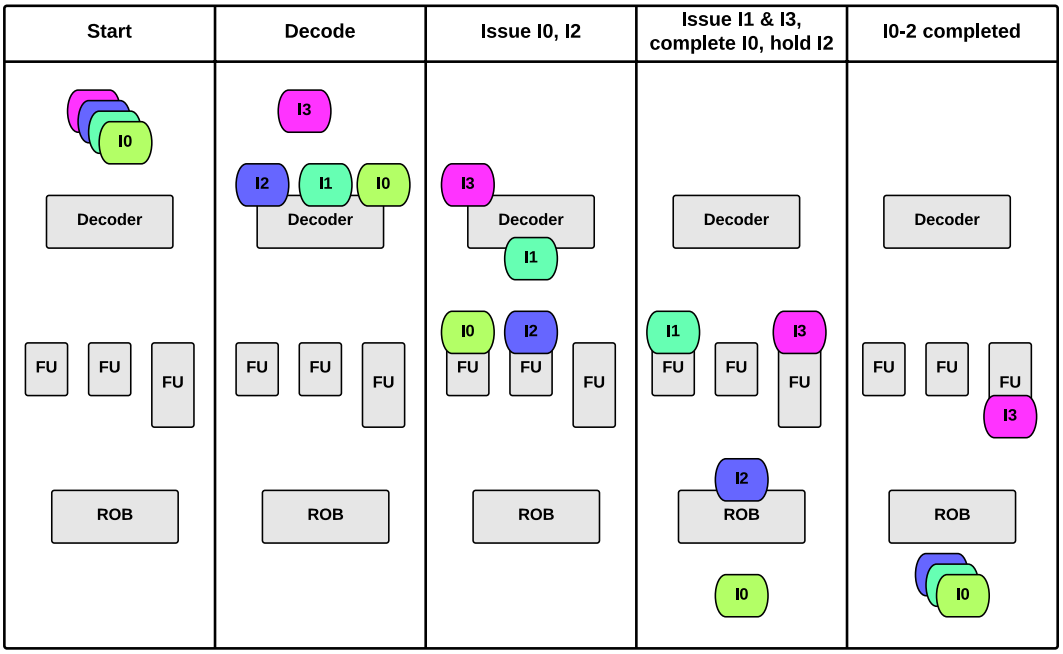


Figure 2.2: An abstract example of instruction flow through a super-scalar processor.

consecutive instructions may use the result of the previous instruction as a source operand. If the previous instruction is not completed before the current instruction proceeds, the wrong operand value would be used. This is a *data hazard* which must be detected and avoided, either by *stalling* the pipeline (shown at cycle 3 in the table), which reduces IPC, or adding forwarding logic into the pipeline [Pat85, pp. 17–19], increasing pipeline complexity.

Another example is branching, where a decision to branch may invalidate instructions that have already entered the pipeline. This requires the pipeline to be *flushed* (emptied), again reducing IPC. As the depth of a pipeline increases, the performance penalty from a flush increases. Branch prediction techniques [Smi81] or branch delays [Pat85, pp. 12–13] can be used to try to avoid this scenario.

2.2.2. Super-scalar

Where a processor possesses multiple Functional Units (FUs), such as Arithmetic Logic Unit (ALU), Floating Point Unit (FPU) and memory unit, IPC can be increased by attempting to utilise as many of these in parallel as possible. If one unit is in use and requires multiple cycles to complete, it may be possible to issue an instruction to another unit, provided there are no data hazards between instructions. These *super-scalar* designs allow multiple instructions to be executed simultaneously [Joh89]. Throughput can be improved further by allowing out of order execution, where instructions are issued internally in an order that maximises FU utilisation whilst avoiding data hazards, with re-ordering hardware at the end of the pipeline [HP06, pp. 104–114], so that instructions are seen externally to complete in the order that was expressed within the software thread.

An abstract view of a super-scalar processor is shown in Figure 2.2, capturing the progression of several sequential instructions at various points in time. The diagram shows several techniques that contribute to Instruction Level Parallelism (ILP) in a super-scalar system, including pipelining, sub-pipelining, multiple instruction issue and re-ordering. Initially, two of the instructions can be issued to different FUs, but a third cannot as the target FU is already in use. This results in instruction `I2` overtaking `I1`, so it must be held in the Re-Order Buffer (ROB). Once `I1` completes, the ROB can retire both `I1` and `I2`. `I3` is still executing at this point, because the sub-pipeline of the FU that it is utilising takes a larger number of cycles than some other FUs.

2.2.3. Hardware threads

Introducing hardware threads to a processor core allows further exploitation of the previously described methods by fetching multiple sequences of instructions that can be fed into the pipeline and FUs. The presence of multiple hardware threads provides the benefit of avoiding data hazards between instructions by having more than one context to choose from.

Hardware multi-threading requires multiple register banks, one for each thread, along with additional logic to fetch and buffer instructions from multiple memory locations. Further, there may be some replication in the instruction decode logic. This adds to processor complexity.

In ideal operation, multiple threads can be used to keep all FUs and pipeline stages full, to the benefit of IPC. However, it is also possible that threads may contend the same FUs, leading to similar performance to a single-threaded processor. At an OS level, the scheduler may need to be aware how a processor's multi-threading is implemented, as the OS may otherwise treat it as an independent processor core, assuming that resources on that core are uncontested.

An example of multi-threading can be found in Intel's Hyper-Threading [Int03a] technology, which provides two front-ends to a single super-scalar core and has been used in a variety of Intel processors including the Pentium 4, Xeon, Atom and Core product ranges. Other processors implement multiple cores each with multiple front-ends, such as the Sun Sparc 32-way Niagara processor [KAO05], which comprises eight cores each with four-way multi-threading. The AMD Bulldozer micro-architecture implements two threads per core, but with a shared FPU and two integer pipelines [But+11]. This created an example of the aforementioned OS scheduling issues, which needed to be resolved to ensure best performance, for example in the Windows 7 OS [Shi12]. The XMOS XS1 processor [May09b], which is described in detail in § 5.1, provides eight hardware threads, sharing a simple four-stage pipeline, in which IPC is only maximised when four or more threads are active.

2.2.4. Data parallelism

The previously described techniques all apply to SISD structures. However, SIMD can be exploited in a single processor core for certain data maniuplation tasks. Various ISAs contain extensions that provide SIMD instructions and registers. For example, ARM NEON [ARM14] and Intel Streaming SIMD Extensions (SSE) [RPK00] can perform instructions upon wide vectors of data up to 128 bits. Intel Advanced Vector Extensions (AVX) [Lom11] can operate on 256-bit wide data sets.

GPUs, in particular those with General Purpose GPU (GP-GPU) capabilities can handle a large amount of data parallelism per core. Such devices, while present in some high performance embedded devices, like mobile phones, do not fall within the area of research explored in this thesis.

Very Long Instruction Word (VLIW) processors conform to a MIMD organisation, where multiple operations are performed on a set of data in a single instruction. Such technology is most frequently used in embedded Digital Signal Processors (DSPs) [FDF98], where software-pipelined activities can be expressed as a series of VLIW sub-instructions. VLIW processors perform MIMD in lock-step, where the long instruction encodes the various operations that will be performed on each operand. Therefore, in VLIW processors, the compiler must be able to schedule instructions in order to maximise IPC and satisfy data dependences, otherwise hand-optimisation may be required to attempt to perform useful operations in the slots available in the instruction encoding.

2.3. Multi-core processing

Multi-core processors provide several independent processing units, with no contention for the resources on each core, forming a MIMD organisation. This forms the distinction between these and multi-threaded processors, where a multi-threaded processor creates the possibility to execute more than one instruction sequence simultaneously, but shares FUs internally and may not necessarily have MIMD characteristics. However, contention of resources is not completely removed by multi-core architectures. The memory hierarchy and interconnection between cores can still be contended and indeed this forms a significant problem in achieving good performance in multi-core systems. This is particularly significant if it is not possible to scale the interconnect with the rest of the

system, which is another observation of Dennard [DGY74] that is problematic in modern processor design [Boh07].

A multi-core processor has more than one core on a single die or chip, distinguishing it from a multi-processor system in structure. Ultimately a system comprising a large number of cores may be formed of multiple processors, each with multiple cores, and each capable of multi-threading. Indeed, this is the case for the Swallow system described in Chapter 8 as well as many server systems. For simplicity, this thesis refers to systems of multiple multi-threading capable cores as MTMC, distinguishing between chip-local multi-core and system level multi-processing only where necessary.

A wider view of the different types of multi-process architectures, predominantly from a general purpose computing and server perspective, is given by Roberts and Akhter [RA06, pp. 5–13].

2.3.1. General purpose multi-core

The first general purpose x86 multi-core processors were introduced in 2005, with both AMD and Intel offering dual-core products. The number of cores has since expanded, with currently announced products containing as many as 18 cores, for example the Intel Xeon E5-2699V3. Many ARM-based products are also multi-core, including a number of Cortex-A series devices, commonly found on mobile phones, but also in servers and high performance embedded multimedia devices.

Multi-core is reasonably well suited to general purpose computing, where even a single-user machine is frequently used for multiple concurrent tasks. These may include user-triggered activities, such as multimedia, web-browsing and gaming, but also compute-intensive background tasks, such as virus scanning and data indexing. However, there is still sufficient demand for single-threaded performance, that multi-core processors may offer aggressive Dynamic Voltage and Frequency Scaling (DVFS) strategies that provide temporary boosts to core frequency and voltage when a single thread will benefit. Intel's Turbo Boost [Int15] is an example of this. Such techniques are typically only temporary performance boosts because the power demand would push the processor beyond its Thermal Design Power (TDP) with prolonged use and sustained operation at this higher frequency/voltage would likely have a negative impact on device longevity and reliability.

2.3.2. Embedded multi-core

Multi-core processors in the embedded space include various designs, often targeting communication or other hard real-time environments. Companies including Picochip, EZChip, XMOS and Adapteva have developed architectures that directly serve embedded use.

The Picochip PicoArray processor [DPT03], contains an array of signal processing cells on a Time Division Multiplexing (TDM) network, principally for implementing cellular communication modems and codecs. The Adapteva Epiphany and EZChip Tile processors are described in more detail in Chapter 10, forming part of this thesis' discussion of modelling a wider range of multi-core devices.

The application space for embedded processors is different to that of general purpose computing. As such, the way in which multi-core is exploited is different. For example, the embedded system is typically utilised closer to the limit of its capabilities, in order to ensure cost effectiveness. Its life-span may be significantly longer than a general purpose device, either due to its more restricted set of uses, or the difficulty of access to replace or upgrade it. This is indicated by support periods for embedded versions of software such as Windows, for which the embedded variants have longer support periods than regular versions [Mic14]. The motivation for energy efficiency in software is therefore stronger, because the energy savings that may be obtained from hardware improvements may be less readily available in the longer product cycle. This, coupled with the scarcity of energy in many embedded system deployments, strengthens the motivation further.

2.4. Summarising parallelism and concurrency

Concurrency can be defined at multiple levels, from programming languages through to OS-level task definition. Parallelism can be provided by the physical computer architecture to allow exploitation of concurrency. Expression of concurrency or parallelism at one level does not necessitate

that other levels be aware of or exploit it. Architectures can exploit implicit parallelism that may be present in sequential programs, where independent groups of instructions may be executed concurrently for improved performance.

Converging upon multi-core processing, such hardware requires parallelism to be present at higher levels of abstraction in order for the hardware to be utilised efficiently. This efficiency can be measured both in terms of performance and energy. Processors that belong to this group form the core interest area for this thesis, wherein their relatively recent introduction poses new research challenges. This includes the complexity in programming them, modelling their behaviour, maximising their performance and minimising energy. While this chapter has focused on the concepts and mechanisms of concurrency and parallelism, it has not addressed energy efficiency at length. Considerations for energy efficiency, particularly with respect to embedded systems hardware and software, are examined in detail in Chapter 4.

The XMOS XS1-L processor, which is introduced in Chapter 5, draws upon several of the paradigms and technologies described in this chapter. Of particular interest are:

- Multi-threaded programming (§ 2.1.1) via a C dialect, `XC`.
- The multi-threaded pipeline in the processor core (§ 2.2.3).
- Multi-core (§ 2.3), featuring:
  - Channel based message passing in software.
  - Shared memory on a single core.
  - Hardware based channel communication on- and off-core, backed by a routed interconnect.

These are explored throughout this thesis. This review of parallelism has provided background on these and other complementary methods, in order to frame the unique contributions of this work around the state of the art and alternative approaches.

3. Energy modelling

This thesis seeks to establish both a single and multi core energy model for XS1 based embedded processors. The processor, which is discussed in Chapter 5, has multi-threaded and multi-core networking properties that necessitate new modelling approaches. In order to communicate the energy consumption of this architecture with respect to software, prior work must be considered in order to determine a feasible approach.

This chapter presents a review of energy modelling techniques at various abstraction levels, extracting useful techniques that are applicable to this thesis, identifying the further work needed within, thus justifying the research conducted in subsequent chapters. There are two main sections: hardware energy modelling (§ 3.1) and software energy modelling (§ 3.2), where the line between these sections is at times blurred. Therefore, these are simple categorisations, where hardware modelling best serves device designers and software modelling may better serve software developers. Both single and multi-core modelling techniques are considered, where some modelling areas have more significant developments for MTMC than others.

Table 3.1 presents the key features of each model approach, including a section reference that describes the approach in more detail as well as specific implementations and uses.

Model type	Summary	Ref
Hardware-oriented modelling		
Component parameter exploration	– Provide estimates based on key parameters of devices such as memory hierarchy, width, etc. – Rapid design space exploration. – Require external, lower-level simulations to provide the most reliable estimates.	§ 3.1.1
Modular system level	– Modular construction of various architectures and other system components. – Energy models can be attached to components. – Energy modelling accuracy decreases with more complex systems.	§ 3.1.2
Transaction level	– Analogous to activity (data/commands) exchanged system components. – Can provide a high level view of system behaviour. – Modelling of individual components can be done differently. – Harder to associate with software blocks than e.g. ISA level.	§ 3.1.3
Software-oriented modelling		
Functional block level	– Reflect energy consumed by functional units (multiplier, FPU, etc) at the behest of instructions. – Relationship between ISA and micro-architecture can be made. – More access to processor design details required, depending on profiling method.	§ 3.2.1
ISA level	– Device profiling and simulation to give ISA energy model. – "External effects" such as cache-misses considered. – Various implementations for different architectures.	§ 3.2.2
Performance counter based	– Estimate energy based on properties such as cache hit rate. – A possible alternative to direct hardware energy measurement. – Can also be used in simulation. – Requires hardware performance counters that provide sufficient data for accurate modelling.	§ 3.2.3
Software functional level	– Build database of energy consumed by software library calls. – Estimate program energy based on those calls. – Relies upon profiled library calls occupying majority of execution.	§ 3.2.4

Table 3.1: Energy modelling technique overview.

3.1. Hardware energy modelling

Approaching energy modelling from the perspective of hardware, physical properties such as device size, transistor behaviour and interconnection types dictate how energy consumption can be calculated. This section examines modelling approaches that describe these characteristics in a relatively high level of detail. They are still usable for energy consumption analysis of software. However lower level models are typically concerned with a level of detail that would make it impractical to model long sequences of instructions, particularly in terms of the investment a software engineer would be willing to put into such an activity. The time taken to perform this form of modelling does not fit easily into a software developer's compilation and testing process.

3.1.1. Component parameter exploration

Processors have a number of fundamental components, including processing elements, memories and interconnects. Exploring different configurations of these can give insight into creating a more energy efficient implementation of a processor.

CACTI

CACTI [WJ96] is a cache access, cycle time and power modeller that captures the behaviour of cache implementations with sufficient accuracy that its initial version is accurate to within 6 % of lower level electrical simulations such as those performed in Hspice.

The aim of CACTI is to represent the cache as a model of various key parameters, extended on prior research by introducing features such as a transistor-level model for the decoder and load-dependent transistor sizes. Version 1 of CACTI is purely for timing modelling. However, version 2 [RJ00] and beyond include power models. At the time of writing, the latest version of CACTI is 6.5 [HP 14]. It features a web interface and downloadable source, and focuses on many types of interconnect related to the memory hierarchy, including routed data and various wire types.

CACTI serves as a design space exploration tool for memory hierarchies in processor architectures. In addition, its power models can be exploited in energy models at various levels, where cache access patterns are available. For example, Bathen et al. utilise CACTI to establish memory subsystem power costs of software optimisations that seek to lower overall energy consumption [Bat+09].

The movement of data around a processor and the surrounding system can form a large part of the system's activity. As such, architectures where memory and cache access dominate the energy consumption can be modelled with reasonable accuracy with an appropriate CACTI configuration.

McPAT

The Multi-core Power Analysis and Timing (McPAT) tool provides a framework for modelling various micro-architectural properties of contemporary multi-core systems, with a view to estimating the inter-related characteristics of power, size and timing [Li+09]. McPAT uses eXtensible Markup Language (XML) files as a configuration interface to its underlying models, allowing external performance, power and thermal simulators to be used as a source of further simulation data. McPAT can be seen as a similar tool to CACTI, but one that is aimed at capturing characteristics of more than just the memory hierarchy.

Early examples of McPAT processor models yield power and area accuracy ranging between −10.84 % and −27.3 %. These examples include Niagara, Alpha and Xeon architecture variants. McPAT is then demonstrated as a design space exploration tool by examining the effect of changing various parameters such as feature size and number of cores per shared cache. This was then used to demonstrate that at 22nm, a 4-core cluster gives a better Energy Delay Product (EDP) than 8 cores.

More recently, McPAT has been combined with the Sniper x86 performance simulator, to give early energy optimisation opportunities for hardware and software that are both still in development [Hei+12]. Performance counters from Sniper include component utilisation levels (duty cycles) and cache miss rates. The reported error is between 8.3 % and 22.1 % when compared to a real Intel Nehalem based system measured at its 230 V AC supply input. The performance counter

estimations are then used to implement improvements energy efficiency by 25 % and performance by 66 %. This results in a software implementation that is a particularly good fit to the underlying hardware. Finding a good fit was expressed in § 1.1 and is considered important in creating low energy software. This is discussed further in Chapter 4.

The memory hierarchies and performance counters that relate to them form a significant part of this type of performance and power modelling. This serves relatively large processor architectures well. However, in smaller, simpler architectures targeted at embedded systems, the memory hierarchy and system structure can be quite different, as is explained in Chapter 5 with respect to the XS1, and Chapter 10 in relation to a variety of other architectures.

3.1.2. Gem5

Gem5 is a freely available [Gem14] system level simulator that allows a combination of components to be characterised and assembled together. Typically, Gem5 is used as a platform to simulate new system designs in order to perform design-space exploration and the possibility to test software prior to the construction of a physical system.

The Gem5 simulator, as described by Binkert et al. [Bin+11], draws upon the work of two prior simulators — M5 [Bin+06], with its configurable ISA and processor models, and GEMS [Mar+05], which has good memory and interconnect configuration and simulation capabilities — to create a highly configurable simulator for a full embedded system.

Accuracy in Gem5 varies depending on the complexity of the system and the behaviour of the software that is executed on the simulated platform. Butko et al. [But+12] demonstrate that errors in performance modelling can reach almost 18 % if there are heavy external memory accesses to a complex Double Data Rate (DDR) Dynamic Random Access Memory (DRAM). However, in other cases performance error is as low as 1.39 %.

Energy modelling with Gem5

Although Gem5 not itself an energy modelling tool, its modular nature allows energy models to be attached to the components that are assembled into a simulated system, from which energy data can be extracted.

Rethinagiri demonstrates a system level power estimation tool [Ret+14] that uses performance counters from various simulated components in Gem5, combined with other properties of the system, in order to estimate the power for various ARM-based systems. Performance counters include features such as external memory accesses, cache misses and the IPC of processor cores. System properties include bus and core frequencies. A set of simple assembly programs, consisting of test vectors of small algorithms, were used to characterise the costs that needed to be associated with the various performance counters and system properties.

This modelling technique is shown to achieve less than 4 % error on average, which performs significantly better in comparison to McPAT. However, the research focuses on a single frequency and voltage operation for each processor core, where the static property of frequency is the dominant term in all cost equations, thus the relative error from the simulated components (performance counters) is low. This is more profound in the Cortex-A9 dual-core processor, where the measured system power varies by single percentage points across all tests. It is unclear how the estimation model would perform when DVFS or other aggressive power saving features are used.

3.1.3. Transaction level modelling

In Transaction Level Modelling (TLM), a system's components can be represented at different levels of abstraction, but the modelling is driven from the perspective of data exchanges between these components. System activity is viewed as combinations of reads and write events, possibly with timing information attached to these events. The components involved in the transaction can then update model state based on parameters given by the transaction. In the context of energy modelling, the focus is upon using these transactions to increment energy consumption of components as they act upon transactions, yielding an overall energy consumption estimate for the system. The motivation behind this approach is improved performance over finer-grained

modelling such as gate-level simulation of the entire system, allowing more rapid design space exploration whilst retaining good modelling accuracy.

TLM is demonstrated for a Multi Processor System on Chip (MPSoC) in [AND07]. The work models an MPSoC with a group of MIPS R3000 procesors combined with caches, a crossbar interconnect, Static Random Access Memory (SRAM) and other peripheral components. Each of the components has an energy model associated with it, which can be constructed separately using an appropriate method. For example, the SRAM component is modelled based on data extracted from analogue simulation of a range of device configurations, yielding a parameterised model based on the device's capacity and organisation (e.g. word length).

The final SRAM model considers three activities in relation to the TLM: read, write and idle. The underlying behaviour of the state machine that forms the SRAM, which would be present in a cycle-accurate simulation, is omitted in this approach. The work demonstrates a speedup in simulation of more than 20 % compared to a cycle accurate approach for 16 processors, with power estimation error of less than 10 % in all test cases. Designs with smaller numbers of processors and larger caches were the most accurate, as error from simulation of contention in the interconnect has less of an impact.

Other research has extended TLM to support multiple levels of accuracy within the modelling framework. Beltrame et al. [BSS07] are motivated by providing good simulation performance that uses more resources to model *interesting* parts of the simulation. Conversely the *uninteresting* periods are simulated at higher speed with lower accuracy. The communication channels that carry the transactions and the modules (or components) that they interact with must be modelled at more than one level of detail in order to make this approach possible. It is left to the designer of the target system to choose where to switch between accuracy levels, although the research does provide the accuracy/performance trade-off which can help in making this decision.

3.2. Software energy modelling

A software-centric energy modelling approach can, as with hardware, take place at various levels of abstraction with trade-offs in performance, accuracy, and the granularity at which information can be presented. This section begins the functional block level, then increases the abstraction level through the ISA, performance counters and finally at a purely software level.

3.2.1. Functional block level modelling

A similar approach to the above seeks to identify processor energy by modelling activity within particular functional blocks in the processor. For example, a processor may have various functional units for simple arithmetic, multiplication, division and memory operations.

In [IRF08], a TI C6416T VLIW processor is separated into six blocks: the clock tree, fetch and dispatch components, the processing unit, internal memory and the L1 cache, split into data and instruction parts. Parameters to the model include read and write access rates from and to these components, along with cache miss rates. Validation across a series of DSP-centric benchmarks shows a worst case energy estimation error of 3.6 %.

Other work explores a different set of processor designs and alternative methods for classifying functional groups and associated instructions. Blume et al. [Blu+07] show that classifying instructions into 6 groups for the ARM 940T is the optimal grouping, where fewer classes significantly increase error and more yield only a small improvement. Additional characteristics such as memory must also be accounted for. This is presented as a hybrid functional/instruction level power analysis approach, with a worst case estimation error of 9 % for the aforementioned ARM 940T and 4 % for the OMAP5912 across a set of benchmarks. The work also shows that reducing model complexity by ignoring behaviours such as cache misses can result in estimation error increasing by over five times.

3.2.2. ISA level modelling methods

Tiwari's early work into x86 instruction set modelling [TMW94b] seeks to estimate the energy, E, of a program, p, by considering three components of execution: the *base instruction cost* of each

instruction that is executed, the *inter-instruction overheads* of switching between one instruction and another, and any *external effects* such as cache misses. These values are extracted from a target system with a test harness executing specific instruction sequences and measurement equipment collecting energy consumption data. The model is then expressed by Equation 3.1 [Tiw+96]. For all instructions, i, in the target ISA, the base instruction cost, B_i, is multiplied by, N_i, the number of times the instruction, i, is executed. For each pair of instructions executed in sequence, i, j, the inter-instruction overhead, $O_{i,j}$, and frequency of occurrence, $N_{i,j}$, is counted. Finally, for each external component, k, the energy cost of external effects, E_k, is determined, for example with an external cache model.

$$E_p = \sum_i (B_i \times N_i) + \sum_{i,j} (O_{i,j} \times N_{i,j}) + \sum_k E_k \tag{3.1}$$

Building on this research are energy modelling tools such as the Wattch framework [BTM00]. Wattch produces energy estimates of software through simulation, by modelling key components of a processor architecture, such as the cache hierarchy and size, functional unit utilisation and branch prediction capabilities. Wattch can model software targeting various architectures, to within 10 % of commercial low-level hardware modelling tools. The SimpleScalar [Aus02] architecture modelling software was used as a basis for a similar power model, resulting in Sim-Panalyzer [Sim04].

The idea of measuring instructions and their interactions can be broken down further, a model for which was proposed by Steinke et al. [Ste+01a]. This model extracts more information on the source of energy consumption in the processor pipeline, such as the cost of switching in each read action upon the register file, as well as the cost of addressing different registers for read and write-back. The precision of the approach is shown to be within 1.7 % of the target hardware, although it significantly increases the number of variables that must be considered when implementing the model. Other types of processor architectures have also been modelled in similar ways, such as VLIW DSPs [Sam+02; IRH08], with average accuracies of 4.8 % and 1.05 % respectively.

To model complex micro-architectures or large instruction sets, linear regression analysis can be used. With sufficient supporting empirical data, a solution to a parameterised model can be found that establishes values for any unknown terms. This has been utilised to model an ARM7TDMI processor [LEM01], using empirical energy data from observed test programs to aid the solver, yielding a model with a 2.5 % average error.

These approaches can all deliver an accuracy of 1–10 % across various architectures. However, the architectures that they analyse are either single threaded, or special purpose DSPs. As such, these models are not equipped to model a hardware multi-threaded processor. Either these approaches must be extended, or an alternative approach found, ideally whilst maintaining comparable accuracy to prior models.

3.2.3. Performance counter based modelling

In a number of modelling methods, hardware performance counters are used to estimate energy consumption. The benefit is that these counters can be used by a wider range of users who do not necessarily possess direct energy measurement capabilities for their target system.

In [CM05], a set of performance events are monitored via an Intel PXA255's configurable performance counter sampling mechanism. These include characteristics that have been modelled via various means throughout the literature review in this section, such as cache misses, but also instruction counts, data dependency events and an abstraction of main memory behaviour through some of these events. The work states that the embedded PXA255 has fewer counters than a larger processor, requiring profiling runs in order to gather sufficient data for a robust model. It is shown that the average error is 4 % for the SPEC2000 and Java benchmarks run on this processor via a Linux based OS.

The Xeon Phi, whose architecture is discussed in Chapter 10, is modelled in a similar way [SB13]. In this case, a set of micro-benchmarks are used to exercise various behaviours and extract a performance counter lead model. The Phi is significantly more complex than a PXA255, in that it contains multiple x86 cores and multi-threading. As such, multi-threaded behaviour must also be accounted for, with the model containing a scaling term defined by the number of active threads in a core. The model accuracy is stated as being within 5 % of hardware energy for real world

benchmarks, and the information from this model is used to demonstrate that code from the Linpack benchmark suite can be energy optimised for the Phi, increasing efficiency by up to 10 %.

The previous examples use performance counters in lieu of direct energy measurements, the motivation being that instrumenting most hardware to measure energy is time consuming or a technical barrier for software developers. However, simulated performance counters can be used, to estimate the energy consumption of a program on a device that the developer does not have access to. For example, architecture simulators such as Sniper and system simulators such as Gem5 can provide performance counters that can be used to the same effect as those found in real hardware. Modelling that centers around these simulators is discussed earlier in this chapter, in § 3.1.

3.2.4. Software functional level modelling

Analogous to modelling hardware activity at a functional block level, the compartmentalisation of software into libraries of frequently used functions can also be used as the basis for an energy model. In this case, the structure of the underlying hardware is not a concern. Instead, the energy consumption of sets of software library calls is measured and these are then used in an energy model, based on the frequency with which each of the calls is made within a program [Qu+00].

The work of Qu shows that if a suitably large database of library call energy consumption is built, a program's energy can be modelled to within 3 % of hardware. For this method to work well, it is assumed that a program spends a majority of execution time in library calls, therefore executing code for which the energy cost is already known.

Beyond the challenge of having a suitable distribution of library calls in the modelled program, there are several other potential issues to using this method. Firstly, the architecture-oblivious approach may require re-profiling of library calls if a new target processor is used in order to preserve accuracy. Secondly, library calls for which energy consumption is heavily dependent on the supplied arguments may have poor accuracy if these dependencies are not considered.

3.3. Summary

This chapter has provided a review of various energy modelling approaches for a broad range of architectures. A number of these approaches focus on detailed hardware characteristics, for example pipeline and functional unit behaviour, whilst others utilise indicators at a higher level, such as performance counters or library calls. The accuracy of these models vary, from within single digit percentage error, up to 20 % or more. The trends observed in published works suggests that sub-10 % error margins are more than sufficient for *useful* energy modelling of software. The comparisons drawn between approaches with respect to their accuracy is somewhat subjective, due to various ways in which accuracy is calculated and the ground truth energy figures obtained, i.e. low level gate simulation versus direct hardware measurement.

The granularity of modelling can affect the impact that such errors have. The wider the view taken, the less of an impact sub-components have. For example, an energy model of a register file might have an error margin of ±15 %, but if its contribution to the processor's energy consumption is only 10 %, its effects are diminished. A pragmatic approach to evaluating error must therefore be taken, where the *impact* of an error must be considered alongside its magnitude at a given level of detail.

Many of these works are for single-threaded processors, although it is shown that more recently, multi-threaded and multi-core architectures can also be modelled and energy consumption estimates for software running upon them can be given. However, in the embedded space, multi-threading and multi-core is less prolific that in High Performance Computing (HPC) or larger, more general purpose systems. Moreover, there remain novel architectures for which successful energy modelling approaches are not demonstrated or sufficiently explored. One such embedded architecture — the XMOS XS1 — forms the focus of this thesis.

This thesis is motivated by energy modelling of *software* and design space exploration where the software is the focus of the design effort. As such, modelling approaches that lean towards the software part of the stack provide the best foundation for further work. The energy consumption of

the hardware must be relatable to the software running upon it in order for meaningful information to be made available that could prompt energy-saving changes to the software. In particular, modelling at the ISA level provides a good intersection between the hardware and software because it is in many senses the bridge between the two domains. This will be the main abstraction level developed further by this thesis, although other levels will necessarily be incorporated and extended as well.

4. Influencing software energy consumption in embedded systems

The previous chapters have reviewed parallelism paradigms for software and hardware, as well as existing energy modelling approaches. This chapter establishes the design space exploration challenges that are present when attempting to save energy in an embedded system. This comprises general approaches that are applicable to all systems and software, but also includes specific focus on the additional constraints that are present in the embedded hardware/software design space.

This analysis of energy saving discusses both requirements and techniques. First, a set of objectives are discussed in § 4.1. Then, § 4.2 reviews the various interacting facets that govern energy consumption, giving consideration to constraints that are present in embedded real-time systems. § 4.3 focuses on historical and continuing reliance on *Moore's Law* and how the technology roadmap has changed in recent years in response to the ceilings of certain physical constraints being reached, and the increase in demand for devices that are more energy efficient. In § 4.4, the virtues of event-driven programming are explored, where the avoidance of waiting-loops can dramatically improve both performance and energy efficiency. The final section summarises the background that has been covered and relates it to the contributions this thesis makes in Part II.

4.1. Forming objectives to save energy in software

Taking a software-oriented approach to saving energy, there are multiple opportunities for the savings to be made. However, the scale of impact varies between approaches. Therefore, objectives must be enumerated with appropriate priorities, in order to yield the best results and to avoid simply deflecting the inefficiency into another area.

In § 1.1, a number of research questions and thesis statements were made. This included the argument that making sure software is a good fit to the hardware is essential for energy optimised systems. This section expands on this argument and goes deeper into the energy optimisation process, drawing on arguments made by Roy and Johnson [RJ97].

Roy and Johnson's work details a number of techniques and considerations for energy optimisation in the design of software. Written in 1997, the state of embedded system processors is somewhat different to how they are now, but the remarks made still apply, albeit with some adaptation in places. These will now be summarised and related to contemporary embedded processors and software.

4.1.1. Algorithm choice is the first and most important step

A poorly implemented piece of software is likely to perform poorly. In terms of energy, this can be manifested by using algorithms that do not fit well with the computation hardware. For example, code that relies on frequent divergent branching will not fit well to GP-GPU pipelines, leading to significant inefficiencies, harming both performance and energy consumption. Similarly, code that is compiled for a generic instruction set, rather than a specific instruction that that includes additional features present in the target processor, misses out on opportunities to perform optimally.

An excellent example of this exists in general purpose computing, wherein a long-established binary-heap tree algorithm is shown to be sub-optimal when virtual memory is considered [Kam10]. When given correct consideration to the underlying system, in this case including an OS, the algorithm can be improved to deliver a further ten times performance in some cases. Specifically, the proposed modification to a binary-heap tree populates memory pages vertically, matching the direction in which the data structure itself is populated. This results in fewer page changes during

accessing the data structure. This gives a very strong argument for knowledge of elements in levels of the system stack below that at which a developer is actually implementing software. Virtual memory is not typically a consideration in an embedded system, but other similar factors, such as RTOS driven context switching, can be considered similarly important.

Fundamentally, this change affects performance the most. However, execution time is a significant factor in energy consumption too, and so remains essential when considering energy optimisation. Las Vegas style algorithms become an interesting counter-example to this, when parallelism is exploited in a system. A Las Vegas algorithm will always produce the correct answer, but some randomisation in its approach means that the execution time is variable [LE94]. If such an algorithm is replicated in parallel, there is a higher likelihood of finding the fastest possible variant. However, doing so effectively wastes the energy consumed by all instances except the fastest. Thus, parallelized Las Vegas algorithms can potentially be time-optimal, but very bad for energy consumption.

4.1.2. Manage memory accesses

The previous examples also apply somewhat to this particular claim. Accessing memory takes time, and in a memory hierarchy, that time can be unpredictable due to caches, with a significant difference in both time and energy consumption in the worst case. In [Kam10] the concern was in virtual memory, but here, the physical memory implementation is the subject of interest.

In a memory hierarchy, the further away that data is kept, the longer it takes to access. At each level of caching, the performance penalty is typically an additional order of magnitude [Dre07, p. 16]. For example, if register accesses take a single cycle, then a level-1 cache access may taken in the region of five cycles, whilst a level-2 access could take 15. A main memory access may take hundreds of cycles. Although some components may be able to enter a low power state during longer-running memory accesses [CSP04], avoiding these accesses altogether is preferable.

Once again, improvements in this area are best achieved by modifying the algorithm, finding ways to reduce the memory footprint and establishing a memory access pattern that makes the best use of available caches. In an embedded system, the memory hierarchy may be far simpler than a general purpose processor, to the point where the may be no caches at all. However, memory access should still be considered.

At the very least, register accesses are faster and consume less energy than accessing Random Access Memory (RAM). Thus, ensuring that spills to memory are minimised will help performance and energy. Many embedded systems execute code directly out of flash memory, allowing them to have a smaller RAM. However, even if access times between flash and RAM are equal, the energy consumption may differ, such that relocating frequently accessed code segments to RAM may be preferable [PEH14]. This optimisation can be performed by a compiler, but a developer may be able to indicate the best candidate code sections to apply this optimisation on, given their understanding of the algorithm.

4.1.3. Utilise parallelism

If a processor has parallelism in it, then using these features will further improve performance. When suggested by Roy and Johnson, this mainly considered parallelism in a single core, as discussed in § 2.2. This can now be expanded to include multi-core, although at the multi-core level, the algorithm must be fundamentally parallel in order to exploit the hardware fully, returning the priority back to the original goal in this section: mapping the algorithm to the hardware. More subtly, implicit parallelism in code, such as independent sequences of instructions can be exploited through the compilation process or software pipelining.

4.1.4. Utilise power management features

This goal somewhat counter intuitively sits relatively low down the list of priorities. If the underlying device has power saving features, such as low power states, DVFS or the disabling of inactive units, then they should be made use of. This implies that these features are software controlled, which is not always the case. For example, tuning of device voltage can be done with

an appropriate combination of hardware and software [KE15a], but this can also be done purely in hardware [Bur+00].

Utilising power management will save some energy, but the saving may be less than could be obtained through the previously stated measures. The most apparent reason for this is that the number of operations that the software needs to perform is not affected by such measures. In an embedded system context, the best that can be achieved is to lower the frequency and voltage to the minimum that enables deadlines to be met, then disable unused components. However, changes at higher levels may instead use more components, but reduce the number of operations, thus allowing even more aggressive power saving to then be applied. The balance of priority here clearly leans towards the algorithm first, then power management; there is not a *chicken or egg* dilemma to resolve.

4.1.5. Minimise inter-instruction overheads

This is the final goal defined by Roy and Johnson, and operates at the same level as a number of the energy models described in Chapter 3, such as the Tiwari ISA model [TMW94b]. At the ISA level, the trace of instructions that are executed is valuable for determining energy consumption and it is shown that the precise sequence can have an impact through changing inter-instruction effects. However, the gains to be made from careful instruction ordering and register addressing are small compared to other efforts.

This assumes that the scheduling of instructions is purely to minimise switching activity in the processor as the sequence of instructions progress through the pipeline. This does not have an impact on performance. However, if the re-scheduling allows parallel utilisation of FUs or avoids a pipeline stall, this would improve performance. In this case, the change would be categorised in one of the earlier goals.

4.1.6. Multi-threaded, multi-core specific considerations

The previously stated goals largely hold today, as they did when originally stated. However, additional multi-threaded and multi-core considerations can be added with respect to a number of the points made.

A sequential algorithm does not necessarily parallelise well, if at all. As such, a new dimension is added to the challenge of developing an algorithm that maps well to the underlying hardware. In particular, giving consideration to Amdahl's Law [Amd67], sequential parts must be kept to a minimum.

Certain strategies, such as instruction scheduling to minimize switching, may be impossible in a multi-threaded system. If the instruction sequence in the pipeline is sourced from multiple threads, then the ability to control that sequence in a normal program is likely impossible. Fortunately, there are other higher priority strategies that will yield better energy reductions regardless of this.

In a multi-core system, the management of memory access becomes a more complex problem than before. If shared memory is used as the underlying mechanism for communication between threads (regardless of the programming model that is used), then the latency of the memory subsystem can be a significant bottleneck, reducing performance and costing energy. The impact of the memory subsystem may vary, depending upon which threads are exchanging information. Higher level caches, such as level-2 or level-3, may be shared between multiple cores, reducing latency in data sharing. Multi-core caches must implement coherency mechanisms to ensure that changes made from one core are visible to other cores when needed. The access of each core to memory may not be equal in terms of performance of connectivity, resulting in a Non-Uniform Memory Architecture (NUMA), further reducing the ability to predict how to make changes that will yield an improvement. In a larger scale system messages may need to be passed along a network in order to access information in the main memory of another compute node.

Departing from shared memory and instead using a message passing implementation, such as that described for the XS1-L and Swallow in Chapter 5, may remove a number of layers of complexity from the memory hierarchy, but still requires more consideration than single-core software development. If messages between cores are traversing a network, then the bandwidth and latency of that network must be considered. If a core is able to do other work whilst waiting for a network

communication to take place, then latency can be hidden. However, whilst in such a case performance may be improved, the energy cost may be higher than optimal. For example, if a program suitably hides latency, but communication takes place between cores that are more distant than is necessary (task placement upon the set of available processors is poor), then energy could be reduced by re-arranging tasks. In addition to this, many intercommunicating threads may create contention in the network, thus increasing latency and reducing throughput in an unpredictable manner.

4.1.7. Summary

This section has examined a list of goals that a software developer can seek to achieve in order to reduce the energy of their program. These goals were then related specifically to embedded systems where appropriate, and then extended from the original set [RJ97] to consider multi-threaded and multi-core systems, which are the focus of this thesis.

The first target of energy optimisation in software should always be making the program a good fit to the underlying hardware. The benefit of exploiting more fine grained goals is small in comparison, therefore should only be explored after the best possible fit is found. As such, strategies such as DVFS and switching minimisation through instruction scheduling should not be the first optimisation activities.

In multi-core systems, both shared memory and message passing architectures add complexity to the challenge of reducing the cost of exchanging data between threads. However, once the processing power of the system is utilised optimally, the movement of information becomes the most important goal, and so understanding the latencies and bandwidths present when moving data, as well as how much these characteristics may vary, is particularly important.

4.2. Energy's many relationships

The previous section focused on energy optimisation strategies from a software perspective. This section examines the underlying hardware properties that dictate how energy is consumed and how changes can be made to reduce (or increase) energy consumption. The properties and the interactions between them, form necessary understanding for the energy profiling and modelling that is performed in Part II, and provides clarification for some of the reasons for the order of priorities given in § 4.1.

4.2.1. The power, time, energy triangle

The relationship between power, time and energy was defined within the introductory chapter (§ 1.4). The distinction between power and energy is essential for this work, whereas in other contexts it may be possible to interchange the terms without consequence.

In its simplest form, $E = P \times T$, where the energy consumption of a system, E, is the product of the power dissipation, P and time, T. Power is not usually a fixed value in a system, because system activity is constantly changing, resulting in an integral form of the equation Eq. (1.1). An intuitive energy saving objective is to minimize P, T or both simultaneously, in order to lower the energy consumption of a system. However, the changes that must be made to deliver such a reduction have to work within the limitations of the system itself and the components that form it.

Seeking to improve one parameter may in fact have an opposite effect on the other. In such cases, the desire is for the opposing negative impact to be less than the positive impact that is introduced.

4.2.2. Supplying power

In the previous subsection it is clear that energy can be reduced if time is reduced, even at the cost of increased power dissipation, provided the former is more significant, giving a desirable net effect. However, the change in power profile may have effects reaching beyond the computational parts of the system.

Batteries are capable of storing a certain amount of charge, in order to provide energy to a system when no other source is feasible. However, the rate of energy transfer (power), has an impact on the available charge, or effective capacity of the battery. It is shown in [PW99] that battery capacity is affected by various factors, but of particular interest is the current and its behaviour over time.

A higher current (implying higher power dissipation if the voltage is unchanged), reduces the efficiency of the battery, thus under higher loads it will not provide as much total energy to the system. Further, lowering the average current is not necessarily sufficient to improve efficiency. Pedram et al. [PW99] also show that a current with high variance also has a negative impact on efficiency.

Reconciling this against some of the previously described energy saving scenarios, it may not be desirable to make optimisations that result in a smaller execution time if the power profile is higher, or becomes more variant.

Other power sources, such as DC-DC converters, also have current and voltage dependent efficiency characteristics. A concrete example of this is the supplies used in the Swallow system used in this thesis. The NCP1529 [ON 10] converters are most efficient at approximately 5 % of their maximum rated output current. At very low current, efficiency is extremely poor (asymptotic to 0), with a less dramatic reduction in efficiency as the load increases towards the maximum rated current.

The consequences of making poor choices when seeking energy savings may vary depending on the power supply. For example, a current with high variance in a battery-powered system may result in the device ceasing to work before it is expected, and it may be non-trivial to access the device and replace the battery. However, sub-optimal DC-DC efficiency may not be so catastrophic. Nevertheless, one cannot aggressively seek changes to execution time or power dissipation without also considering the behaviour of the power supplies in the system.

4.2.3. Power dissipation in silicon and DVFS

The technique of DVFS is motivated by a desire to minimise energy consumption by balancing the trade-off between power vs. performance for a given workload [Bur+00]. In the Complementary Metal Oxide Semiconductor (CMOS) technology used by the majority of processors, DVFS is affected mainly by two components: static and dynamic power.

Static power

The main component of static power is the leakage current of the transistors in the silicon. This is present regardless of the on/off state of transistors. As processors are fabricated on smaller process nodes, the percentage of overall power dissipation that is attributed to leakage is growing [Kim+03], for example due to increased leakage through thinner gate oxide layers, which must be combated with technology such as improved high-k gate dielectrics [WWA01].

$$P_s = V I_{\text{leak}} \tag{4.1}$$

In Equation 4.1, the static power, P_s, is proportional to the product of the device voltage, V, and the leakage current, I_{leak}. This is a simplified linear relationship between operating voltage and static power. In reality, the relationship is quadratic and dependent on multiple factors, including supply voltage, temperature, feature size and gate oxide thickness [BR06]. However, a linear approximation can be sufficient at a high level of modelling that does not reach any extremes of operation. Most importantly, however, static power is not directly influenced by circuit switching activity.

Dynamic power

Power dissipated in order to switch transistors on or off is termed dynamic power, P_d, and is expressed in Equation 4.2.

$$P_d = \alpha C_{\text{sw}} V^2 F \tag{4.2}$$

C_{sw} is the capacitance of the transistors in the device and α is an activity factor or the proportion of them that are switched. Activity factor is workload specific, but often estimated as switching half of the transistors in the device [BTM00], giving $\alpha = 0.5$. F is the operating frequency of the device. Observe that changes in V have the biggest influence on dynamic power dissipation.

A reduction in V, however, will slow the transistor switching speed, increasing the delay in the critical path, requiring that F also be lowered. Thus, there is a trade-off between reduced power dissipation and the total energy consumption due to longer execution time — in some cases it is not beneficial to slow the device down further. Choosing a strategy for energy saving, be it tuning the frequency to avoid slack time, or racing to idle by operating at high speed briefly, then reducing to a low power state, is dependent on the type of work and the behaviour of the system; there is not one strategy that works in all cases [ANG08].

The relationship between voltage and frequency varies depending on manufacturing process and device implementation. Simplistic representations, such as that in [Kim+03], represent the relationship as $F \propto \frac{V - V_{th}}{V}$, where V_{th} is the threshold voltage of the transistor. This representation projects that as V approaches V_{th}, F approaches zero. However, Sub-Threshold Voltage (STV) operation *is* possible [Zha+09] and F only drops exponentially, therefore slow, very low voltage devices can be made.

Working above STV, the nominal operating frequency and voltage, F_{norm} and V_{norm} respectively, can therefore be represented as Equation 4.3, taken from [Kim+03], where V_{max} is the maximum operating voltage of the transistor.

$$V_{norm} = F_{norm}\left(1 - \frac{V_{th}}{V_{max}}\right) + \frac{V_{th}}{V_{max}} \tag{4.3}$$

A step reduction in voltage requires a larger step reduction in frequency. With a conservative view, where preserving correct operation is required, the relationship can be represented linearly.

Other losses

Conditions such as short-circuit current can also be factored into the overall power dissipation of a device. Techniques such as the α-power law MOS model consider these [Sak88]. In this thesis, however, these additional effects are considered to be part of either dynamic or static power, depending on their relationship to transistor switching activity.

Environment and workload affect silicon speed

Transistor switching speed increases in an approximately linear relationship to voltage whereas speed's relationship with temperature is feature size dependent. However, higher voltages result in greater dynamic and static power dissipation, and so the relationships between design thresholds, workload, speed, voltage and temperature are not always straightforward. For example, for larger feature sizes of 65 nm or more, the relationship between temperature and threshold voltage can typically be represented linearly, but the static current leakage has an exponential relationship with temperature [WA12]. Sub-65 nm exhibits an inversion in the temperature-speed relationship [KK06].

Processor temperature may be influenced by the ambient temperature of the operating environment, but also by the workload run upon it, as this will increase energy consumption and thus power dissipated as heat.

In order to provide a reasonable expectation of safety in a voltage tuned chip, its speed should either be constantly monitored, or if this is not possible, it should be measured during a period of slowest silicon performance. Inadequate monitoring or profiling could lead to an environmental change triggering a fault, or simply sub-optimal energy usage.

Summary

This section has demonstrated that energy saving is a multi-dimensional problem, where any one effort to reduce energy may, inadvertently increase it in some other way. Providing visibility of this is therefore essential if any form of design space exploration, at the hardware or software level, is to be effective.

4.2.4. Racing to idle in a real-time system

In a general purpose system, DVFS can be used as part of a *race to idle* strategy, where the device voltage and frequency can be aggressively scaled back upon completion of the current task, significantly reducing power dissipation. It may even be possible to turn-off certain components through power gating, removing static leakage as well.

In an embedded real time system, hard deadlines and responsiveness constraints can work against this strategy.

In a real time application, for a given block of code there exists a minimum timing constraint t between its endpoints. Given the number of cycles c needed to execute the block, the minimum frequency at which the block can operate and meet those constraints is:

$$F = \frac{c}{t} \tag{4.4}$$

In a system involving external I/O, consider the entry point to a block as the receipt of a stimulus (i.e. an interrupt or event) and the exit point some after time t is a response to that stimulus. If the I/O activity is not monotonic, the system must always preserve a sufficient level of readiness to respond within time t.

If DVFS is applied within idle periods, such that F is lower than required to satisfy t, then the event triggered by the I/O stimulus will require F to then be raised, either automatically be the hardware, or in software. In either case, this takes time, during which either the clock is halted [Int03b, p. 31], or continues to run at the slower frequency. In response to this, the active period may require a new, higher frequency to be used, in order to complete the code within t.

MII Ethernet receiver

To illustrate the above problem, an example of this is given in the form of a physical layer Media Independent Interface (MII) to an Ethernet receiver. The interface has a strict timing requirement and must meet it to avoid corrupting an incoming packet. The XMOS XS1-L architecture, which is explained in more detail in Chapter 5, is used as the case study processor for this example.

A 100 Megabits per second (Mbps) Ethernet frame is formed of a preamble, Start of Packet (SoP) token, up to 1500 bytes of data followed by 4 bytes of Cyclic Redundancy Check (CRC). There is a minimum inter-frame gap of 920 ns. The preamble takes 600 ns. Data is delivered via a port `RXD` that is 4-bits wide, receiving a nibble every 40 ns. With a buffered input on an XS1-L, 32-bits can be read at a time, thus allowing 320 ns to process each word. A separate input `RXDV` from the MII indicates that data is being received.

If a slow clock frequency is used during the inter-frame gap, then there is a 600 ns period from detecting `RXDV` changing, to being ready to receive the SoP and subsequent data.

If 16 instructions are required to input and process each word of the Ethernet frame, then a 200 MHz core clock is needed to satisfy the instruction rate needed by the receiving thread in the XS1-L, as per Eq. (5.2). Once `RXDV` goes low to process the end of the packet, let us assume that a further 8 instructions are needed before entering a lower frequency.

Assuming the best case delay in DVFS is a single instruction cycle, the lowest slow clock would be 6.67MHz (600nS period). When the mode switch latency is 29 cycles, the slow clock can be 193.33MHz. Beyond this, the slow clock would equal or need to be higher than the fast clock in order to satisfy response times and thus would be counter-productive. If the Ethernet interface is not 100 % utilised, then the inter-frame gaps may be larger. If the duty cycle of the Ethernet frames is known, then this can potentially be exploited to further save energy, by allowing limited periods of slower operation.

Figure 4.1 shows the trade-off between power saving, frequency reduction with a known duty cycle, and the DVFS mode switch latency. The mode switching latency dominates the ability to save energy. The steep edges on the surface plot are the points at which the switching latency requires a slow clock that uses a higher core voltage, reducing the potential power savings.

Online versus offline energy optimisation

It is possible to set DVFS scaling points offline, before programs run, or to determine them online during execution. The latter has the potential to be more flexible, in that certain properties that

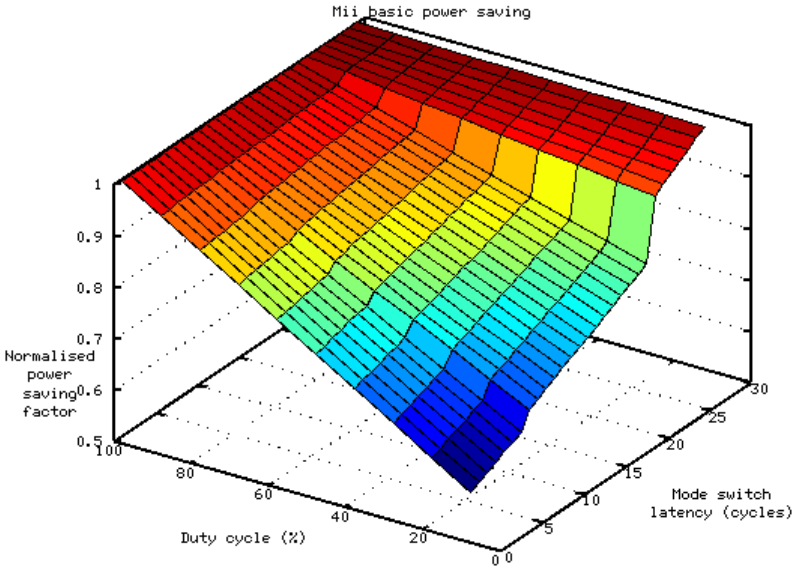


Figure 4.1: Percentage power saving obtained with varying Ethernet packet duty cycles and mode switch latencies.

might not be known statically, such as the actual ingress rate of Ethernet frames, can be used to guide parameter selection. However, performing these calculations online incurs an overhead, which may itself have a negative impact on energy consumption. This can be mitigated by combining both offline and online scheduling techniques [CLH09].

It is also possible to reverse the goals, instead controlling the quality of service in response to the energy available. Computational effort can be modulated in response to the energy available to a system [Yak11]. Appropriate decisions must be made as to how the application degrades if energy becomes scarce, and this requires online data from the hardware along with suitably adaptive software.

Summary

This subsection has demonstrated mostly hardware-oriented efforts that can be made to save power, working within the constraints that may be imposed upon an embedded system. Whilst DVFS can be beneficial, the timing constraints in an embedded system necessitate that changes in frequency and voltage be very fast.

Once again, the software becomes the focus of optimisation effort after the hardware's energy saving features cannot help any further. This lends further credence to the prioritisation of goals stated in § 4.1, where this thesis focuses on the software level, where the greatest potentials for savings can be made.

4.3. Can we sit back and let Moore's Law do the work?

The previous section demonstrated how hardware features such as DVFS have limits, particularly in embedded real time systems. However, the pragmatic software developer may choose to assume that the next generation of a hardware component can deliver sufficient improvements in energy consumption and/or performance, that spending effort in energy optimisation at the software level is without merit. This section argues against such a viewpoint.

The often cited *Moore's Law* [Moo65] is long-standing assertion relating to the progress made in microprocessor manufacturing over time. Moore observed that "the complexity for minimum component costs has increased at a rate of roughly a factor of two per year" and that this was likely to be a constant rate. This lead to the now popular interpretation of Moore's Law that states the number of transistors in a processor doubles every two years.

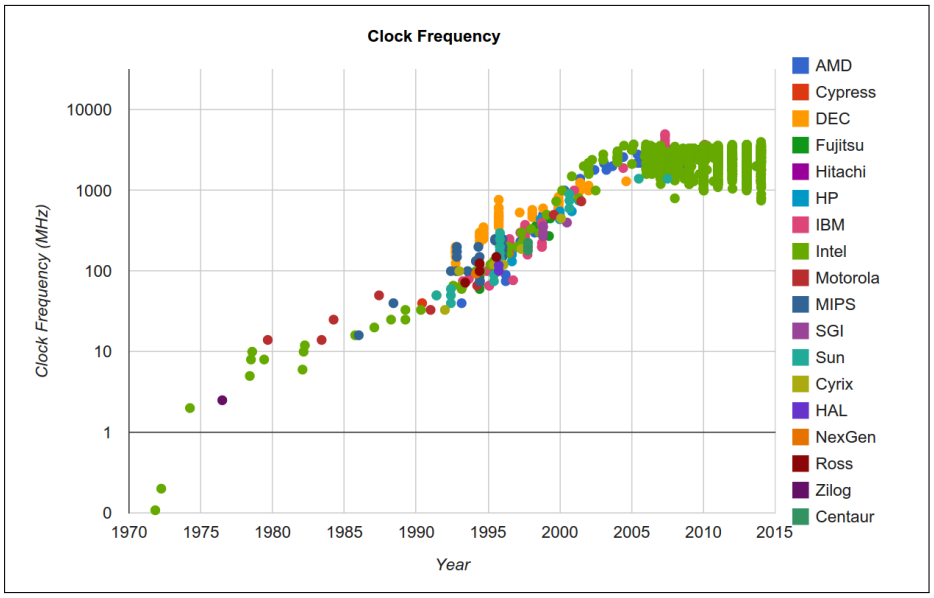


Figure 4.2: CPU frequencies since 1972. Generated by the Stanford CPU database [Sta12].

When viewed with some flexibility this remains the case in 2015, some fifty years after Moore's observations were first stated. Based on this continuing trend of smaller, more capable devices and the improvements in performance and energy efficiency that come as part of this, a plausible solution to energy efficiency might be to sit back and benefit from improvements to hardware. However, multiple factors conspire to limit the benefits of Moore's Law, or even negate them.

Other aspects of processor designs that previously benefited from growth in line with Moore's Law no longer do so. The most notable example is the near stall in device clock frequencies since 2005, with current International Technology Roadmap for Semiconductors (ITRS) reports observing low single-digit percentage increases in clock speeds year on year [Kah13]. This is caused by device operating voltages no longer reducing in line with Dennard's scaling observations [DGY74]. Transistor counts continue to increase, but frequency boosts are restricted by thermal design limits. Increased performance must now be extracted principally through multi-core or other forms of parallelism. This can easily be verified by examining the Stanford CPU DB [Dan+12; Sta12], wherein a plateau of CPU frequencies is clearly visible after 2005, as shown in Figure 4.2.

In addition to the above, the operating voltage of processors approaches a floor, as the difference between V_{th} and V_{dd} becomes very small. The benefits of reduced operating voltages was discussed in § 4.2.3. Work continues into silicon devices that can operate at Near-Threshold Voltage (NTV) [Kau+12] and STV [Zha+09].

Even continuing to increase the transistor count in devices, another parameter that is essential in manufacturing products is cost. New generations of technology node become prohibitively expensive as a result [OrB14b], except for larger organisations and large product runs [OrB14a].

In the software realm, Wirth's Law [Wir95] argues that as systems grow in size and capability, the software running upon them grows in complexity, with a slow-down associated with that growth. May's Law refines this argument with respect to Moore's Law's two-year cycle, asserting that "software efficiency halves every 18 months, compensating for Moore's Law" [Ead11]. In parallel programming, this is particularly significant, because the impact of slow sequential parts of a program can lead to profound inefficiency at the behest of Amdahl's Law [Amd67].

From this smörgåsbord of laws and observations comes a strong motivation to specifically target software when improving energy efficiency, or indeed any kind of efficiency in a system. Without doing so, many of the potential benefits made possible through hardware improvements are lost, particularly in the new era of parallelism.

4.4. Efficiency through event-driven paradigms

All computing systems must perform some form of I/O. This requires some form of data exchange between a processor and an external device. In many cases, the availability of data varies based on external, unpredictable parameters, such as the speed at which a user types or the network delay before the receipt of a new Ethernet frame. *Spending time waiting* for these unpredictable time periods to lapse is *wasting energy waiting*. In concurrent programs, delays may be incurred from waiting for another thread to reach a particular state. In either the case of concurrency or I/O, some form synchronisation must be performed.

Historically, software has often adopted a busy-waiting approach to delays. Various algorithms for synchronisation exist [GT90]. Typically they may involve a spinlock or some other form of *busy loop*. A superior alternative to this is to adopt an event-driven paradigm, where a wait condition is specified and an event vector followed upon the satisfaction of that condition. Prior to the event condition, the processor can execute other software, such as additional tasks if a multi-tasking OS is used, or the processor can enter a lower power state where no instructions are executed. An outline of the programming styles of these two methods is shown in Listing 4.1 and 4.2.

```
1  void lockedTask(lock_t l,
2      resource_t r)
3  {
4      do {
5          // Spin, repeatedly
6          // testing lock
7      } while(tryLock(l) == 0);
8      performActivity(r);
9      unlock(l);
10 }
```

Listing 4.1: Spinlock loop.

```
1  void lockedTask(lock_t l,
2      resource_t r)
3  {
4      waitLock(l); //Blocks thread
5      performActivity(r);
6      unlock(l);
7
8
9
10 }
```

Listing 4.2: Event-driven wait.

In order for a system to reap energy savings from event driven paradigms, both the software and hardware must support it. Without hardware support for interrupts and associated conditions, the best effort a kernel or application can do is to emulate the checking of these conditions in what effectively becomes another spinlock. Thus, events are abstracted into busy-waiting loops. In Listing 4.2, line 4, it is assumed that the blocking of the thread in order to wait for the acquisition of a lock will result in de-scheduling and therefore either idle time or the execution of another thread. However, it may be that the implementation of `waitLock` elaborates to lines 4–7 of Listing 4.1.

In a review of synchronisation methods for MPSoCs [GLP07], the sleep based methods consume significantly less energy than all others. These sleep based synchronisation algorithms either apply DVFS in idle periods between checks, or obtain notification from hardware, reducing activity even further.

An RTOS may provide a framework for writing tasks that make use of interrupts. The RTOS itself may use timer interrupts to allow it to enforce task scheduling without tasks needing to manually yield or make any kind of system call. The same is true of the kernel in a general purpose OS, although the exposure of interrupt events is typically kept to device drivers, with software libraries providing further abstractions between the hardware and user-space applications.

The XMOS XS1 architecture is built around events. In the XS1 lexicon, an event is analogous to an interrupt without state-saving, where the handling of an event is assumed to be the intended outcome of a thread. This is contrary to an interrupt, which assumes that the thread will resume from its previous position after the ISR is complete, or at some other point thereafter, if kernel scheduling takes place. This is examined in more detail in Chapter 5.

4.5. Summary

This chapter has introduced a set of problems relating to the goal of saving energy in an embedded system. These problems are typically multi-dimensional, and the ideal outcome constrained with respect to these dimensions.

Strategies such as DVFS have clear benefits in certain contexts. However, selecting the correct DVFS parameters, particularly in a real-time system, is non-trivial.

A common failing in energy saving efforts is to push the problem from one dimension into another. For example, aggressively optimising one part of the software may place an additional burden on another. Similarly, lack of awareness of timing constraints may preclude any benefits from being obtained. The prioritisation of effort is particularly important. In § 4.1 it was shown that from a software level, the algorithm is critically important to energy consumption, and effort such as using software to better exploit low-level hardware energy saving features, is wasted if the algorithms used in a piece of software do not map well onto the underlying hardware.

An objective of this thesis is to further the state of the art in awareness of energy consumption in MTMC embedded systems. By doing this, the developer is empowered to explore energy saving options such as those described in this chapter. Most importantly, the developer of embedded systems software is given sufficient visibility to identify where energy optimisation effort would be wasted, allow them to favour areas with greater potential for improvements (low hanging fruit). This helps the developer establish a good balance between potentially numerous complex trade-offs.

5. A multi-threaded, multi-core embedded system

This is the final chapter in Part I of this thesis. It details both the XS1 processor architecture at the center of the modelling effort Part II, along with the *Swallow* platform, an assembly of these processors into a networked, Multi-Threaded and Multi-Core (MTMC) embedded system, which is used to extend the modelling for networked, multi-core embedded software.

This chapter begins with a discussion of why the XS1-L is used as the focus of this work, in response to the thesis statements put forward in § 1.1.

Motivation of selection

To explore the modelling of software running on a MTMC embedded system, a suitable hardware platform is required. The XMOS XS1-L processor meets this need for a number of reasons:

- Each XS1-L core is hardware multi-threaded.
- XS1-L processors can be interconnected to form a multi-core system.
- Energy efficiency is a consideration of the architecture, with event-driven paradigms and efficient multi-threading built into the ISA.
- Software written for these processors is typically run bare-metal (without an OS), in the relatively low level languages C and XC.
- The software has a large amount of direct control over the hardware's behaviour, due to the processor's target market.

In addition, the architecture has a number of characteristics that make it unique and worthy of exploration with respect to energy modelling, complementing techniques applied to existing architectures. In particular:

- Time-deterministic instruction execution.
- No cache hierarchy.
- Single-cycle memory.
- A focus on message passing rather than shared-memory, both in software abstraction and hardware implementation.
- I/O control and communication are directly implemented in the ISA.

Motivated by these characteristics, this chapter details these and other relevant parts of the processor's ISA, micro-architecture, and physical properties in § 5.1. These details form essential background for the single-core, multi-threaded profiling and modelling contributions made in Chapters 6 and 7.

The Swallow project is then introduced in § 5.2, a platform with which grids of XS1-L chips can be connected and programmed. A significant amount of research effort was placed into making these boards useful for MTMC research. These multi-core implementation details form the understanding required for the contributions made in Chapters 8 and 9.

5.1. The XS1-L processor family

The XMOS XS1-L processor [May+08] is an embedded processor implementing the XS1 ISA that allows hardware interfaces to be written in parallel software. The execution of software is time-deterministic and the instruction set places I/O hardware extremely close to the software. This

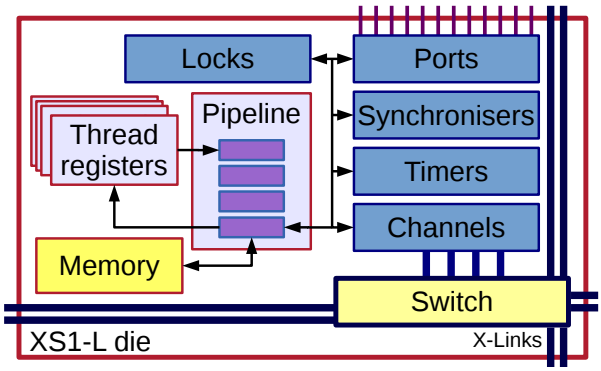


Figure 5.1: Block diagram of XS1 architecture, showing pipeline, per-thread register banks, peripherals and network resources.

provides timing guarantees and advanced General Purpose Input/Output (GPIO) capabilities that mean components such as MII Ethernet, Serial Peripheral Interface (SPI), I^2C and Universal Serial Bus (USB) can be expressed as software rather than fixed hardware units. The application space of this processor is therefore between a traditional programmable microprocessor and an FPGA.

Energy efficiency is sought both in the hardware and programming model by introducing *event driven* paradigms, previously discussed in §§ 2.1.4 and 4.4, whereby a thread may defer its own execution until the hardware observes a particular event, such as a change in state of an I/O pin. The hardware scheduler eliminates the need for polling loops and similarly, due to multi-threading, interrupt service routines are not required.

The architecture also features a custom asynchronous interconnect that can be used to assemble networks of XS1-L processors to distribute programs over many cores, with communication taking place over two- or five-wire *X-Links*. This section focuses on multi-threaded operation of a single core. The network is explained in more detail in § 5.2.

5.1.1. The XS1-L multi-threaded pipeline

To bring the software as close to the physical interfaces as possible, the XS1 ISA manages threads in hardware, with machine instructions and hardware resources dedicated to thread creation, synchronisation and destruction [May09b]. In addition, each thread has its own bank of registers, containing a program counter, general purpose and special purpose registers. This removes the overhead of an OS and allows threads to be created in tens of clock cycles, but places a hard-limit on the number of threads that can exist on the processor at any one time.

A block-level view of the XS1-L implementation is shown in Figure 5.1. In the XS1-L, up to eight threads are supported. These eight threads are executed in a round-robin fashion through a four-stage pipeline. The pipeline avoids data hazards and dependencies by allowing only a single instruction per thread to be active within the pipeline at any one time. As such, the pipeline is only full if four or more threads are active. If there are fewer than four threads able to run, then there will be clock cycles in which pipeline stages are inactive. This makes the micro-architecture relatively simple to reason about, but requires at least four active threads in order to use the full computational power of the processor. When more than four threads are active, maximum pipeline throughput is maintained, but compute time is divided between the active threads.

Calculating instruction throughput

With a 4-stage pipeline structure, the Instructions Per Second computed by the processor, IPS_p, in relation to the core frequency, F, is simply $\text{IPS}_p = F$, in the case where the pipeline is full. It can be expressed for any number of threads, N_t, as shown in Equation 5.1.

$$\text{IPS}_p = F\frac{\min(4, N_t)}{4} \tag{5.1}$$

The instruction frequency of an individual thread, IPS_t, can similarly be expressed as:

$$\mathrm{IPS}_t = \frac{F}{\max(4, N_t)} \tag{5.2}$$

An XS1-L processor typically operates at 400 MHz or 500 MHz, the latter providing a total instruction throughput of 500 Million Instructions Per Second (MIPS) or at most 125 MIPS per thread.

Context switching is free in time, but not in energy

By virtue of hardware threads and a four stage pipeline, the processor effectively performs a "context switch" on every clock cycle, assuming there are sufficient active threads. In terms of performance, this delivers significant benefits to multi-threaded programming in a single-core by removing large overheads. However, from an energy perspective this creates interesting behaviours.

Firstly, with each clock cycle comes a completely new state into the pipeline — the proceeding instruction and data will be from the next thread in the schedule. This introduces an energy cost through switching in the control and data logic of the processor. Secondly, these context switches on every cycle change the energy characteristics of the pipeline in a way that existing ISA level energy models do not account for.

The fetch-noop and event-noop

The XS1-L pipeline structure avoids data hazards and other behaviours that would trigger pipeline stalls and flushes in other architectures. However, there are limited scenarios in which a delay in execution may still be incurred, in the form of a "fetch no-op" (`FNOP`).

The `FNOP` occurs when a thread's instruction buffer does not contain a full instruction ready to be issued. The conditions that may lead to starvation of the instruction buffer are explained in [May09a] and summarised here:

- Multiple sequential memory accesses in a thread prevent fetches, because fetches are performed in the memory stage of execution.
- Branch operations flush the instruction before, then fetch the branch target when word-aligned.
- Instructions are 16-bit aligned in memory, but are 16- or 32-bits in length.

If the above properties conspire to starve the instruction buffer, then a `FNOP` is issued. `FNOP`s can be avoided by scheduling instructions to avoid runs of memory operations in a thread and by aligning 32-bit instructions that are branch targets to 32-bit boundaries. If either of these is impossible or undesirable due to the other overheads that may be incurred, then the conditions that result in an `FNOP`s are sufficiently well defined that they can be determined statically, as is the case in the XMOS Timing Analyzer [XMO10].

If an event or interrupt vector is followed by the processor, then the instruction buffer for the affected thread must be flushed in a similar way, leading to a dedicated fetch, in this case termed an *event no-op*.

These implicit no-ops, while simple to reason about, can have an important impact on the time and also energy consumed by a program. Inclusion of these behaviours is therefore necessary in simulation and energy modelling of the processor.

5.1.2. Instruction set

The XS1-L implements the XS1 ISA. This ISA is best described as a Reduced Instruction Set Computer (RISC) construction, containing 203 instructions in total. Along with a set of typical arithmetic, logic, memory and branch operations, as described in the ISA manual [May09b], additional groups of instructions provide simple DSP, peripheral component control, thread management, event handling and communication. This subsection elaborates on these ISA features and their significance with respect to the unique requirements and opportunities they create when modelling the energy consumption of software running on the device.

Directly accessed peripheral blocks

In a conventional computer architecture, peripheral components such as timers, device interfaces and Direct Memory Access (DMA) units are *memory mapped*. A memory mapped peripheral occupies a section of the processor's address space. Within that address space reside registers for control, status and data relating to that peripheral. The processor interacts with a peripheral by performing memory reads and writes to these locations [Rei99].

In the XS1 ISA, peripheral components are accessed directly with a set of *resource instructions*. These allow peripherals to be allocated to a thread, controlled, and data read from/written to them. This separates activities that can be considered I/O from memory access in the instruction set as well as the memory hierarchy. Other ISAs, such as x86 [Int11, pp.115,176], distinguish from memory and I/O in the instruction set through I/O specific instructions. However, I/O and memory still share the address bus.

In XS1-L, the following resources are made available:

- Thread synchronisers
- Communication channels
- Timers
- I/O ports
- Locks

ISA operations performed in relation to these instructions include data input, data output and configuration. The exact behaviour is resource specific. For example, communication channels need to be configured with a destination address before use, whereas locks are a simple resource which use data in and out instructions to obtain and release the lock, with no other configuration required.

All resources can be associated with interrupt or event vectors, causing a context switch or jump upon the resource triggering some condition. For example, a channel resource would trigger an event upon the availability of new data from a remote channel end. A timer could trigger an event in a thread that has set a comparison condition against the timer, in order to then perform some action at a specified time.

A thread can be *de-scheduled* when waiting for an event, meaning that it is no longer executing instructions and thus not consuming any time within the execution pipeline. Alternatively, a thread may continue running instructions until an event takes place. In the latter case, interrupts are likely more useful then pure events, as the context of the thread is variable, thus some state should be saved.

Hardware thread management

The scheduling of threads is handled in hardware. There is no requirement for a RTOS to be used to manage threads. The ISA provides mechanisms for thread handling through a series of `TINIT` instructions, which can be used to initialise the program counter, stack pointer and other pointers of the target thread's register file.

Threads can be initialised as unsynchronised, or associated with a synchroniser resource to allow barrier synchronisation of groups of threads. The XS1-L supports up to eight threads, which share time round-robin in the previously described four stage pipeline.

An allocated thread can be in either a running state, where instructions from that thread are issued into the pipeline, or de-scheduled against a condition. The conditions against which a thread waits are typically resource-driven, for example waiting for the expiration of a timer, or the arrival of data on a port.

Given that scheduling is implemented in hardware, there is no need to check the status of a thread. This combines well with the event and interrupt system of the processor. When an event occurs that will allow the thread to be scheduled, the hardware scheduler will take action. Busy-waiting loops can therefore be avoided.

Communication channels

The XS1 architecture uses channel communication for the exchange of data between threads, modelled upon the CSP formalisation [Hoa78]. Channel communication is included in XC, a custom C dialect created by XMOS. It is also present at the ISA and hardware level.

An XS1-L core has 32 *channel end* resources that can be allocated to threads. Each channel end has an address, identified as a composition of its network node ID, local channel end ID and resource type. The bit-wise construction is shown in Eq. (5.3). To send a message over a channel end, a destination address must be specified with a `SETD` instruction against the local channel end. All `OUT` instructions using that channel end will then be sent to the specified channel end. A simplified sequence of these instructions is shown in Listing 5.1 and 5.2, where a single word (the sender's own channel end ID, in this case), is sent to a receiving channel end.

$$\begin{aligned} \text{ID}[31:16] &= \text{Node ID} \\ \text{ID}[15:8] &= \text{Channel end ID} \\ \text{ID}[7:0] &= \texttt{0x2} \end{aligned} \tag{5.3}$$

```
1 getr r0,2        # Get chanend
2 ldw  r1,cp[0]    # Load dst
3 setd res[r0],r1  # Set dst
4 out  res[r0],r0  # TX word
```

Listing 5.1: Sending on a channel.

```
1 getr r0,2        # Get chanend
2 ldw  r1,cp[0]    # Load dst
3 setd res[r0],r1  # Set dst
4 in   r0,res[r0]  # RX word
```

Listing 5.2: Receiving on a channel.

A channel end will receive data from anywhere that addresses it. Thus, at the architecture level, many-to-one communication is possible, but one-to-many multicast or broadcast is not. At a higher levels of abstraction in XC, channels are expressed purely as point-to-point communication, without the ability to change the destination address dynamically. Further, although the architecture can also permit core-local message passing through shared memory, the original version of XC does not support this, due to strict parallel memory usage rules.

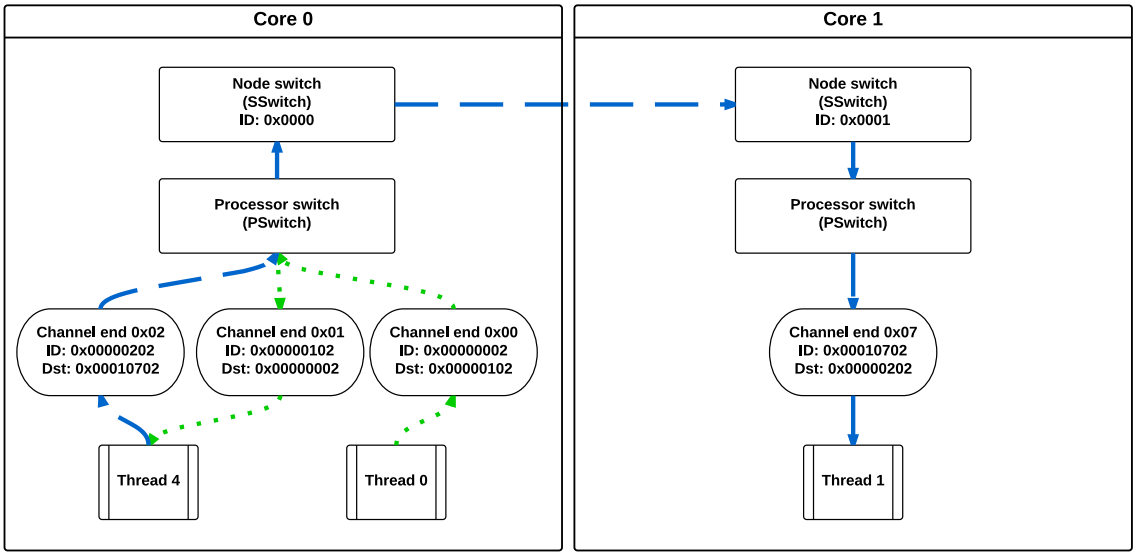


Figure 5.2: Channel communication in the XS1 ISA. Both core-local (green, dotted) and multi-core (blue, dashed) communication is shown, between two pairs of channel ends allocated across three threads in total.

With core-local channel communication, node IDs will be the same for source and destination. A data rate of 2 Gigabits per second (Gbps) can be achieved locally with a 400 MHz core clock. Multi-core communication uses the node ID to route messages to the correct core and is bandwidth limited

by the interconnect. Figure 5.2 depicts channel how channel communication between threads and cores takes place. The network implementation is explained in more detail in § 5.2.

5.1.3. XS1 product and micro-architecture variants

A number of devices are based on the XS1-L micro-architecture. Not all variations are of interest to this work, as they introduce features that are not central to the research. In addition, the naming of devices and some architectural components has changed during the undertaking of this research. This work has adopted the original conventions wherever possible, for consistency.

Product naming conventions

The devices and names that may referenced in this thesis and their key differences are explained below.

XS1
: The ISA, shared by all the XMOS processors modelled in this work.

XS1-G
: A quad-core, 90 nm implementation of XS1. This processor is not actively used in this research.

XS1-L
: A single-core, 65-nm implementation of XS1, at the centre of this research.

XS1-L1 and XS1-L2
: XS1-L based devices, assembled into either single- or dual-core products in a single package. The XS1-L2 devices are used in Swallow.

XS1-SU1
: An XS1-L based device packaged with a USB Physical layer (PHY), Analog-to-Digital Converters (ADCs) and voltage controllers. The peripheral devices are accessed using the XS1's channel communication paradigms.

XS1-A8, A16, U8 and U16
: Single- and dual-core variations of the XS1-SU1, using the more modern XMOS naming scheme. Devices prefixed with "U" contain USB and analogue peripheral components, whereas "A" devices omit USB.

Architectural naming conventions

A *thread*, as defined in § 2.1.1 and the original XMOS terminology, is a *logical core* in the new terminology. A *core* is then termed a *tile*. The distinction between old and new styles can be made by observing that care is taken to refer to "logical cores", not simply "cores" in the new terminology. Further, the term threads is not used in the new style, and tiles is not used in the old.

The changes made to the naming conventions create some potential for confusion when cross-referencing material. However, in isolation, this thesis maintains consistency in its use of terms.

5.1.4. Summary

This section has given an outline of the features of the XS1-L, particularly those of interest to this research, as listed at the beginning of the chapter. The very tightly coupled hardware scheduled threads present a new challenge for ISA level energy modelling, whilst the channel communication, event- and resource-driven parts present new opportunities for analysis of programs in a MTMC context.

The examination and modelling of a single XS1-L core in Chapters 6 and 7 yields new insight into hardware energy characteristics and a new approach to energy modelling of embedded software. However, multiple such devices form a more complex and interesting subject of study. The next section describes a system of this nature.

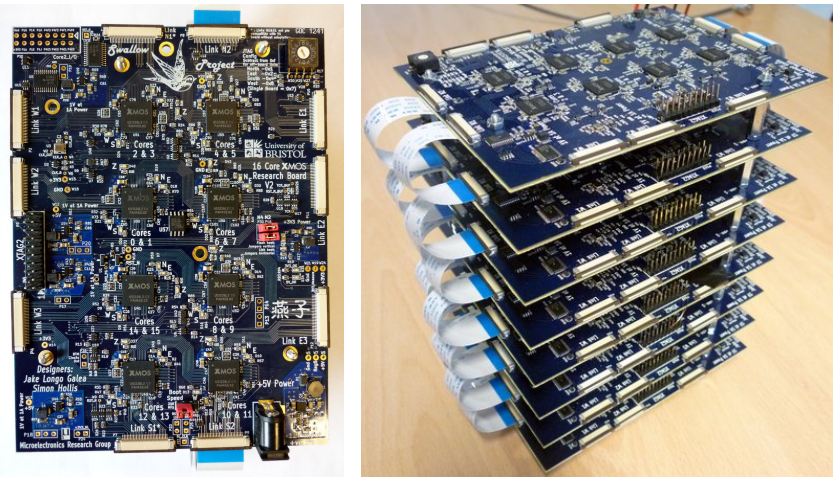
5.2. Swallow multi-core research platform

The Swallow multi-core research platform is a project established within the Microelectronics Research Group at the University of Bristol during the course of the research presented in this thesis. A significant amount of research and development effort was put into the tools and software for Swallow to ensure that it can serve as a platform for supporting the multi-core component of the multi-level energy model demonstrated in Chapter 9.

This chapter details the Swallow platform, its purpose in relation to this thesis and how it and the tools developed for it are used to further the research conducted herein. A more general description of the Swallow platform, in particular more detail on aspects not directly relevant to this thesis, can be found in [HK15].

5.2.1. System design

Swallow, pictured in Figure 5.3 is designed to allow a multi-core embedded system in the order of hundreds of cores to be assembled and used for a variety of experiments, in particular work focusing on multi-core task allocation, network utilisation and energy efficient multi-core computing. It achieves this with XMOS XS1 based hardware. As a result, it does not exploit emerging chip technologies such as large on-chip networks of many cores and 3D stacking of components. However, it does provide an experimental platform for exploring some of the considerations that must also be taken into account in devices that use networks to communicate.



(a) A single 16-core Swallow board. (b) A 1x8 board stack.

Figure 5.3: Photos of the Swallow platform.

The key components that make up the Swallow platform are as follows:

- XMOS XS1-L2 dual-core, 16-thread chip.
- Eight L2 processors assembled onto a single board, giving 16 cores per board.
- External link interfaces to allow multiple boards to be assembled both horizontally and vertically.
- Power measurement shunts designed into the board's various power supplies, with pin-out to allow measurement equipment to be coupled to the boards easily.
- Some I/O exposed to allow external interaction when the X-link network cannot be used.
- Support for *peripheral boards* that feature additional XS1 processors and provide peripherals such as additional DRAM and Ethernet connectivity.
- JTAG, flash and Ethernet based booting of cores, as well as JTAG debugging.

These will each be examined in more detail in the remainder of this section.

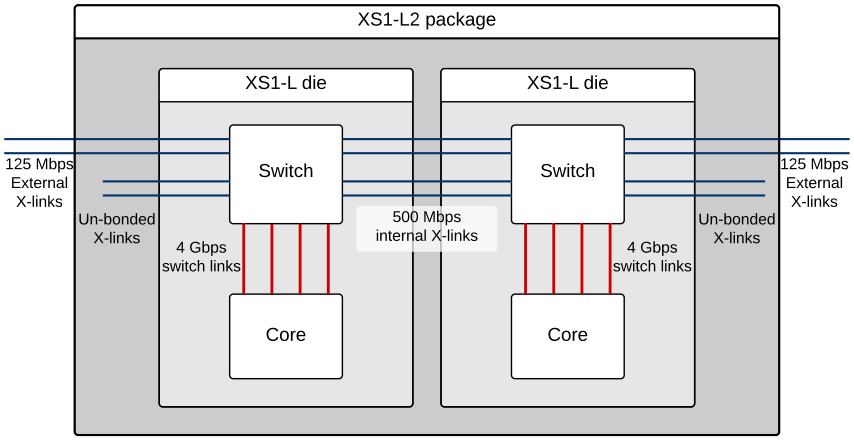


Figure 5.4: The XS1-L2 package and its relationship to the two L-series cores and switches contained within it. There are a total of 16 X-links, with four connected to pins on the package, two from each switch.

XS1-L2 processor

At the time of design, the XS1-L2 processor had the largest core count of any XMOS L-series processor. The G-series features 4 cores, but uses an older process technology, has a more restrictive network topology requirement, and has no Dynamic Frequency Scaling (DFS) capabilities. As such, the XS1-L2 was the best choice for achieving maximum core density, whilst enabling exploration of network utilisation and energy efficiency in current and future work.

The XS1-L2 is two L-series die assembled in a single package, a representation of which is depicted in Figure 5.4. Each die contains an XS1-L core and switch. The switch provides eight X-links, four of which are bonded to the switch of the neighbouring XS1-L within the package. Each switch also has two of the remaining X-Links bonded out onto external pins. A number of I/O ports are also bonded out. A single clock source and set of power supplies is shared with both cores via the package pin-out. The exact pin-out is described in the XS1-L2 datasheet [XMO12].

The pin-out presents some limitations with respect to efficiently laying out the chips and assembling multiple chips into a network. Due to each package containing two switches and the connectivity of those switches in relation to the package, assembling a mesh network using a north-south/east-west connection method would result in a sub-optimal maximum number of hops for any given assembly of such chips. It was therefore necessary to connect north-west/south-east. This is described in more detail in the Swallow technical report [HK15]. The resultant network and routing strategy is described in § 5.2.2 of this thesis.

Eight chip Swallow board

Each Swallow board contains eight dual-core XS1-L2 processors. Vertical pairs of chips are powered from separate voltage controllers for V_{core}, with a global V_{io} for all chips and other I/O components such as Light Emitting Diodes (LEDs).

All four external links of each XS1-L2 chip are used and are either connected to a neighbouring chip, or an external connector for off-board communication. The network topology is described in more detail in § 5.2.2.

External interfaces

Ribbon connectors provide off-board transit for X-Link, I/O and JTAG signalling. The connectors are not homogeneous in pin-out, restricting how boards can be connected. Swallow boards must be aligned when connected, such that the top-left connector of one interfaces to the top-right of another, and so on. Further, peripheral boards are only compatible with north and south connectors, not any of the east or west connectors.

Power measurement

Shunt resistors are included on the 5 V, 3.30 V and each of the 1 V DC-DC power supplies, so that the current supplied by them can be monitored when appropriate hardware is attached, such as an INA219 measurement chip [Tex11]. The 5 V and 3.30 V supplies have a pin-out that allows a measurement board to be attached above them.

I/O

Due to the prolific X-link usage and chip density per board, there are few spare I/O ports and many that can be routed to a connector. However, there are still some restricted I/O capabilities:

- Six 1-bit and two 4-bit ports from `core 0` on the top-left chip, wired to a 2x8 header at the corner of the board. This can be used for GPIO.
- Three 1-bit ports on `core 6`, connected to a 64 Megabit (Mb) SPI flash chip on the Swallow board. This can be used for persistent data storage.
- Four 1-bit ports connected to `core 10` This is intended to connect to an energy measurement board, providing either sufficient I/O for the I^2C interfaces to the measurement chips, or as a simple interface to the device controlling the measurement board, in order to provide triggering and synchronisation of measurements.

Additional I/O is possible if external X-Links are re-purposed; the package-accessible X-Links on the XS1-L2 are multiplexed with various I/O ports [XMO12]. However, a more flexible approach is to connect an additional XS1 device over an X-Link that is designed to serve as an I/O controller.

Peripheral boards

The Swallow grid can be supported by peripheral devices consisting of one or more additional XS1 chips and additional I/O components. Currently, one such peripheral board exists.

The peripheral board features a single-core XS1-L1, controlling 32 Megabytes (MBs) of DRAM and has a connector for interfacing with *Slicekit* peripherals. XMOS Slicekits are a system of modular board and peripherals that can be connected in various ways [XMO15]. In the case of the Swallow peripheral board, the connector is intended to be used with an XMOS *Ethernet slice*, which is a Slicekit compatible network adapter with PHY chip and RJ-45 connection.

The XS1-L1 on the board acts as an interface to the DRAM and Ethernet and can communicate with the grid using channel communication. As such it can serve as a network bridge to allow data to flow into and out of the grid over Ethernet and also as a volatile memory store. This allows the grid to access significantly more memory than the 64 Kilobytes (KBs) SRAM of each of the XS1-L cores. It is also possible to load program images onto the grid over Ethernet via TFTP, which is significantly faster than JTAG, particularly for large numbers of cores, where the JTAG chain size increases load times quadratically.

JTAG

JTAG provides a method of programming and debugging devices by forming a chain of Test Access Ports (TAPs) that can be read and written serially [Rob94]. The performance of JTAG is limited by the length of the chain and the maximum clock rate at which the slowest TAP can operate.

A single XS1-L device contains four TAPs, two for the core and two for the switch [May+08, p. 31]. On a swallow board there are 16 XS1-L devices, forming a chain of 64 TAPs on a single board. The chain is formed along the chips in a clock-wise fashion, entering at the leftmost chip on the second row. Figure 5.5 gives a graphical representation of how the chain is formed, including optional connectivity to other boards and the debug device. Control of external JTAG connections is made via a 4-bit rotary switch, which drives the select inputs on a set of multiplexers.

When multiple boards are connected, the chain extends horizontally between boards, with vertical chaining along the leftmost set of boards. A script was written as part of this work in order to generate both the network configuration and correct JTAG chain ordering for arbitrary board arrangements [Ker14].

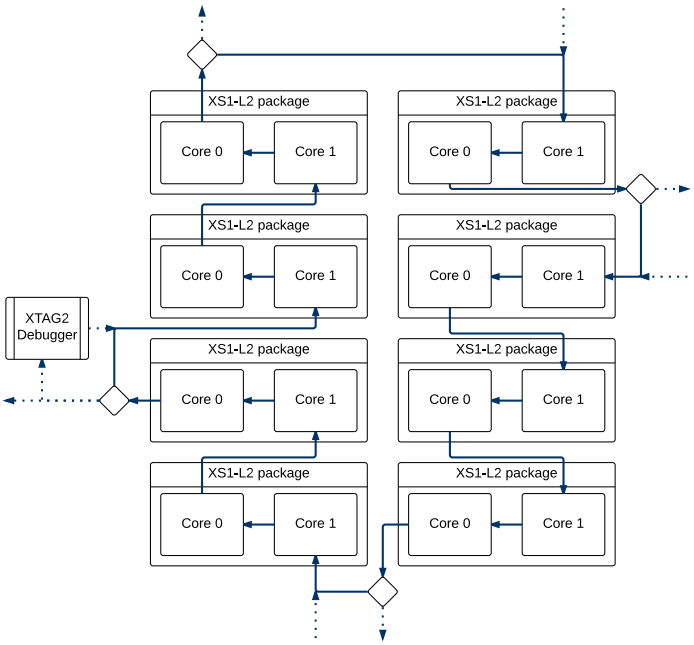


Figure 5.5: JTAG chain of a single Swallow board, showing switching points to form chains with additional boards.

Any JTAG read or write operation must be shifted through the chain by a clock, with sufficient clock cycles provided to allow any response data from TAPs to also be shifted along the chain. The approximate time, t_{msg}, to send an m-bit message along a JTAG chain is therefore constrained by its clock frequency, F, and chain length, c, described in Eq. (5.4). The achievable throughput of multiple messages is dependent on the response time of the TAPs and the size of the response that they give, assuming that the response needs to be interpreted before sending another message.

$$t_{\text{msg}} \approx \frac{m \times c}{F} \tag{5.4}$$

5.2.2. Network implementation

The Swallow network forms a two-layer *unwoven lattice* structure. There are three dimensions to the network: *horizontal* and *vertical*, with respect to a board and its neighbours, and *layer*, with respect to the cores within a single chip. Figure 5.6 visualises the connectivity formed by this topology.

The connectivity of the chips is such that each core only has the freedom to communicate in two of the three available dimensions, one of which is always the layer dimension. This forms two layers, one in which horizontal communication is possible, and one in which vertical communication takes place. The first core in each chip is connected to the vertical layer, whilst the second is connected to the horizontal layer.

XS1 communication principles

When multi-core channel communication is performed in XS1, X-links are used to transmit and receive data. The links use a credit-based flow control mechanism [May+08, pp. 12–13] to block transmission if upstream buffers are full. Messages are transmitted on the wire as one-byte tokens, although ISA allows transmission in single tokens via `outt` and `int`, or four-byte words with `out` and `in`. A message begins with a three token destination address header, which is automatically prepended to the first token emitted from a channel end by an `out` or `outt` instruction.

If the destination address is non-local, then the local switch begins to receive the message over an internal link to the processor core. The most significant 16 bits of the destination address are then

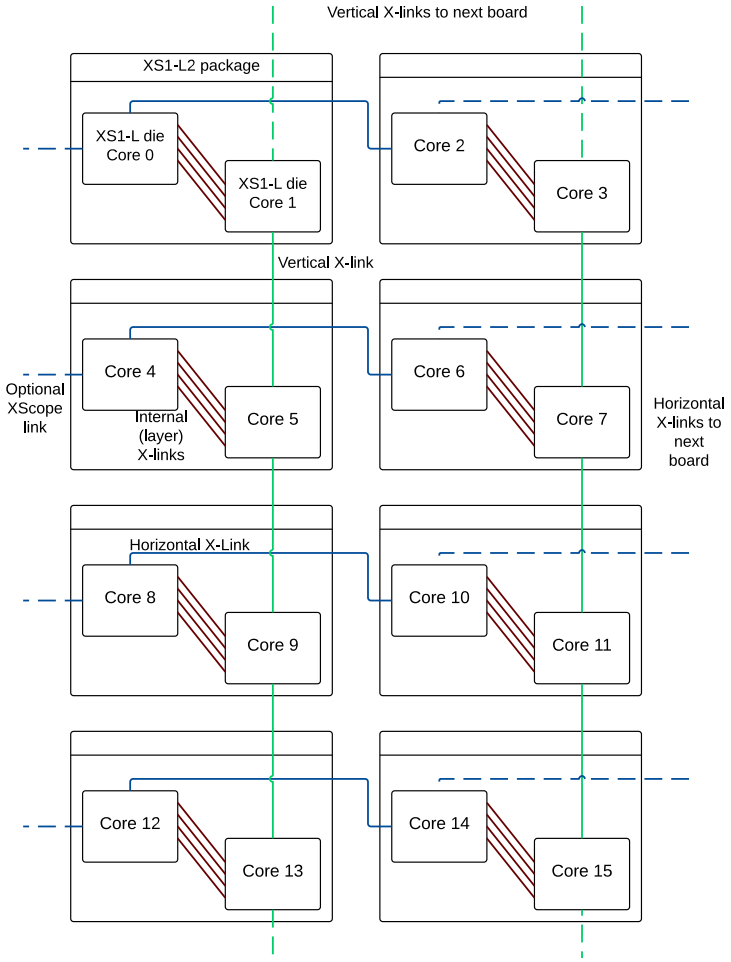


Figure 5.6: Swallow network topology, with on-chip links providing layer transitions, and each core having either vertical or horizontal external connectivity, similar to a unwoven lattice on a pie. The second horizontal X-link on the left can connect to another board, or optionally an XMOS XTAG2 for faster debug output than JTAG.

used to determine the X-link over which the message should be forwarded. A lookup is performed against the position of the most significant destination bit that is different to the local node ID, where the position number determines which of the device's X-links will then be used. Switches receiving a message over an X-link perform the same activity. If the top 16 bits match the local node ID, then the remaining header bits (the channel ID) are forwarded to the local core, along with the rest of the message.

Once a message has begun and the header sent, any X-links along the route are reserved in the direction of communication. A closing control token must be sent along the route in order to free these links for use by other messages. Either a `PAUSE` or an `END` token can be sent to achieve this, where the former is discarded by the destination switch and the latter propagated to the destination channel end.

This wormhole routing strategy allows both packeted messages and dedicated communication channels to be used. In the latter case, no `PAUSE` or `END` tokens are ever issued between a pair of channel ends, leaving the route between them permanently allocated to them. Contention can be resolved by using multiple links between nodes (such as the four links between the two cores in the XS1-L2 chip) and by sending information in packets, where the overhead of sending the three-byte header should be considered, particularly for small packets.

Routing table				
Bit	15	14	···	0
Use link	0	0	···	1
Example				
Node ID	0	**1**	···	1
Destination ID	0	**0**	···	1
Forwarding link	**0**			

Table 5.1: XS1-L routing table example, where the second most significant bit of an incoming header is different to the local node ID. A lookup against the routing table indicates that link 0 will be used to forward the message.

Dimension-order routing over layers

Dimension-order, or e-cube routing allows deadlock-free routing over N-dimensional network structures [SB77]. The direction of travel for communication is done in a pre-determined order. For example, in a 2D grid, a valid dimension-order routing strategy is to always move to the correct horizontal position on the grid first, then to the correct vertical position, at which point the destination has been reached. In this routing strategy, the request and response may take different paths if their locations differ in more than one dimension. This avoids deadlock by preventing cycles between groups of communicating nodes.

In the case of Swallow, nodes do not have connectivity to all dimensions. A best-effort is achieved by applying 2D dimension-order routing, where vertical positions are resolved first, followed by horizontal. When resolving the vertical position, if a node is on the horizontal plane, the message will first pass to the vertical layer. For the horizontal stage, the message will reach the horizontally routed node closest to the destination, then pass along to the node in the vertical layer if necessary. The result is that at most two layer transitions may be required (the first hop and/or the last hop), but vertical and horizontal traversals happen in dimension-order.

The routing strategy is governed by the switch configuration as defined earlier in § 5.2.2. A configuration for any $m \times n$ configuration of Swallow boards can be generated by a tool developed during the course of this thesis [Ker14].

Network speed and width

On-chip, each core has four links to its neighbour, with a maximum data rate for a link of 500 Mbps, giving an on-chip bisection bandwidth [HP06] of 2 Gbps. Between chips, there are single links vertically and horizontally. This extends to other boards. The link speed is also 500 Mbps maximum, but due to wire lengths it is typically a quarter of this in order to provide stability, although this can be tuned. The bisection bandwidth of a single board, with 125 Mbps links, is 250 Mbps. Bisecting a grid of boards horizontally (there are half as many vertical links has horizontal links per board), the bandwidth, b is related to the number of boards horizontally, w, and link speed, l, in Eq. (5.5).

$$2wl = b \text{ bps} \tag{5.5}$$

Link speeds are configurable either at compile time through an XN file, or dynamically through configuration commands to the relevant switches. The five-wire links used in Swallow transmit two bits per symbol and therefore four symbols per token. The transmit time of a token is $3T_s + T_t$, where T_s is the inter-symbol delay and T_t an inter-token delay. These delays are relative to the switch clock, which is typically either 400 MHz or 500 MHz and is usually the same as the core clock. The minimum delay parameters are $T_s = 2$, $T_t = 1$. Both delays are 11-bit values, allowing for link speeds significantly lower than the switch clock frequency.

Name	Load time	Size	Debug	Notes
JTAG	Slow	Limited	Reasonable	Partial network (straight line), core numbering in debugger can be unintuitive. No more than 128 cores.
JTAGv2	Slow	Limited	Good	Full network, logical core numbering. Still limited to 128 core debug. Requires v13.0 of XMOS tools.
Etherboot [Ker12b]	Fast	Unlimited	Poor	No debug symbols; assembly only. 128 core debug limit. Uses X-Link network to boot 2 orders of magnitude faster than JTAG.*
Etherboot + JTAG	Medium	Limited	Reasonable	Full network, logical core numbering and debug. Code loaded over Ethernet then debugging via JTAG.
*Video of boot process: `https://www.youtube.com/watch?v=kUo11tTeYK0`				

Table 5.2: Swallow boot methods developed during the course of this research, each of which has trade-offs to consider. The majority of work contributing to this thesis is done via the JTAGv2 method.

5.2.3. Compiling & loading software for Swallow

Several techniques have been developed for loading software onto Swallow over the course of this research, some of which have become possible due to improvements to the vendor's toolchain, whilst others have required more customized implementations.

The *JTAGv2* boot method (Table 5.2) is used for the profiling and testing performed in Chapter 9, as this provides an appropriate level of debug capabilities on a full network implementation which can also be simulated.

5.2.4. Summary of Swallow

The statements of this thesis, made in § 1.1, demand a multi-core system of embedded multi-threaded processors in order to perform the desired research. This section has described the Swallow platform, a system which serves this purpose.

The Swallow platform introduces hardware profiling and software energy modelling challenges beyond those of a single multi-threaded core, for several reasons:

- A significant amount of effort was required to construct, configure and program for this system.
- Multiple cores and power supplies must now be considered.
- Communication of data over a credit-based, cut-through routed network can be observed.

A more general exploration of Swallow's capabilities is presented in [HK15]. Energy profiling of swallow is performed in this thesis in Chapter 8 and a model proposed and evaluated in Chapter 9.

5.3. Research enabled by the XS1-L and Swallow

A collection of XS1-L processors assembled into a lattice of embedded compute nodes create a rich set of features that enable the novel work of this thesis to take place. These features begin in the core with the paradigms established in the ISA and reach as far as the network-level communication that departs from the more conventional shared memory approaches. Most importantly, these processors are embedded devices, not high-end application specific components, or large general purpose CPUs.

The work presented in this chapter forms the working knowledge necessary to conduct research along the themes defined in § 1.1. Tools for booting, running and debugging Swallow were contributed to the Swallow project during the course of this research. This hardware/software platform

can be used both for this and future research activities. The key contributions from this chapter will now be summarised, in relation to those research themes.

Use a multi-threaded embedded real time processor. Prior work, discussed in Chapter 3, focuses on single-threaded devices, and although recent research includes new parallel architectures, the selection of the XS1-L allows a number of unique properties to be explored in this space. In particular, the XS1-L ISA puts the software very close to the hardware, which may aid the modelling of energy for software running on the device.

Extend the system into a multi-core network of processors. In response to the limits of Dennard scaling, discussed in Chapter 4, parallelism is a necessary dimension into which both hardware and software must expand. The Swallow platform allows this to be studied in the embedded real time system space, where other platforms cater to different compute tasks.

Utilise novel or rarely used paradigms compared to previous work. The XS1-L and Swallow have several characteristics worthy of exploring in relation to the goals sought by this thesis. In particular, hardware-managed threads, time-deterministic execution, dedicated I/O instructions with no memory mapping, and a channel based communication abstraction provide a compelling list of features upon which to conduct novel research.

Provide a means of profiling hardware and evaluating modelling on multiple levels. The presented processor and Swallow system can be profiled as a single core or multiple cores. In the multi-core scenario, the core power supplies and I/O supplies can be measured, in order to establish different facets of energy consumption, enriching the energy models proposed in the remainder of this thesis.

Enable the cost of communication to be assessed in a message passing, rather than shared memory architecture The XS1 architecture provides message passing at the lowest implementation levels. This is propagated up to the software level through the XC programming language. Swallow allows many of these message passing cores to be utilised by parallel programs. Thus, these characteristics can be profiled and modelled to seek useful predictions of the energy consumption of such programs.

This chapter has reviewed the XS1-L processor and the multi-core XS1-L based Swallow system, highlighting the architecture and system level properties that are important to implementing event-driven and multi-threaded software. An adequate understanding of the principles underpinning the XS1-L and Swallow allow the research questions posed in this thesis to be further studied.

This concludes Part I of this thesis. It has provided the background research and knowledge pertinent to the exploration of the new research questions posed in Chapter 1. Part II details the efforts to answer those questions in earnest.

Part II.

Constructing a multi-threaded, multi-core energy model

Introduction

In Part I, three relevant areas of prior research were discussed, in addition to details of the hardware devices and platforms selected for use in this thesis. Part II presents the main contributions of this thesis, forming answers to the research questions posed and thesis statements made in Chapter 1. The background material from Part I will be referred to where relevant, and the relationship between the state of the art and this thesis's research contributions will be explored in more technical detail. This part is structured to cover three main areas:

1. energy modelling in relation to one multi-threaded core;
2. modelling a network of such cores at a system level, and;
3. analysis of how these contributions relate to other contemporary architectures.

A concluding chapter evaluates these contributions.

Chapter 6 proposes methods for exploring and capturing energy consumption data at the ISA level for an XS1 processor. It includes analysis of the significant parameters that need to be considered when constructing a model of this processor. These discoveries are then used in Chapter 7 to form a model. This model is then extended through regression techniques and subsequently tested in multiple contexts, including full tracing and statistics based simulation.

Energy profiling of the Swallow platform is presented in Chapter 8 to obtain multi-core communication costs. This new data, combined with the core-level profiling and energy model, is used to build a flexible graph oriented system level model, which is presented in Chapter 9.

Chapter 10 examines architectures other than the XS1, identifying the opportunities to apply the contributions of this thesis to other platforms, as well as highlighting where further work is required to achieve this.

This thesis is concluded with Chapter 11. It contains a summary evaluation of the complete work and draws the final conclusions from the contributions made. Further work is proposed based on the new possibilities created by this thesis, with a view to both improving upon this work and using it for new research.

6. Model design and profiling of an XS1-L multi-threaded core

This chapter details the first step in producing an energy model for a system of XS1-L processors: profiling one multi-threaded core. The goal of the profiling is to collect sufficient empirical data to provide a robust base for the model, expose processor characteristics in need of further investigation and allow extrapolation of more complex model parameters through regression and other methods.

6.1. Strategy

The strategy for constructing the model is comprised of several parts, with a significant amount of automation included to maximise data collection and opportunities for refinement:

- Establish a modelling approach.
- Create a test-bench to acquire data compatible with the selected modelling approach.
- Run the test-bench.
- Inspect the test-bench data, refining the tests and the framework as necessary.
- Construct the model using collected data.
- Verify the model against simple tests and more complex benchmarks, to determine its accuracy.
- Continue to refine the model, both through changes to the model structure and through new tests that provide additional data.

The process flow accommodating these parts is depicted in Figure 6.1. The remaining sections in this chapter describe the profiling method with consideration towards the design of the model, discoveries made during profiling and refinements made as a result. The subsequent chapter explores the model itself.

6.2. Profiling device behaviour

Profiling at the ISA level brings certain benefits and also disadvantages. For example, it does not require gate-level simulation of the device. However, without information on the exact implementation of the micro-architecture, some behavioural details become a *black box*, where the behaviours can potentially be exposed at the ISA level by suitable sequences of stimuli, but the explanation or a full understanding of these behaviours may not be possible at this level.

The profiling performed in this work seeks to expose sufficient information about the processor's energy characteristics so that an energy model constructed against this data yields an acceptable accuracy.

The relatively small instruction set and deterministic execution of the XS1-L processor bears similarity to the parameters used in the ISA energy model proposed by Tiwari, as described in § 3.2.2. Therefore, this approach is taken as a starting point and extended to account for the new parameters necessary to capture multi-threading and other new behaviours in the XS1-L. More model considerations are given in § 6.3.

Such a model requires a *base instruction cost* for instructions as well as *inter-instruction overheads*, plus any other effects not directly expressed by the stream of instructions that are executed. To construct a model in such a style, power measurements must be taken for individual instructions as well as measurements for pairs of instructions, so that the instruction base cost and

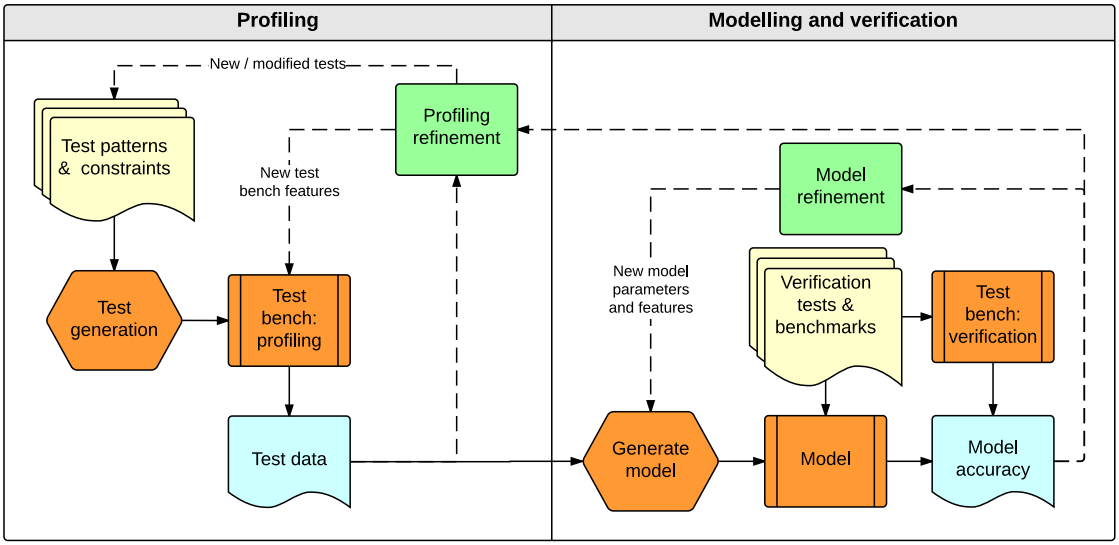


Figure 6.1: The process used to profile the XS1-L then produce and verify an energy model including refinement. Dashed lines denote manual effort and solid lines automatic. The process starts with the definition of test patterns and constraints, becoming a cyclical activity thereafter.

inter-instruction overhead can be considered. This data can then be used to estimate a program's energy, based on the sequence of instructions that it will run.

However, the XS1-L has features that cannot be accounted for in the Tiwari style model, therefore it must be extended. In doing so, the terminology must also be carefully selected and explained, so as to relate the prior models and the model that will be constructed in Chapter 7.

Base costs: processor vs. instruction

In the Tiwari model, the base cost is a *base instruction cost*. That is, each instruction has a contribution to power dissipation, before considering inter-instruction effects, but there is no separation between the instruction cost and any other always-present power dissipation in the processor. Decoupling instruction cost from the underlying processor cost gives a *base processor cost* that is unaffected by the instruction being executed.

In a sequential, single threaded microprocessor, a no-op instruction could represent the energy consumed when the processor is idle and thus be profiled to determine the base processor cost. The costs of executing meaningful instructions and the interactions between them can then be built on top of this base processor cost.

Taking into account the XS1-L's event driven architecture and therefore idle times in which no instructions execute, the base instruction cost can be defined as the minimum energy consumed when there are no active threads. Investigation into and establishment of a base processor cost for the XS1-L is detailed in full in § 6.4.

Instruction and inter-instruction costs

Once a base cost is established, the next challenge is how to handle instructions and inter-instruction overheads in the context of the XS1-L. To determine these in a similar fashion to the Tiwari model, the cost of executing each instruction and of transitioning between pairs of instructions must be determined.

Hardware measurements are required in order to establish the magnitude and variability of inter-instruction overheads, so that an appropriate granularity can be chosen for the model, delivering an acceptable performance/accuracy trade-off. For example, if the contribution of inter-instruction overhead is insignificant in comparison to the cost of each individual instruction, then it may not

be necessary to consider it, or it may be appropriate to generalise it if there is little variation in overhead between instructions.

To produce a model appropriate for the XS1-L's multi-threaded architecture, the processor must be seen as a pipeline that is executing a stream of unrelated instructions from neighbouring threads. Although dependencies and synchronisation may exist between *some* threads at a higher level of abstraction, on a per-instruction level, a pair of instructions travelling together through the pipeline are effectively unrelated in any real-world embedded application.

This precludes using a sequence of instructions in a thread as a means of exercising the processor in order to determine instruction costs and inter-instruction overheads. Instead, instruction overheads must be measured by controlling the instructions that a *collection of threads* are running, such that the exact sequence of instructions passing through the processor pipeline is known. § 6.4.3 describes how the measurement framework achieves these guarantees in order to extract inter-instruction overheads. Pairs of instructions remain sufficient for determining overheads, despite the device's four stage pipeline, due to the deterministic scheduling and in-order progression of instructions through the pipeline.

Thread cost

In addition to instruction costs, the parallelism present in the XS1-L, in the form of its hardware thread schedule, must be considered. It must be determined whether the number of active threads, and therefore amount of parallelism present in the system at any given time, has a measurable impact on power that should be accounted for in the model.

6.3. Model design considerations

In addition to the practicalities of collecting power data for the XS1-L, the design and use case for the energy model is also considered. The primary goal of the energy model is to allow a simulation of a piece of software to produce an estimate of the energy consumed by that software. It can provide novelty through its exposure of previously un-modelled characteristics (for example, due to the unique design of the target processor) and by providing simulation performance that is better than lower level hardware models such as Register Transfer Logic (RTL) based approaches.

6.3.1. Simulation performance

For a software energy model to be useful, it must be more convenient to run it than to instrument and measure a hardware system. The modelling approach used in this thesis requires the use of an Instruction Set Simulator (ISS). On a 2.26 GHz Intel Core i3 CPU, a full instruction trace simulation using the standard XMOS tool, `xsim` [JGL09] takes 51 minutes for a 0.4 second real-time benchmark. A simulation producing only execution statistics, using the faster `axe` [Osb11; Ker12a] tool, takes 40 seconds. The `axe` simulation accuracy is the same whether or not a trace or only statistics are produced, so the reduced information present in statistics is the only risk to model accuracy if `axe` is the chosen simulator. However, `xsim` is more accurate overall. Work on improving the accuracy of `axe` to bring it in line with `xsim` is discussed later, in § 9.2.

Thus, there is motivation to construct a model that can rely on instruction statistics rather than complete trace data. However, statistics alone make it impossible to account for inter-instruction overheads at a per-instruction level, because the exact sequences of executed instructions are not recorded. The impact of forgoing this must be considered during data collection.

6.3.2. Architecture Comparison

Table 6.1 illustrates the key differences between the target processor for this research and a sample of other processors used in previous work as detailed in § 3.2.2. The significant differences in pipeline implementation, threading methods, communication model and memory hierarchy serve to justify the goal of this work in creating the foundations of a model for the XS1-L. Chapter 10 compares a wider range of processor types to the XS1-L in more detail and discusses the applicability of the modelling approaches used in this work to those processors.

Feature	XS1-L	ARM7TDMI [LEM01]	C641T [IRF08]	Xeon Phi [SB13]
Cores	1	1	1	60+
Threads	8	1	1	4 per core
Instr. sched.	Round-robin threads, in order	In-order	In-order 2x4 VLIW	In-order
Forwarding	No	Yes	No	Yes
Com. model	Channels	Shared memory		
Mem. / cache	1-cycle SRAM, no cache	Optional caches	L2	L2 + tag-cache on ring network

Table 6.1: Comparison of key differences between various architectures.

The XS1-L has a unique multi-threading method compared to other modelled processor. Further, the single-cycle SRAM removes the need for a cache model. However, the channel communication implementation and underlying interconnect demand new profiling and modelling techniques. These are examined in detail with respect to multi-core modelling in Chapters 8 and 9.

6.4. `XMProfile`: A framework for profiling the XS1-L

`XMProfile` is the hardware-measurement framework constructed for this work to gather energy consumption data for the XS1-L. It is built with consideration to the following aims:

1. To execute code with a level of granularity that delivers certainty as to the trace of instructions through the pipeline.
2. To provide a measurement interface in order to easily collect energy data and attribute it to test cases.
3. To perform constrained generation of tests for automation of the profiling process.
4. To support the inclusion of benchmark code to enable comparisons between the resulting model and the actual energy characteristics of the target hardware.

As such, `XMProfile` is both a test-generation framework and an energy measurement tool. They can be used together or separately, although the generated test kernels are very tightly integrated into the structure of the measurement framework.

6.4.1. Hardware

The hardware platform for the energy profiling effort consists of two XS1-L devices, an XK-1 development board containing the *master* processor and a bespoke XMOS board containing an additional XS1-L — the *slave* processor or the Device Under Test (DUT). The bespoke board was modified to provide easy access to the core power supply of its XS1-L. The XK-1 development board controls a DC-DC power supply and an INA219 power measurement chip [Tex11], allowing power dissipation of the power supply to be sampled at a rate of up to 8 K samples per second with up to 11-bit resolution with a least-significant sample bit of 680 µW for the expected maximum current of the XS1-L.

In addition to controlling and monitoring the power supply of the DUT, the master processor is also responsible for synchronising tests against energy measurements in order to automate the collection of model data.

6.4.2. Software

The collection of software in `XMProfile` can be broken down into four key components:

1. Power measurement and data streaming to host PC.

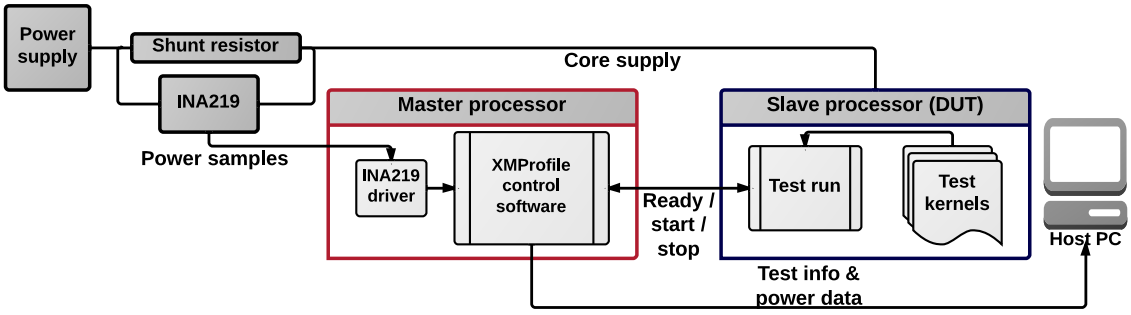


Figure 6.2: XMProfile test harness hardware and software structure.

2. Test loading and synchronisation with power measurements.

3. Test case generation with constrained random data for all instruction permutations in a given instruction subset.

4. Test control software, delivering fine-grained management of instruction flow during test kernel execution.

The master processor runs software that samples power values from the INA219. These samples are then streamed out over a USB interface to a host PC. At the end of each test run, an average power figure is calculated. This combination of streamed data and test run averages provides sufficient data to feed into an energy consumption model.

Tests are synchronised with power measurements by using the XMOS XS1's communication architecture. The master processor and DUT form a network over a 2-wire X-Link. The link is used as a trigger to signal the start of the next test, and halt the test once the test period is over.

6.4.3. Controlling the pipeline

Establishing the instruction costs and overheads through the pipeline requires the ability to control the order of instructions progressing through it. When subsequent instructions come from different threads, this is difficult to guarantee at a high level, such as with compiled C code. However, the XS1-L's single-cycle thread synchronisation allows the test harness to have precise control over which instructions in a thread are executing at any one time, provided the tests do not introduce any non-determinism with respect to execution time, such as through I/O operations. A typical test flow is depicted in Figure 6.3.

A test thread is a loop containing the body of instructions to be profiled, with minimal prologue and epilogue, but sufficient to ensure synchronisation and allow correct termination. Four threads, T0 to T3, are required to fill the pipeline and create instruction interactions on every clock cycle. To observe inter-instruction effects, the body of odd-numbered threads are populated with one instruction I_{odd}, whilst even-numbered threads are populated with another I_{even}. As the threads execute round-robin, the instruction executed at a given pipeline stage will alternate between I_{odd} and I_{even}, allowing specific inter-instruction effects to be measured.

All threads are synchronised against the thread that creates them, known as the master thread. In this case, T0 is the master and T1–T3 are its slaves. As such, the loop prologue and epilogue of the master thread is slightly different to that of the slave threads. During the test, at the start of each loop the slaves perform a synchronisation (`SSYNC` instruction) against the master thread (`MSYNC` instruction). If the master has received the end of test signal from the test harness, then it performs an `MJOIN` instead of an `MSYNC`. This kills the slave threads when they next synchronise. The slave threads execute no-op instructions when the master is performing the above checks.

To minimise the overhead of the execution of loop prologues and epilogues, the loop body must be sufficiently long. The number of body instructions, N_b, required to achieve a body to total instruction ratio, R, with N_o overhead instructions, is determined through Eq. (6.1).

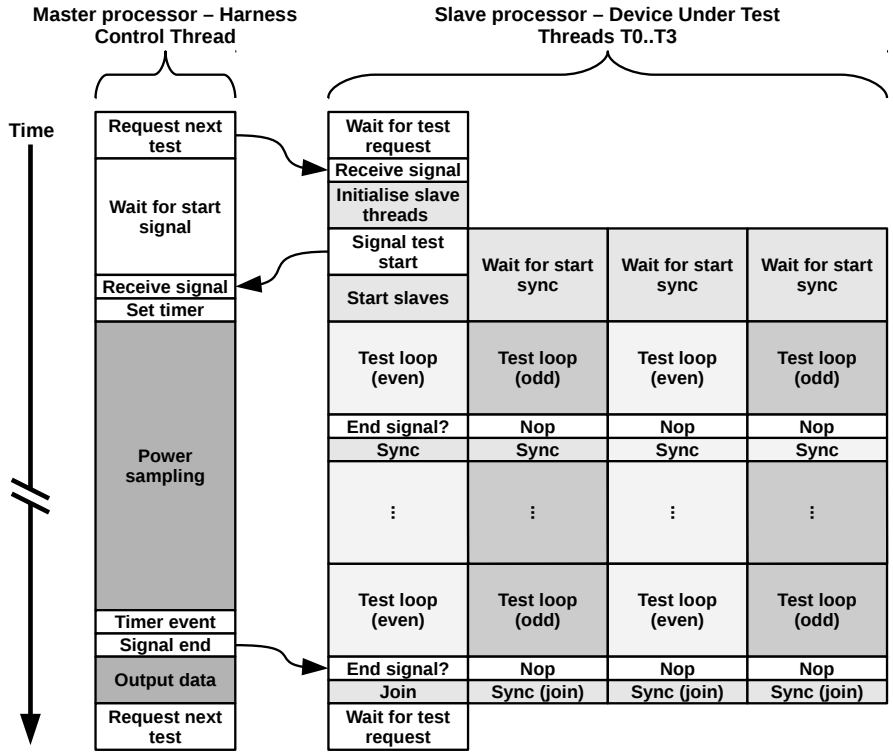


Figure 6.3: Test harness and DUT process flow.

$$N_b = \left\lceil \frac{N_o R}{1 - R} \right\rceil \tag{6.1}$$

Listing 6.1 and 6.2 show the minimal code padding used for a group of kernels used by threads in an example test titled `TestName`. The event vector for the first thread is configured to point at label `TestNameEnd`. When the test harness triggers this event, the thread will immediately jump to this address, provided line 7 has been executed at least once. This code gives $N_o = 4$, a very low overhead.

```
1  TestNameT0Loop:
2      # Retrieve synchroniser
3      ldw    r11,sp[0x3]
4      msync  res[r11]
5      # Unrolled instructions
6      # ...
7      setsr  0x1
8      bu     TestNameT0Loop
9      # ...
10 TestNameEnd:
11     mjoin  res[r3]
12     # Cleanup
```

Listing 6.1: Example kernel of first thread on the DUT.

```
1  TestNameT1Loop: #T2Loop, etc
2
3      nop
4      ssync #Break or proceed
5      # Unrolled instructions
6      # ...
7      nop
8      bu     TestNameT1Loop
9
10
11
12     # No cleanup
```

Listing 6.2: Example kernel of further slave threads.

The correctness of the thread synchronisation harness was validated in two ways. Firstly, against the XMOS architectural simulator `xsim`, to provide a cycle-by-cycle trace of the harness' execution. Secondly, the behaviour was confirmed on the hardware by putting I/O operations in the test body for each thread and observing the associated ports on an oscilloscope, to ensure that timing of the signal edges was as expected.

Thread schedule. Early testing yielded an interesting discovery in relation to how threads are scheduled into the pipeline. With a single active thread an instruction is issued once every four clock

Time-step	1 thread	2 threads	3 threads	4 threads	5 threads
1	$T_{0,0}$	$T_{0,0}$	$T_{0,0}$	$T_{0,0}$	$T_{0,0}$
2	—	—	$T_{1,0}$	$T_{1,0}$	$T_{1,0}$
3	—	$T_{1,0}$	$T_{2,0}$	$T_{2,0}$	$T_{2,0}$
4	—	—	—	$T_{3,0}$	$T_{3,0}$
5	$T_{0,1}$	$T_{0,1}$	$T_{0,1}$	$T_{0,1}$	$T_{4,0}$
6	—	—	$T_{1,1}$	$T_{1,1}$	$T_{0,1}$

Table 6.2: Representation of instruction sequence for various *active* thread counts, with threads represented as $T_{n,i}$, for thread number n and instruction number i.

cycles. When two threads are active, instructions are issued every other clock cycle. The alternative would be to issue two instructions (one from each thread), and then have two cycles where no instructions are issued. This is functionally equivalent, but it may have energy implications because it affects the switching within the pipeline.

With three active threads, an instruction is issued for three in every four clock cycles. For four or more active threads, an instruction is issued on every clock. Allocated, but inactive threads (i.e. threads waiting on events) do not issue instructions, so have no influence on scheduling. Table 6.2 illustrates the XS1-L's instruction and thread schedule in line with this observation.

6.5. Generating tests

Blocks of instructions are required to fill the loop bodies of test threads, the expectation being that the majority of test time will be spent executing those body instructions, giving a power figure for them. A number of ALU instructions hand-coded into test loops are used to gain understanding of what to expect and also to determine a good approach for automation.

Following this initial setup, the process of creating tests is largely automated. For 36 arithmetic operations, tests are generated for every possible pairing of them.

To account for data variation, constrained random data as well as constrained random source and destination operands are generated. This ensures that for each instruction the supplied data is valid (i.e. cannot cause an exception condition) and that results do not overwrite source registers, avoiding value convergence over the course of the loop body.

Constrained random data generation is used to provide different data widths to the test loops, with bit-widths of 32 (full width), 24, 16, 8, 4, 2, 1 and 0. Bit-masking is applied at code generation time to constrain the data range, so the test loops themselves are identical between runs at various data widths.

Exclusions

This approach is applied to 36 of the 203 instructions in the ISA, principally covering arithmetic operations in the CPU. This excludes branches, I/O, memory, communication and other resource related instructions. These other instructions can affect control flow, take multiple cycles or exhibit non-deterministic timing and so are not suitable for profiling in this way.

The divide instruction was also excluded from automated tests. The divide unit in the XS1-L is a serial divider with early-out capability. Thus, it will take up to 32 clock cycles to complete, potentially spanning multiple thread cycles. If the divide unit is in contention, then threads will remain scheduled and wait until they can claim it. This affects the thread execution timing and for this reason was avoided in the automated data collection phase.

Although it is quite possible to build test loops that utilise many of them, they cannot necessarily be generated or the result data be interpreted in the same automated way. It is necessary to either construct specific tests for these cases, with significantly more constraints than auto-generated tests, or produce more complex test loops comprising multiple instructions, extrapolating instruction costs using a suitable analysis method. Further tests are developed in § 6.5.1 and modelling of un-profiled instructions is explained in § 7.4

Generation process

The process to generate tests for each ALU instruction automatically is as follows:

1. Describe constraints on all immediate encodings (value range or set of possible values).
2. Describe characteristics of each instruction in terms of length, encoding, operand count (source & destination), immediate type and the number of source/destination registers to allocate.
3. For each unique pairing of instructions, generate odd and even threads for a test kernel, with the following generated contents:
   - For each instruction in a test, generate random values to populate the source registers within that instruction's constraints.
   - For each instruction in a test, generate random source and destination register addresses within specified range.
   - If an instruction has an immediate value, generate a random immediate within constraints.
   - Generate N_b instructions, satisfying Eq. (6.1).
4. Add test to list of tests to run.
5. Compile group of tests into framework, split into separate binaries if the processor's program memory limit is exceeded.

As an example of a set of constraints, take the instruction `ashr`, *arithmetic shift right*, with an immediate shift value. In the C programming language, `int Y = X >> I`, with constant shift amount `I`, is functionally equivalent to the `ashr` instruction. This is encoded as a 32-bit instruction. It has one input register and one output register, with an immediate value $\mathrm{I} \in \{1, 2, 3, 4, 5, 6, 7, 8, 16, 24, 32\}$, encoded together with the register addresses into 11 bits. The operands of the instruction are constrained by these parameters to ensure that only valid assembly instruction sequences are generated. Shifts by other amounts than those listed must be done using the three-register form of the instruction. Due to encoding techniques in the XS1 ISA, this is encoded together with the source and destination register addresses. As such it consumes fewer than four bits in the instruction, but this is recorded here as a 4-bit wide immediate value for simplicity.

With this process in place, energy data for the majority of arithmetic instructions can be collected in approximately 90 minutes. Discussion and analysis of these results is presented in the following section.

6.5.1. Custom profiling and extended tests

A number of instructions cannot be profiled using the completely automated methods described in the previous section. However, the `XMProfile` framework supports hand-written test patterns and constraints, allowing for custom profiling of instructions with more complex dependencies or behaviours.

For example, memory operations require a section of memory to access, and this must be populated with random data in order for a profiling run to yield realistic measurements. `XMProfile` is able to support this by providing a heap containing constrained random data which is re-initialised from a *shadow heap* between tests. The tests themselves require additional customisation, however, because of the fetch behaviour of the XS1-L. Repeated memory operations starve the instruction buffer for a thread, resulting in `FNOP`s occurring during execution (§ 5.1.1). Tests inducing varying frequencies of `FNOP` allow the cost of the `FNOP` to be separated from the memory operation under test.

Instructions covered by custom profiling runs include:

- Unconditional branching.
- Divide/remainder.

- Memory loading and storing of all available widths.
- Core-local channel communication.

These custom profiling tests require additional scrutiny and parameterisation before inclusion into the energy model. Moreover, the custom profiling tests do not cover the remainder of the instruction set. Further work is done to fill in these gaps. These custom instructions and unprofiled instructions are examined in § 7.4, after § 7.2 demonstrates and evaluates a model with these absent.

6.6. Profiling summary

This chapter has detailed the process of profiling the XS1-L in order to collect data for an ISA level energy model. The model is explored in the next chapter.

The profiling is largely automated thanks to the creation of the `XMProfile` framework for test generation and measurement. The profiling process allows very tight control over the processor's pipeline, and the test generation can be fully automated or customised as required.

This framework serves not just to provide data to be processed into a model, but also to allow the behaviour of the processor to be examined and reasoned about. For example, the precise thread schedule detailed in § 6.4.3 comes from the use of this framework and not processor documentation. This allows energy characteristics of the processor to be explained as well as modelled, furthering this thesis' goal of providing better insight into the energy consumption of embedded processors.

7. Core level XS1-L model implementation

The model presented in this chapter draws upon the research discussed in Chapter 3, extending that work to give consideration to the behaviours distinct to the hardware multi-threaded architecture of the XS1-L. It uses the `XMProfile` framework, described in the previous chapter. A significant portion of this work is published in [KE15b].

The outcome of the work presented in this chapter is a model and workflow that can be used to estimate the energy consumed by embedded multi-threaded programs run on the XS1-L processor. The error of the resultant models is as low as 2.67 %, as enhancements are implemented throughout the chapter, based on both observations, improvements to the modelling technique and new features in the modelling software.

The first stage of the modelling process focuses on the automatically obtained data via `XMProfile`, creating what is termed the *initial model*. This is presented in § 7.2. A model produced from more extensive profiling, through customised `XMProfile` runs and regression techniques, termed the *extended model*, is presented in § 7.4. In addition to the model construction and accuracy evaluation, this chapter presents a discussion of model performance in terms of the levels at which it can be applied, from trace-based simulation up to higher-level static analysis, in § 7.6.

7.1. Workflow

The experimental modelling tools proposed in this thesis aim to fit within a software development workflow. Throughout this and subsequent chapters, the tools are extended. However, they are built upon the workflow shown in Figure 7.1.

The flow is considered in three stages: compilation, simulation and inspection. The compilation stage is the standard compiler toolchain workflow and uses existing tools with no modification. The simulation stage utilises an Instruction Set Simulation (ISS), in this case `xsim` [JGL09] or `axe` [Osb11], bundled with the toolchain or available online, respectively. This is then fed into a trace analysis tool, `XMTraceM`.

The `XMTraceM` tool is the novel contribution to the workflow, applying an energy model to the simulated program in order to determine energy consumption and power dissipation in addition to the execution time information that the simulator can already provide. A report is then produced, which is considered within the final stage of the workflow, inspection. The inspection stage is an opportunity for the developer to determine, from the energy report, whether they wish to make further code changes and then repeat the workflow in an attempt to improve energy consumption.

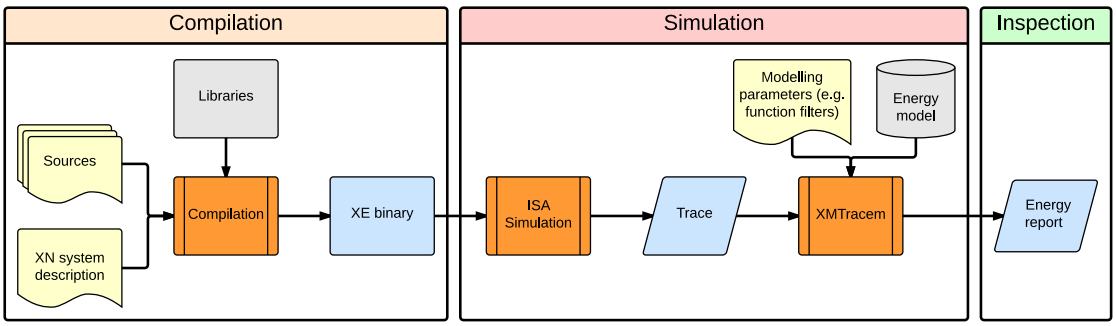


Figure 7.1: `XMTraceM` workflow for a single-core multi-threaded XMOS device.

As with a typical compiler and/or simulator workflow, outputs from various intermediate stages can also be useful to the developer. The binary produced by the compiler can be examined with other tools (such as debuggers or linters) if desired, and the simulation trace can also be valuable in reasoning about the resultant energy report. As such, the workflow should not be viewed as a strictly end-to-end process, but a staged progression with useful output in each stage.

7.2. A preliminary model

The initial runs of the automated `XMProfile` framework generate sufficient data to enable the following:

- Specify a base processor cost.
- Identification of the costs for executing a variable number of threads.
- Determine the energy consumed by different arithmetic instructions and their different encodings.
- Observe the extent of inter-instruction overheads in concurrent threads.
- Demonstrate the impact of data values on processor energy consumption.

These characteristics are discussed in turn, followed by further analysis looking at worst case energy and strategies for providing a simple method for modelling instructions that are not captured directly by profiling.

7.2.1. Base processor cost

The model requires a base procoessor cost to be established, as discussed previously in § 6.2. When a thread is waiting on an event, such as communication from another thread, an I/O event, or a timer comparison, it is de-scheduled and no more instructions from that thread are executed until the occurrence of one of the events it is waiting for. Experiments show, however, that the number of threads allocated, even if they are all de-scheduled and waiting for events, has a small impact on system energy, which can be attributed to the activity in the thread scheduler of the processor.

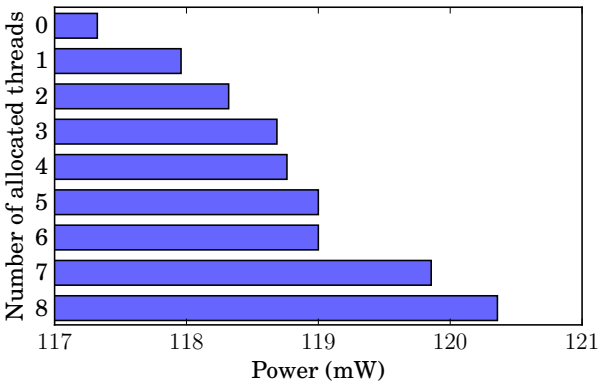
If the XS1-L's software energy model needs to consider not just the instructions that a thread is executing, but the number of threads that are executing, then the base processor cost should aim to capture the energy used when no threads are allocated.

This scenario was created by constructing a program that contained only a single thread that was subsequently released via the `FREET` (free current thread [May09b, p. 96]) instruction, leaving no allocated threads. Indeed, this yielded the lowest observed power dissipation when compared to any number of threads that were idle but still allocated, as shown in Figure 7.2a.

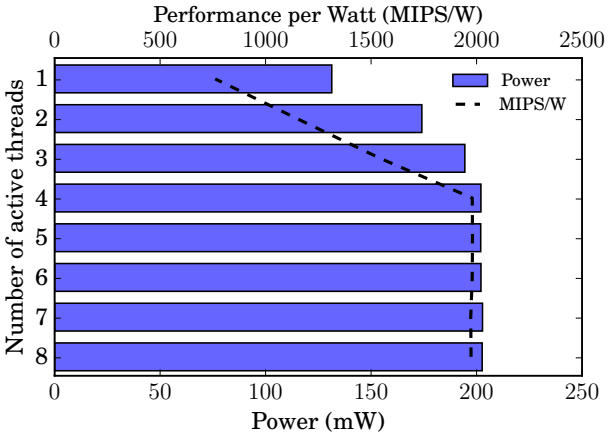
This data gives a base processor cost figure that is independent of both instruction sequences and the number of active threads, creating a stable *minimum power dissipation* upon which the rest of the processor energy model can be built, wherein active processor behaviour can be considered.

There is a non-linear relationship between the number of allocated threads, their state (active or waiting) and the power dissipation of the processor. An allocated thread when idle adds approximately 342 µW to the processor's power dissipation. This is lower than the least significant bit of the power sampling hardware (680 µW, see § 6.4.1), therefore it is subject to the effects of both noise and data averaging.

The idle power of the XS1-L processor is by some embedded standards relatively high. However, this is operating at 400 MHz, 1 V, in a 65 nm process technology and with no power gating. Even with all threads idle, there is port logic, clock trees, a Phase Locked Loop (PLL), the scheduler, and a network switch still active, such that I/O event response can begin at full speed within 10 ns. Additional power saving features such as a lower voltage, or a deep sleep with external wake-up, can be implemented with additional peripheral components, and frequency scaling is natively supported [XMO13b].



(a) Base processor cost analysis for XS1 with various allocated but otherwise idle threads.



(b) Thread costs for XS1-L performing `add` instructions on random data with performance per Watt overlaid.

Figure 7.2: Active and inactive thread costs for the XS1-L processor.

7.2.2. Thread cost

As the number of *active* threads increases, so too does the power dissipation, although the increase is less significant above four threads as the pipeline is always full beyond this thread-count. Figure 7.2b demonstrates this behaviour. The step in energy between 1 and 2 threads is greater than the step between 2 and 3 threads. This is believed to be related to the way in which smaller numbers of active threads are scheduled, as discussed in § 6.4.3. When 2 threads are scheduled, the pipeline transitions between active and idle twice as frequently as with 1 or 3 active threads. The performance per Watt of the processor at each thread count is overlaid onto Figure 7.2b, as per IPS_p in Eq. (5.1), highlighting the inefficiency of running less than 4 threads.

This characteristic bears similarities to the behaviour of the Xeon Phi [SB13], where instruction issue restrictions in the pipeline limit the energy efficiency of single-threaded performance on a core. However, the characterisation of the two processors deviates significantly at the memory hierarchy and communication model, particularly when considering a larger system-level view. The Phi is given more consideration in § 10.2.

From this data a baseline figure for the energy consumption of threads can be established. This can then be used as a component of the model, based on the number of allocated and active threads observed during simulation. The operations performed by the active threads must also be considered, and built on top of these baselines, to account for thread activity.

Encoding	Source registers	Destination registers	Immediate operands	Instruction length (bits)
`rus`	1	1*	1	16
`2r`	2	1*	—	16
`l2r`	2	1*	—	32
`2rus`	1	1	1	16
`3r`	2	1	—	16
`l2rus`	1	1	1	32
`l4r`	3	1	—	32
`l5r`	3	2	—	32
`l6r`	4	2	—	32
*Destination operand address is the same as first source operand.				

Table 7.1: Instruction encoding summary for the XS1 instructions under test.

In summary, both the cost of allocating a thread and the energy characteristics of various numbers of threads has been profiled, the data for which can be used as part of the multi-threaded software energy model.

7.2.3. Instruction cost

The cost of individual instructions and the inter-instruction overheads are closely connected, so they are considered simultaneously.

Using an approach similar to [TMW94a], the measured power for a given pair of instructions, $m_{i,j}$ is represented in array M. The average power when executing each pair is calculated to give an estimate of the inter-instruction overhead as array E, where $e_{i,j} = \frac{m_{i,i}+m_{j,j}}{2}$. Then the actual overhead, A, or the difference between estimated and measured power, $A = M - E$, is calculated.

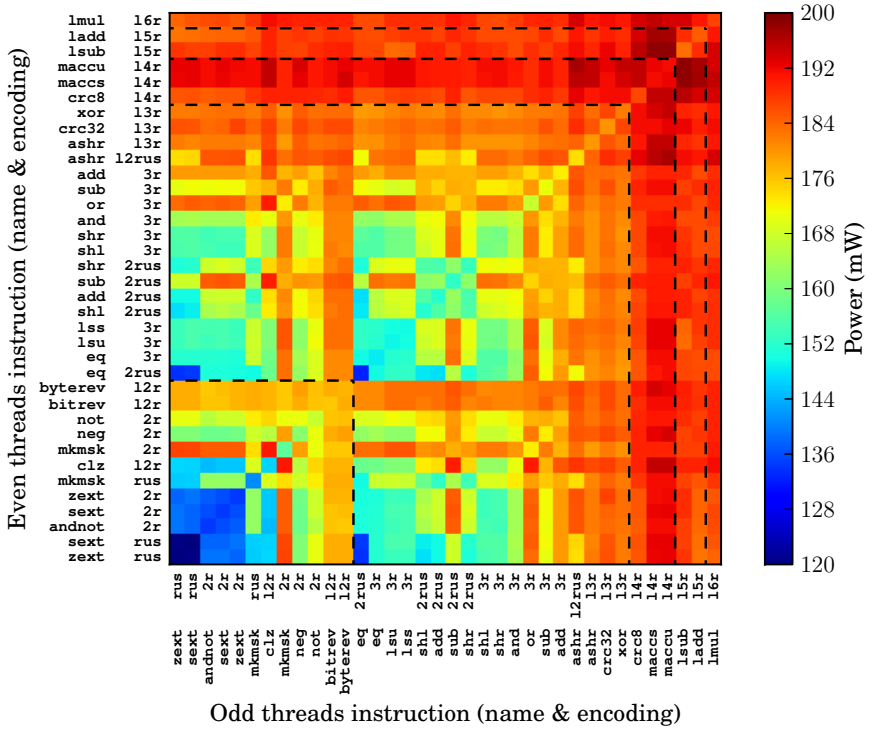
Figure 7.3 is a depiction of these arrays for 32-bit constrained random data. Data is represented as "heat-maps" where the colour represents the measured power. The axes of the graphs show instructions together with their encoding. For example, `add_3r` is an `add` instruction with three operands (two source registers and one destination register), as defined in the XS1 architecture manual [May09b, p. 47]. A brief summary of the encodings is presented in Table 7.1. Axes in the graphs are grouped by instruction operand count and separated by dashed lines, then sorted along the diagonal by individual instruction power in each group. That is, the power observed when all threads are executing the same instruction, thus there is no inter-instruction overhead.

Each cell on the grid of the heat-map is a measurement of the power taken during interleaved execution of the instructions indicated by the axes. The colour map is scaled to encapsulate the maximum and minimum observed values during the test suite and applied to the measured and estimated values for consistency and ease of comparison. The third map is independently scaled in order to expose where the overhead is significantly different from the baseline estimate.

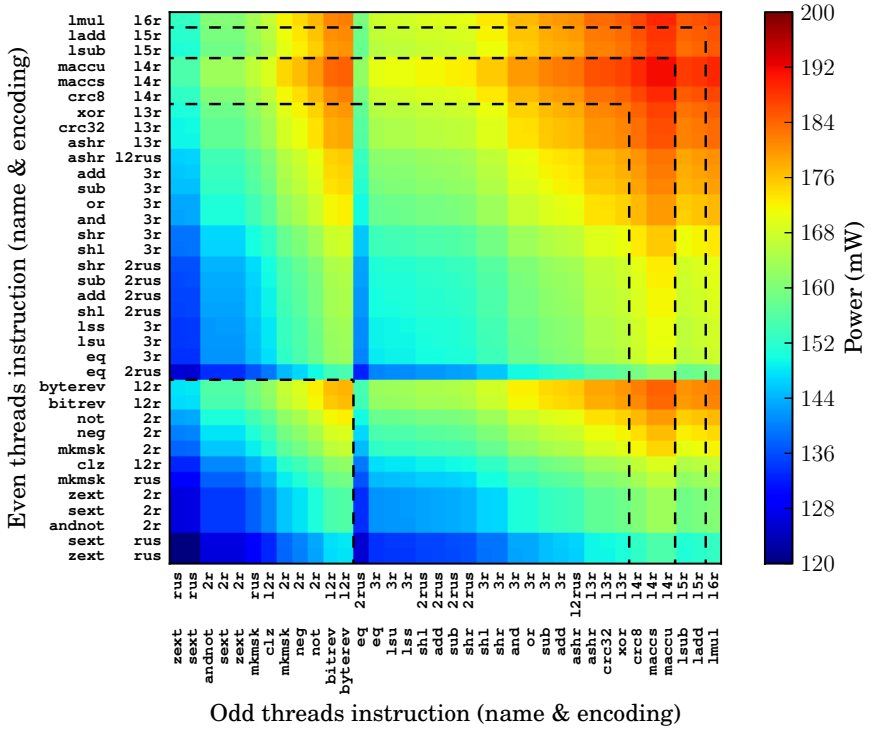
Figure 7.3a is the measured power, M. The diagonal of this map shows the instruction cost, when there is no inter-instruction effect. The lower triangle represents the power from interleaving every instruction in the test set and is mirrored into the upper triangle. From the measured costs in the diagonal of Figure 7.3a the estimated inter-instruction power, E, is calculated and shown in Figure 7.3b. The actual overhead, A, is then shown in Figure 7.3c.

Instructions that use a larger number of operands (`l4r`, `l5r`, `l6r`) exhibit higher power dissipation than other instructions. In addition to using more operands, they are encoded as long instructions, occupying 32-bits per instruction rather than 16-bits and so instruction fetches must be performed twice as often in order to carry them out. Of the 2–3 operand instructions there is a greater number of instructions and also a greater variation in power. The maximum variation in instruction power is as much as 40 % of the total core power.

The difference between the actual inter-instruction overheads and the estimation based on averaging, is typically less than 10 mW. This is an order of magnitude lower than the instruction power. With such a small impact on power, precise calculation may not be necessary in order to produce an effective model.

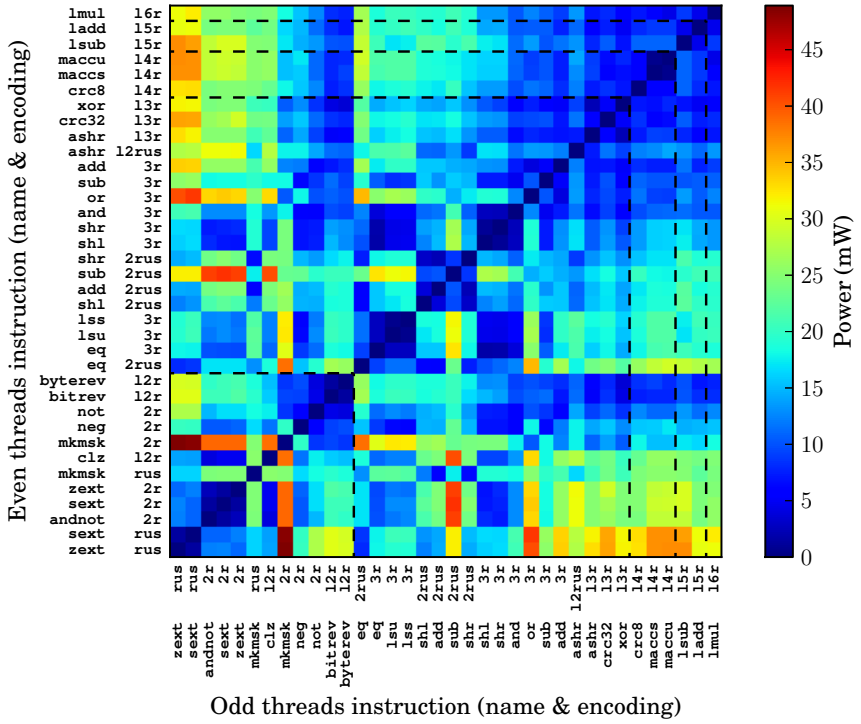


(a) Measured power, M.



(b) Estimated interaction, E.

Figure 7.3: Instruction power data and inter-instruction overhead calculation for 32-bit data, where dashed lines indicate a change in operand count.



(c) Inter-instruction overhead, A.

Figure 7.3: (cont.) Instruction power data and inter-instruction overhead calculation for 32-bit data, where dashed lines indicate a change in operand count.

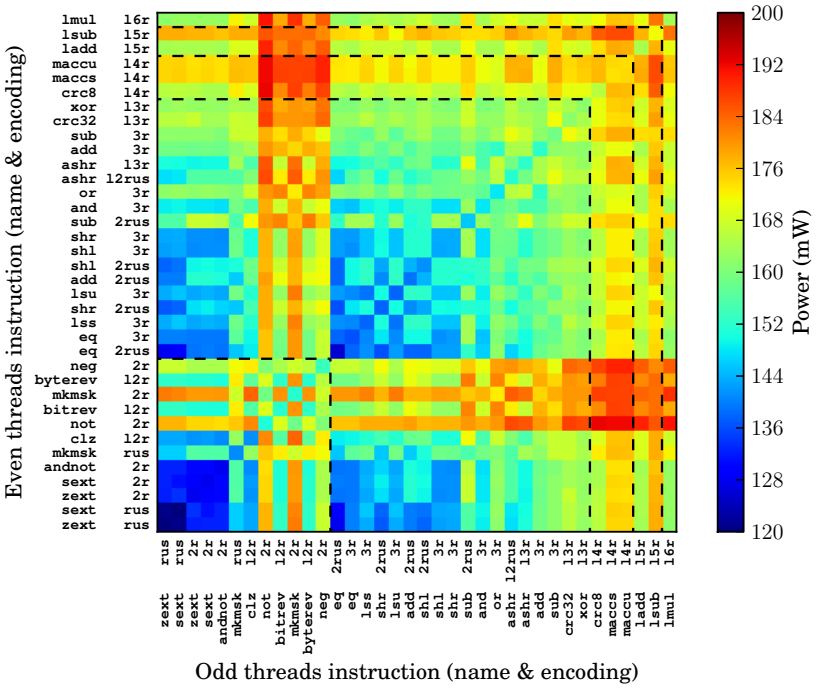
With this data and analysis, the instruction power, combined with the observed inter-instruction overheads can be incorporated into the software energy model. In addition to this, simplification of the model may be possible if the average power per operand count yields sufficient accuracy. However, the variation in instruction cost is significant enough that it is not appropriate to consider all instructions equal in this particular case, unlike in some other architectures, such as that examined by [RJ98]. Finally, due to its limited impact, the inter-instruction overhead does not necessarily need to be considered for every possible instruction combination.

7.2.4. Data width's impact on processor energy consumption

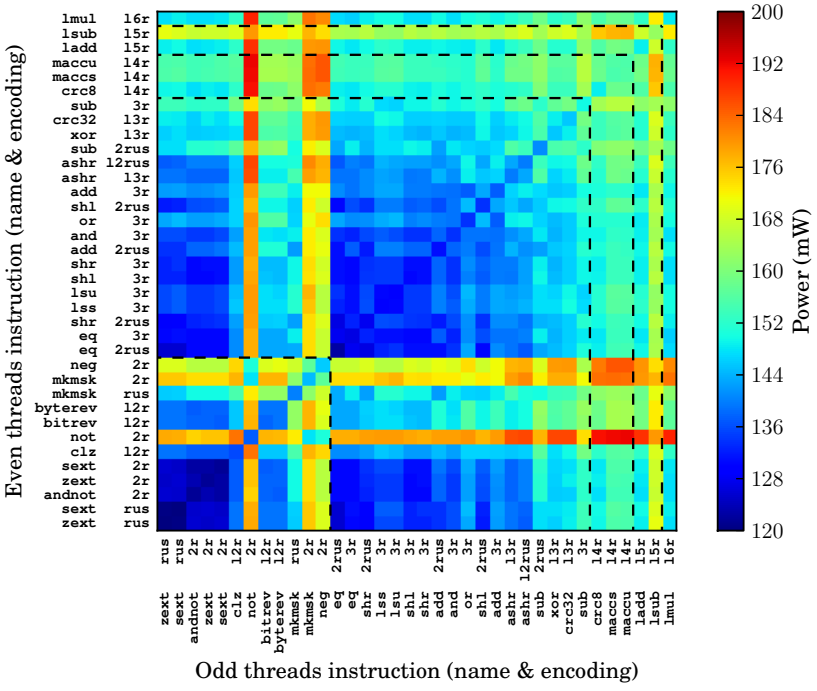
Given the observation in § 7.2.3 that the number of operands has a significant effect on the power, it was hypothesised that a significant relationship exists between the data values and the processor energy consumption. The Tiwari model does not account for data, although variations on it do, such as [Ste+01a]. Therefore, it was necessary to investigate the significance of data width's impact on the XS1-L's energy consumption.

Test runs were re-executed for several constrained random data sets, in this case 0, 1, 2, 4, 8, 16, and 24 bits in addition to the 32-bit data already collected, as described in § 6.5. Using the same plotting technique as described in § 7.2.3, Figure 7.4 shows measured power data, M, for data-widths of 16 and 8 bits.

The figure shows that as we restrict the data width, the energy consumed by instructions drops, with a few exceptions. The extent of the reduction is dependent upon the operation being performed. For example, an addition operation will at most produce a number 1-bit longer than its longest source operand, whereas a multiplication may produce a number twice as long (assuming no truncation due to overflow). Exceptional cases such as `mkmsk` (make-mask, for producing bit-masks) and `not` (bit-wise logical not), typically cause upper-bits in the data-path to flip even for low-range source values. This leads to "hot stripes" in the heat maps, distinguishing data-width dependent instructions from those that are not.



(a) 16-bit data.



(b) 8-bit data.

Figure 7.4: Instruction power data for various data widths, with dashed lines denoting a change in operand count. A reduction in power can be seen for narrower data.

	Input	Output
T0 & T2	0x55555555,0x55555555,0x55555555,0x55555555	0x1c71c721,0xe38e38e3
T1 & T3	0xaaaaaaaa,0xaaaaaaaa,0xaaaaaaaa,0xaaaaaaaa	0x71c71c72,0x38e38e38
XOR	0xffffffff,0xffffffff,0xffffffff,0xffffffff	0x6db6db53,0xdb6db6db
Hamming weight	128	42

Table 7.2: Interleaved `lmul` calculations and the Hamming weight of the inputs and outputs during thread transitions.

0-bit data	16-bit data	32-bit data	Worst case data
131 mW	164 mW	189 mW	222 mW

Table 7.3: Power measurements for `lmul` under differing data conditions.

These results demonstrate that data is a significant contributor to the power dissipation in this processor, although its impact depends on the instructions used by the application. As such, data width should be given some consideration in the multi-threaded software energy model. For example, if necessary, a range-limited application could require that a scaling factor be applied to the model in order to account for reduced data width, in the order of 1-2 mW per bit data width, with some exceptions for instructions such as the outliers exposed in Figure 7.4.

7.2.5. Maximising power dissipation

With low-range constrained random data, the minimum energy used by any given instruction is observed. However, high-range data does not necessarily give a maximum. A test was constructed specifically to try to establish the maximum energy used by the processor's arithmetic unit and data-path.

This test interleaves `lmul` (long multiply and add) instructions, which require the largest number of operands — four source (x, y, v and w), with two destination (d and e) implementing the operation described in Eq. (7.1) and detailed in the XMOS ISA manual [May09b].

$$\begin{aligned} e &= r[31:0] \\ d &= r[63:32] \\ \text{where } r &= x \times y + v + w \end{aligned} \tag{7.1}$$

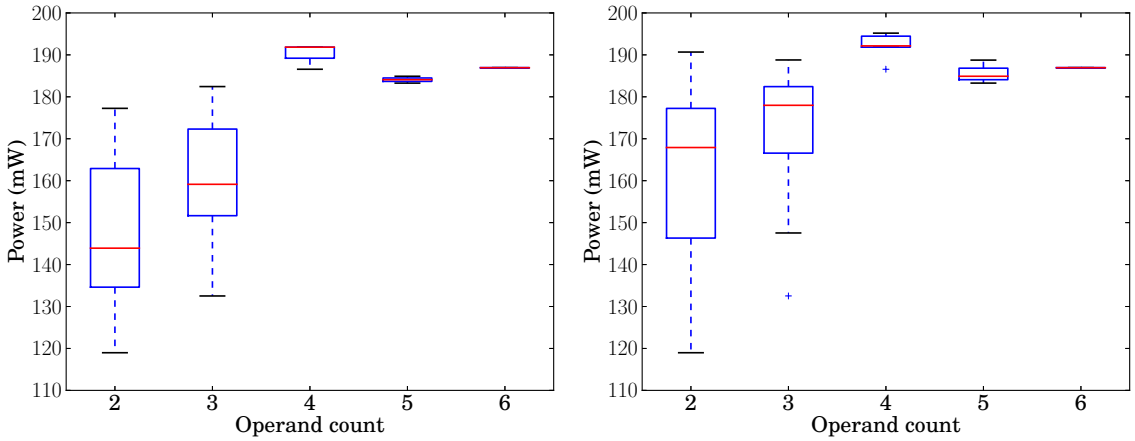
The even pair of test threads were loaded with the value `0x55555555` into all source registers and the odd pair with `0xaaaaaaaa`. This ensures that every bit on the input to the multiplier is flipped on every clock cycle, along with two-thirds of bits in the output. This is demonstrated in Table 7.2, where the Hamming weight (the number of differing bits across a pair of values) between inputs and outputs for the threads is shown.

In testing, this yielded a power dissipation of 222 mW, an approximately 15 % increase in the power of the instruction compared to regular 32-bit tests. Table 7.3 shows the disparity between the worst (or pathological) power dissipation of an instruction and the typical power for random data of various widths. The power used varies by up to 1.7x.

While this is an important observation, such conditions are unlikely to occur frequently. The inclusion of pathological behaviours into the model would increase the model complexity with very low likelihood of improving accuracy. As such, this behaviour is not incorporated into the model. Nevertheless, awareness of pathological energy consumption is useful for debugging, and could be considered alongside other cases such as the possibility of a naïvely implemented algorithm stalling the pipeline and degrading performance with poorly scheduled sequential memory operations.

7.2.6. Grouping instructions

Forming a hypothesis based on the data collected up to this point, it may be possible to group instructions by operand count (or a proxy of it, such as instruction encoding) rather than individually modelling instructions. This subsection seeks to determine what effect this might have on model accuracy.



(a) Diagonal of heat-map group.

(b) Triangle of heat-map group.

Figure 7.5: Box-whisker distribution of measurements for instructions grouped by operand count.

Figure 7.5 shows box plots for the five groupings of instructions, based on the operand counts that were identified from the heat maps earlier in this section. The data was generated from the same set as the heat map in Figure 7.3a. Figure 7.5a is an analysis of the data along the diagonal for each group, whilst Figure 7.5b also includes the triangle on the diagonal of each group.

It is shown that grouping 4, 5 and 6 operand instructions should not significantly impact model accuracy. However, these groupings contain far fewer instructions than the 2 and 3 operand sets. Indeed, the remaining sets show significant variance. The data along the diagonal exhibits similar variance to when the triangle is included.

This data demonstrates that operand count can give an indication of the energy consumed by an instruction, but using it as the sole indicator of an instruction's energy consumption may lead to higher model error, depending on the types of instructions executed by the application under analysis.

7.2.7. Decisions guided by measurement data

From the data collected and the analysis performed, decisions can be made with respect to the approach to be used to construct a model to estimate multi-threaded software energy consumption. The trade off between model accuracy and complexity must be considered, so that performance can be maximised whilst delivering an error margin similar to previous approaches as discussed in § 3.2.2. The following decisions were made:

Generalise the inter-instruction overhead. In Figure 7.3 it was shown that the power per instruction changes in the order of tens of mW, depending on the instruction type, whereas, the inter-instruction overhead varies in the order of less than 10 mW for the majority of cases. An average overhead can be used in place of individual overheads, with a low impact on accuracy, giving more flexibility to the ways in which a software energy model can be implemented and where its data can be sourced from.

Use instruction statistics rather than trace data. In order to establish a reasonable trade-off between performance and accuracy, a fast, higher-level model can use instruction statistics rather than complete trace data. The performance difference between statistics and trace collection is covered in § 6.3.1, where it was established that statistics collection is significantly quicker. The model can be refined and lowered to trace level if a higher level of accuracy is deemed necessary.

Further to the above, some practical issues must also be addressed, in particular the challenge of modelling instructions that were not directly tested by `XMProfile`. Two solutions are proposed:

Group instructions rather than considering their individual power. Grouping instructions simplifies the modelling process whilst capturing what the data suggests to be the most significant contribution to energy consumption: the amount of data-path activity caused by the operand count. In order to establish the impact of grouping, the model will be implemented both in grouped form and at individual instruction level, to allow a comparison of the error margin, as discussed in § 7.2.6. Instructions not directly profiled can be accounted for by assigning them to a group based on their operand count, with the intent of giving a better power estimate than a single default for all un-profiled instructions.

Utilise a default energy cost for instructions that have not been profiled. In the case where instructions are modelled individually rather than groups a default value will be used. This will be based on the 3-register instructions' average power as it is a frequently occurring encoding. This creates a good opportunity to evaluate a preliminary model against the group model and also give insight into whether the profiling needs to expanded to the complete ISA.

These decisions, based on data gathered and hypotheses formed through the earlier parts of this section, aim to build a model that is sufficiently accurate for a software-level energy model. This should be the case in relative terms, but ideally with single-digit percentage error in absolute terms as well. The form of model proposed here can utilise the fastest forms of instruction set simulation and also be flexible enough to be transferable to other levels of abstraction with minimal effort.

7.2.8. Model construction

The model constructs an estimate of software energy consumption by amassing information based on the execution statistics from instruction set simulation. These execution statistics, which can be obtained via a run of the fast `axe` or slower `xsim` simulator, provide a break-down of all the instructions executed on an XS1 core, and how many times an instruction is executed per hardware thread.

From the execution statistics, the following data is available:

- Total execution time (in cycles).
- The number of times each instruction is executed within each thread.
- Number of operands used for each instruction (based on encoding).

This can be combined with the data extracted from `XMProfile` in the previous section, namely:

- The *base power* dissipated by the processor.
- The *thread cost* dependent on how many threads are active.
- *Instruction power* as measured by the profiling runs.
- An average of the *inter-instruction overhead*, to represent switching between instructions, calculated from the pair-wise inter-instruction overheads measured during profiling.

Two main variations on the model were produced: one that groups instruction power based on the operand count of instructions and a version that considers each instruction's power individually. For individual instruction modelling, the 36 directly measured ALU-based instructions have power specified, with a default power value of 150 mW for any other executed instructions. With the grouped model, instructions outside this set of 36 ALU operations are assigned power values according to their operand count.

In both cases, the model uses the execution statistics to determine how much time the processor spends executing each instruction as well as how many threads are active at that time. The thread activity is not a precise calculation because a full trace is not captured. Instead, activity is estimated based on the distribution of instructions executed by each thread over the complete run-time.

Eq. (7.2) describes the energy of a program, E_p, using a similar method to Eq. (3.1). However, time is considered explicitly in this new model. This gives us an energy value rather than power. In addition, this makes it possible to account for idle time, wherein no instructions are executed because no threads are active. To achieve this, the number of cycles with no active threads, N_{idle}, is measured, then multiplied by the clock period, T_{clk}, and base power, P_{base}, which is the power dissipated when the processor is idle.

Next, to account for pipeline activity, for each possible number of concurrent threads up to the maximum, N_t, a multiplier, M_t, is applied for that level of concurrency. Finally, for all instructions in the Instruction Set Architecture, ISA, each instruction, i, at concurrency level t, is assigned an instruction power, P_i, that is scaled by a constant overhead, O, to account for inter-instruction overheads. The base power is then added and the result is multiplied by the clock period and the number of times this instruction is executed at concurrency level t, $N_{i,t}$. This gives an estimate for the total energy consumed over the runtime of the application within the processor core, accounting for potential variation in concurrency level over that time, as well as idle time.

$$E_{\text{p}} = P_{\text{base}} N_{\text{idle}} T_{\text{clk}} + \sum_{t=1}^{N_t} \sum_{i \in \text{ISA}} \left(\left(M_t P_i O + P_{\text{base}} \right) N_{i,t} T_{\text{clk}} \right) \tag{7.2}$$

For grouped instructions, the instruction power, P_i, is replaced with a lookup against which group the instruction belongs to, $P_{G(i)}$, applied as shown in Eq. (7.3).

$$E_{\text{p}} = P_{\text{base}} N_{\text{idle}} T_{\text{clk}} + \sum_{t=1}^{N_t} \sum_{i \in \text{ISA}} \left(\left(M_t P_{G(i)} O + P_{\text{base}} \right) N_{i,t} T_{\text{clk}} \right) \tag{7.3}$$

These models can be used on instruction execution statistics, alongside the data collected for the XS1-L processor, to produce a value representing the estimated energy consumption in Joules of the simulated multi-threaded program, p.

The XS1-L differs from various modelled processors in that it has no caches. Cache hierarchies therefore do not need to be considered, which simplifies one aspect of modelling. However, there are other complexities in the processor that require new approaches in order to account for them. In particular, idle time is captured explicitly, which is appropriate for an I/O centric, event-driven processor. Further, the introduction of threading-level into the model is necessary for sufficiently accurate energy estimation.

Without the M_t term, instruction power would be mis-predicted by an additional 10% or more for single threaded sections of a program, where $M_1 = 0.25$ as used in our modelling. A smaller error would also exist for 2 or 3 threaded execution. An alternative method of accounting for this would be to have separate costs for each instruction at each level of concurrency, bringing the model closer to prior single-threaded work such as [TMW94b]. The M_t parameterized approach is preferred because it reduces the profiling effort required. In addition, the parameterized model is a closer representation of the processor's pipeline characteristics and higher level analysis can also benefit from the model exposing threading in this way.

7.3. Preliminary model evaluation

This section describes the tests that are used to determine the accuracy of the initial core models compared to real hardware, and then evaluates this performance. The outcome is an assessment of which of the two approaches is the best, considering both performance and accuracy.

7.3.1. Testing the model

To test the model, a suite of benchmarks was selected and run through the `axe` XS1 Instruction Set Simulation, the statistics from which were passed through the model, to estimate the benchmark's energy consumption.

The benchmarks represent generic workloads as well as workloads typical of the XS1-L processor. The list of benchmarks is described in Table 7.4. These benchmarks were selected to utilize

Name	Description	Utilised libraries or software	Number of threads	Proportion of instructions profiled directly
Idle	A single thread that sleeps.	*None*	1	10 %
Dhry	Dhrystone benchmark, run once, or twice concurrently.	Dhrystone [Wei84]	1, 2	33 %
LZWK	Modified LZW for low-memory and real-time, single thread.	Own, modified LZW [Wel84]	1	42 %
SHA2	SHA2 hash function with "client" and "server" threads.	`sc_crypto`	2	68 %
Sca-add	Matrix addition with a scalar value, shared memory.	`sc_matrix`	4	39 %
Arr-mul	Piecewise multiplication of two arrays.	`sc_matrix`	4	36 %
Mat-mul	Matrix cross-product.	`sc_matrix`	4	36 %
Mix	Simple audio mixing, 24-bit samples, 4 and 6 input channels.	`sc_matrix`	4, 6	60 %
Mix alt	Two channel audio mixing, more advanced approach.	`sc_audio_mixer`	2	32 %
Matrix, crypto and mixer libraries available from `https://github.com/xcore/`				

Table 7.4: List of benchmarks used to evaluate energy model accuracy.

various processor features, including shared memory, message passing, various degrees of parallelism, integer arithmetic, string processing and fused operations such as multiply-accumulate. The proportion of instructions (based on the number of times each instruction is executed) that are directly modeled is noted in the rightmost column of the table, to give an indication of how complete the model is in relation to the particular test's distribution of instructions. A memory-intensive benchmark will rely more upon the default energy model value, due to this version of the model not directly handling memory instructions. See § 7.4 for a version incorporating them.

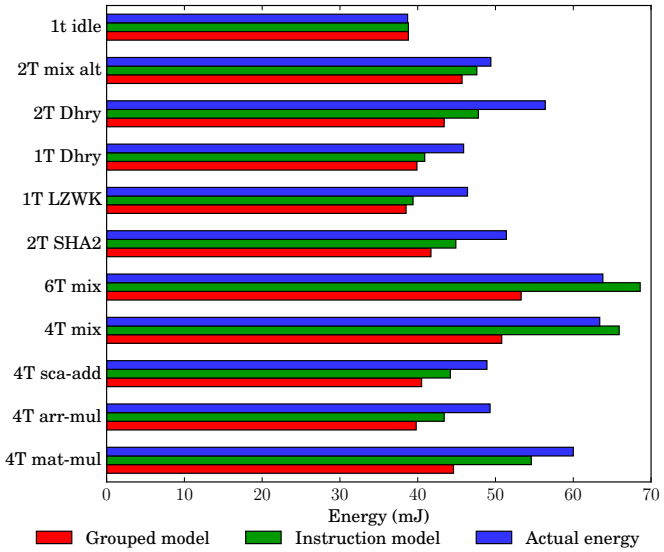
Tests were run for 0.4 seconds to achieve a simulation time of less than 1 minute whilst providing sufficient run-time for thousands of power samples to be collected. This simulation time is a reasonable length of time for a programmer to wait for an energy figure, compared to the effort of instrumenting and measuring physical hardware. If the programmer does not have hardware access at the time, longer simulation may be acceptable. Ultimately, run-time helps to establish what this modelling method can achieve when considering the needs of software developers. Longer and shorter test runs were also performed on several benchmarks to verify that the model and energy readings did not diverge over extended execution.

7.3.2. Evaluation

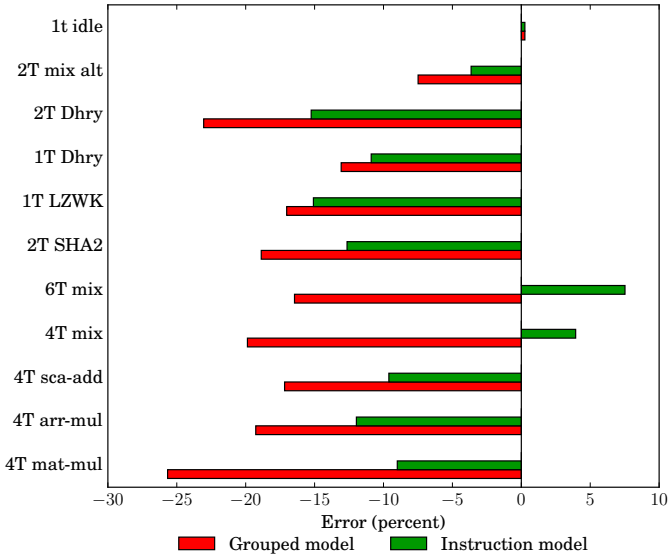
This evaluation presents a comparison between energy estimations from the model and measurements taken on real hardware, when running the benchmarks previously discussed in § 7.3.1. Both Equations (7.2) and (7.3), the individual instruction and grouped instruction models are examined.

In Figure 7.6a, both models are seen to be calibrated well against a single idle thread, where only the base power is present. Figure 7.6b shows that the worst case error for the grouped model is −26 % whilst for the instruction level model it is −16 %. The average error is −16 % for the grouped model and −7 % for the instruction level model.

Given the consistent under-estimation of the grouped model, it should be possible to improve the error margin to approximately ±10 % or better. However, this cannot be achieved with a naïve flat offset, as this would skew idle energy accuracy, which is particularly important in an event driven system that may have a low duty cycle (long periods of inactivity between external stimuli such as network events, user input, etc.).

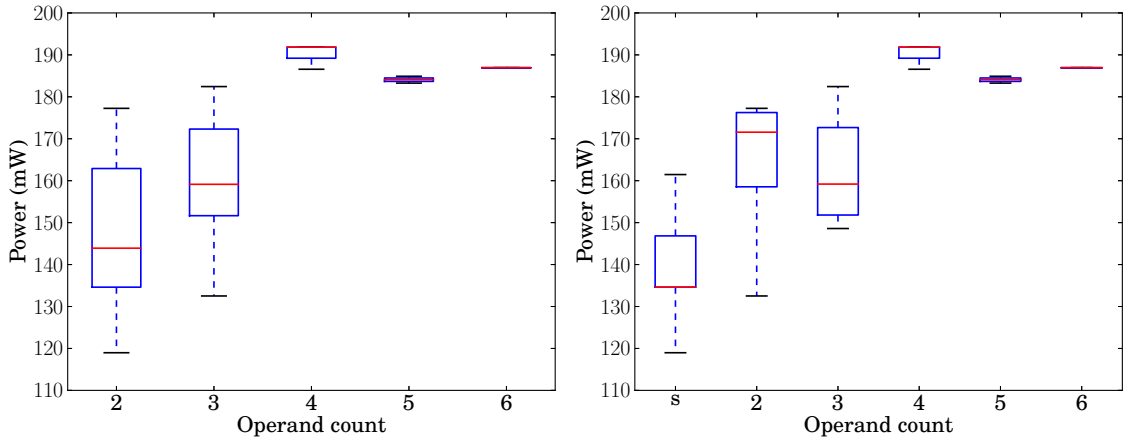


(a) Results of benchmarks for the models vs. actual device measurement.



(b) Accuracy of the two model approaches compared to observed device energy.

Figure 7.6: Benchmark energy results and error margins.



(a) Original grouping

(b) Additional group, `s`

Figure 7.7: Box-whisker comparison between original groupings and with lower power instructions separated into a special group, `s`.

In Sections § 7.2.4 and § 7.2.6, it was observed that instructions which perform value extension, or always produce low-range or Boolean results, are the lowest power instructions. Working under the hypothesis that these instructions contribute the most to the error in the grouped instruction approach, removing them from the 2 & 3 operand sets and putting them into a special set, `s`, may avert this behavior. Figure 7.7 shows that this additional split reduces the variance of the groupings when compared to the initial groupings chosen and analyzed in Figure 7.5.

Further examination and experimentation with the values chosen for the instruction groups, particularly when compared against real-world applications, could be a route to refinement. The same approach could also be used to further improve the instruction level model. Linear regression analysis can be used to estimate the energy of unprofiled instructions, combining the existing data set and properties shared between instructions, such as instruction width and operand count. This approach has been used successfully on other architectures [LEM01; NL13].

The data from our testing clearly indicates that the simplification achieved by the grouped model does not outweigh the additional error that it introduces. The grouped model reduces the size of the lookup table by an order of magnitude, but this simply reduces the space requirement and does not improve lookup time. Our per-instruction model is therefore the better method for this processor.

7.4. An extended core energy model

In § 7.2 it was demonstrated that instruction costs varied by as much as 80 mW, or 40 % of the total core power. In addition not all instructions can be directly profiled by the framework detailed in § 6.4. As such, selecting a "default" estimated instruction cost for an unprofiled instruction can introduce an undesirable error. Careful selection of the default will reduce this impact, but will not eliminate the error in all cases.

It was shown in § 7.2.8 that although a correlation exists between instruction encoding, operand count and power, it was not an appropriate means by which to group the power of all instructions. This is in part due to the use of groupings even for instructions that were directly profiled.

It is proposed that a more accurate estimate can be attained for unprofiled instructions by characterising them based on multiple parameters, all of which can be statically determined, in order to find a group of profiled instructions that closest represent the unprofiled candidate. This is achieved by using linear regression and regression-based decision trees. This seeks to preserve accuracy for direct profiled instructions and improve it for the rest of the ISA.

In addition, specific tests can be crafted that can be exercised within `XMProfile`, but do not fit the same auto-generation patterns of the arithmetic instructions. This section presents a description of the extended profile tests and the regression tree construction process.

7.4.1. Additional tests

Extra profiling runs against specific constrained tests were listed in § 6.5.1. A summary of the results incorporated into the model from these profiling runs is given here.

`FNOP` and memory operations

The behaviour of the `FNOP`, described in § 5.1.1, is an important consideration in this ISA level energy model. Instruction alignment and memory access can cause `FNOP`s that increase the execution time and therefore energy consumption of the program.

The long sequences of loads or stores used in the `XMProfile` profiling runs introduce these `FNOP`s by starving the instruction buffer. As such, additional testing is done, with `nop` instructions interleaved through the test kernels to influence the `FNOP` frequency. This is then reconciled against the known energy consumed by a regular `nop` in order to determine costs for the memory operations and the `FNOP`s.

Memory operations have been shown to take up to 40 % of operations in software run on the XS1-L [PHB13, p. 6]. Incorporating these into the model directly, rather than using a default value, is therefore compelling.

Branching

A sequence of unconditional branches is generated that traverses the entire code space of the kernel, but along a non-linear path. The path is generated by randomly selecting an unvisited address within the kernel, then generating the appropriate branch target offset.

Divide / remainder

The divide and remainder instructions can be generated in the same way as other arithmetic operations, with data constraints to prevent divide by zero exceptions. However, divide and remainder may take several cycles to complete. This affects execution time and is exacerbated when multiple threads contend for access to the divider unit and must wait.

When using divide, a relatively low power is observed of 30 mW above base power, but the total energy consumption is high, due to increased cycle count (up to 32), needed for it to complete.

Communication / resource operations

Core-local communication is profiled, which utilises channel resources, but not any external networking. This is done thorough specific tests that exchange randomly generated data between pairs of threads. The number of `OUT` and `IN` operations must be balanced between the thread pairs to avoid deadlock. As such, the cost of these is considered equal within the model.

Extended profiling energy data

The result of an extended profiling run, including original and custom tests, is presented in Figure 7.8. It can be interpreted in the same way as Figure 7.3a, but includes some additional information that requires further consideration.

The instruction labels are coloured by instruction width, with short 16-bit instructions labelled in green and long 32-bit instructions in red. Typically, the longer instructions have a higher cost. Two reasons can be given for this. Firstly, many of the long instructions use more operands, thus the data-path is wider, resulting in more dynamic power dissipation during execution. Secondly, instructions must be fetched more frequently in order to keep the instruction buffer full. Thus, the fetch unit and memory are more active for tests using long instructions.

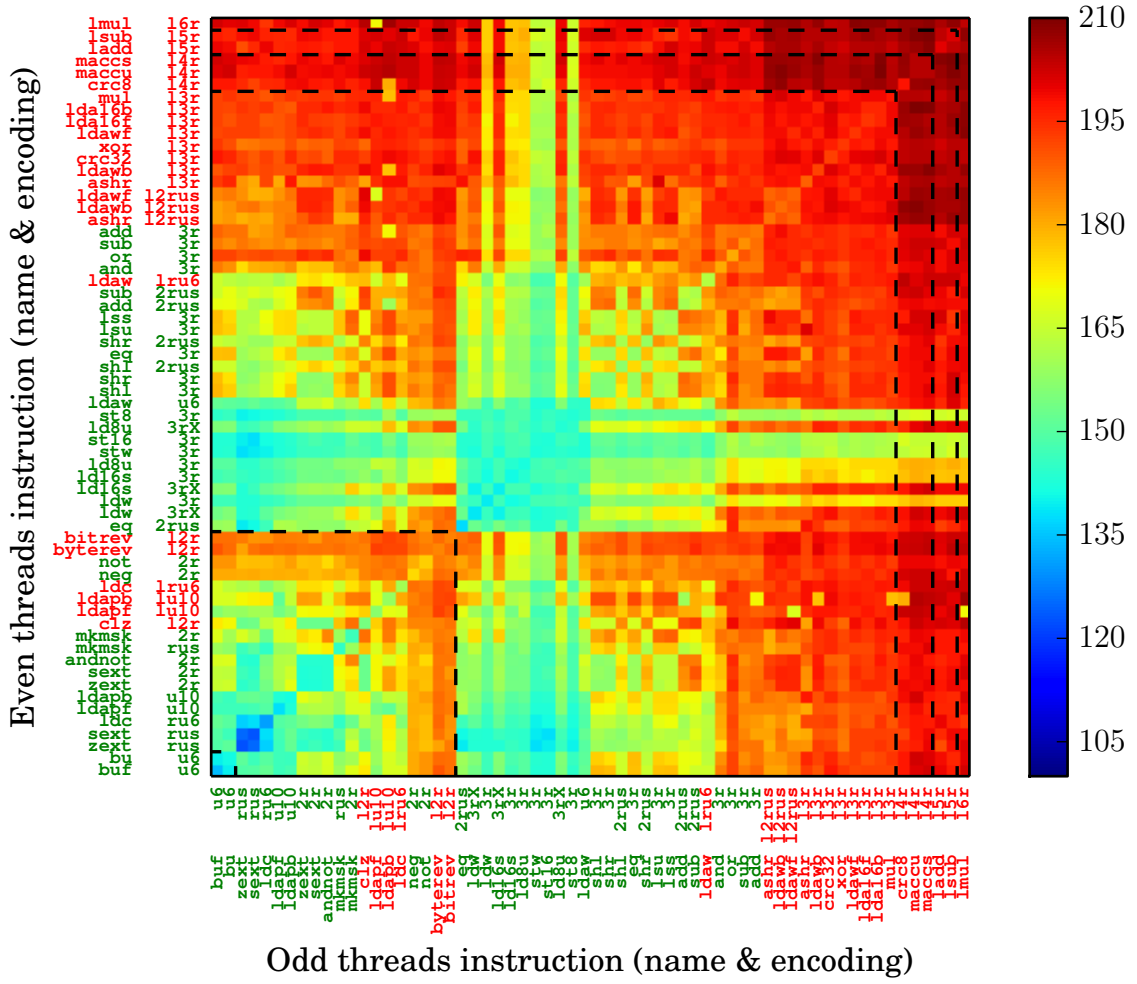


Figure 7.8: Extended profiling energy data. Green instructions are 16-bit, red are 32-bit. Memory operations appear to consume little energy due the presence FNOPs in their test runs.

The branch instructions are low power as they perform very simple computation, simply changing the next fetch target. Although the instruction buffer is flushed for that thread, there is typically no time penalty for this.

Profiling results for memory operations include the FNOP activity as well. This results in what appears to be lower power dissipation, caused by the processor stalling whilst it fetches further instructions. The lock-step execution of the test kernels results in all threads performing an FNOP concurrently, thus lowing power further. An FNOP is determined to be 10 mW above the base power, and with a ratio of three memory operations for every FNOP, memory instruction costs are then calculated and integrated into the model.

7.4.2. Regression tree construction

Estimating the energy of an unprofiled instruction based purely on its encoding fails to capture some particularly useful information, for example whether the instruction interacts with memory or resources. A more sophisticated estimation should therefore capture this information into a set of characteristic parameters, $\mathbb{F}$, where the energy contribution of each parameter is given. Taking all combinations of values in $\mathbb{F}$ for profiled instructions, Ordinary Least-Squares (OLS) [DL06] linear regression can be used to estimate the contribution each parameter in $\mathbb{F}$ makes towards total instruction energy. The potential benefit of this approach is visible when examining the behaviour of instructions in the larger encoding-based groups. For example 3r and l3r the three-register

encodings, are 16- and 32-bit long instructions respectively. Within them there are memory, arithmetic, logical and address operations.
The distinguishing features chosen for the XS1 instruction set, $\mathbb{F}_{\text{xs1}}$, are:

- Instruction length (short or long: 1 or 2)
- Number of source registers (count: 0–4)
- Number of destination registers (count: 0–2)
- Length of immediate operand (num. bits: 4–16)
- A memory operation is performed (Boolean)
- A resource operation is performed (Boolean)

This subsection first explains a flat, OLS regression against these features and the profiled instructions, then extends this to use a more sophisticated regression tree, where different instruction types can give varying significance to each feature.

OLS regression

For OLS, each feature in the set $\mathbb{F}$ must be expressed numerically. In the case of $\mathbb{F}_{\text{xs1}}$, the first four elements can be a count — number of 16-bit instruction words for instruction length, number of registers for source/destination and number of bits in the immediate value. In the case of the latter two elements, which are either true or false, they can be represented as 1 or 0 respectively.

OLS is calculated using Eq. (7.4), where A is a $m \times n$ matrix of m test cases for n parameters, and b is the vector containing the result for each case, resulting in a vector y of solved coefficients.

$$y = \left(A^T A\right)^{-1} A^T b \tag{7.4}$$

To establish estimates for the features in $\mathbb{F}$, the data is expressed as shown in Eq. (7.5). The resultant y vector contains a coefficient for each element in $\mathbb{F}$. The result vector, b, is the instruction power.

$$\begin{aligned} A &= \begin{pmatrix} \mathbb{F}_{0,0} & \cdots & \mathbb{F}_{0,n-1} \\ \vdots & \ddots & \vdots \\ \mathbb{F}_{m-1,0} & \cdots & \mathbb{F}_{m-1,n-1} \end{pmatrix} \\ b &= \begin{pmatrix} P_0 \\ \vdots \\ P_{m-1} \end{pmatrix} \end{aligned} \tag{7.5}$$

From 59 sampled instructions there are 12 unique combinations of $\mathbb{F}$ feature values. Some feature combinations are impossible; for example there are no memory operations using four source registers. Other combinations could not be reached through profiling, as the instructions cannot be repeatedly executed in isolation. The OLS solution using the data from the terms reachable through profiling is shown in Table 7.5. The negative coefficient for immediate length could be considered problematic in isolation, because conceptually it expresses a component of the system as reducing power, or having a negative capacitance, the latter of which is particularly disagreeable. However, provided the sum of all parameters remains positive as in Eq. (7.6), the model remains sound, as each parameter cannot be considered a direct representation of a physical component, therefore the laws of physics are not violated.

$$\sum_{i=0}^{n} \mathbb{F}_i y_i > 0 \tag{7.6}$$

$\mathbb{F}_{\mathbf{xs1}}$ feature	Coefficient
Instruction length	32.7×10^{-3}
Num. source operands	10.9×10^{-3}
Num. destination operands	8.34×10^{-3}
Immediate length	-770×10^{-6}
Memory operation	26.3×10^{-3}
Resource operation	6.01×10^{-3}

Table 7.5: Solved coefficients for features, $\mathbb{F}_{\text{xs1}}$, which can be used to determine the cost of unprofiled instructions.

Regression tree

A "flat" linear regression does not necessarily capture estimates for unprofiled instructions in the most accurate way. Some features in $\mathbb{F}$ have different relationships to each other depending on their value. For example, arithmetic instructions use their source operands differently to resource or memory instructions.

To provide refinement that accounts for these co-variances, a regression tree is used. Regression trees are a form of decision tree [Bre+84]. They are similar to classification trees, which seek to group data based on a training set. Regression trees, however, assume that a continuous output is required, rather than a discrete classification.

The tree is calculated using the same inputs as an OLS regression. The decision tree is binary and the ordering of features for selection as well as the value against which a decision is made can vary along each path. This is demonstrated in Figure 7.9, where the features $\mathbb{F}$ are elements in the list X. The renaming to X is a by-product of the regression tree implementation, for which the Scikit-Learn Python library is used [Sci15].

Examining Figure 7.9 we see that long instructions with greater than one source operand (taking the right branch of the decision tree twice) have higher power dissipation than those which do not. In all cases, the first decision is made with $X[1]$, the number of source operands, then at the next level $X[0]$, the instruction length is used. Subsequent stages have divergent decision features depending on the outcome of the previous decision. The regression tree can also use a feature multiple times. At each decision level, the number of samples reduces, until a leaf produces a value that minimizes the mean-squared error for that set of feature parameters. Not all features are necessarily used, as some may have no relation to the output. With different numbers of decisions made along each path, this can result in an unbalanced tree. With this tree, all instructions from the XS1 ISA can then be assigned a cost by traversing the tree using their particular features.

7.5. Evaluation of the extended model

The completed model is tested using the same general method as for the preliminary models, described in § 7.3.1, with some modifications that take into consideration forward requirements for the model. The hardware measurement process and the benchmarks that are used (Table 7.4) are the same, however.

The two key differences are the use of a trace-based simulation model and improvements to the analysis that allow a single instance of a test kernel to be modelled rather than multiple runs over a fixed period of time.

7.5.1. Trace simulation

With the previous modelling approach, an ISS produced execution statistics, which were analysed to determine the prevalence of multi-threading and the number of times each ISA instruction is executed by a thread. The main justification for this was speed. The completed core level model substitutes this for analysis of a full execution trace. The choice of simulation method and data output was discussed in § 6.3.1, where it was established that the trade-offs between detail and performance would need to be evaluated as work progressed. This subsection identifies

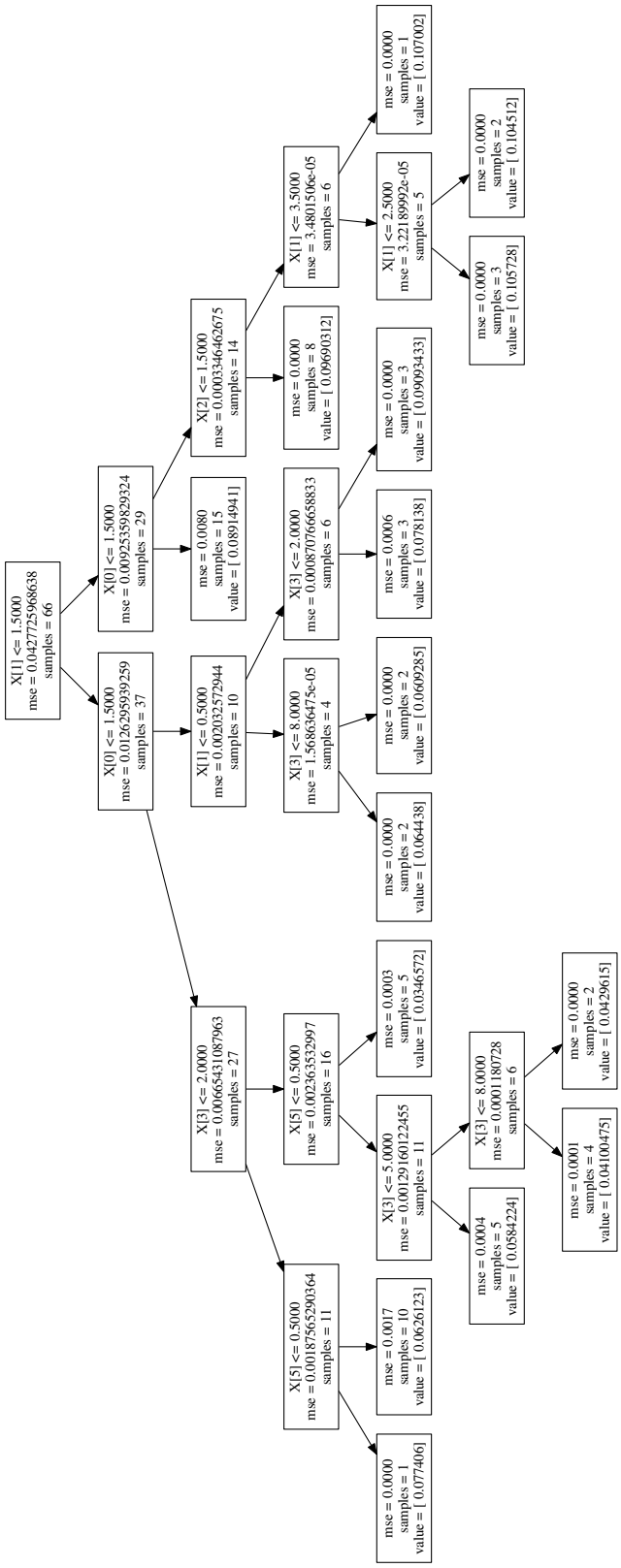


Figure 7.9: Regression tree completing the XS1-L energy model. Each intermediate node shows the decision feature and its threshold, the mean-squared-error and number of samples captured at that node. Leaf nodes have no decision parameter, instead providing a power value. A left branch is taken if the condition is true, otherwise right.

the differences between the two methods used in this chapter, including how a number of initial problems with trace based simulation were mitigated, thus enabling its use in the extended model and the multi-core work in subsequent chapters.

The initial reason for not using trace data was performance. Short programs could take two orders of magnitude longer than real time to simulate. Collecting statistics instead of traces sped up this process, and using the `axe` simulator for statistics collection was particularly fast, as previously discussed. However, in multi-threaded programs, the interactions between threads (which threads are active and when) and the overall timing of the program are less reliable when `axe` and statistics alone are used.

In addition to this, trace data improves the debug process. For example, when comparing the model applied at simulation level, to that applied statically (see § 7.6), a trace contains more information to examine than execution statistics, making it possible to highlight where divergence may be occurring.

Thirdly, to consider the model at a multi-core level, the instructions that perform inter-core communication must be observed. The time at which these occur is important to the multi-core network model. Thus, this change enables the further work detailed in Chapters 8 and 9.

In the move to trace based simulation, the performance issue is addressed. This is achieved by improving the accuracy of the `axe` ISS at some cost to performance, whilst still out-performing the original `xsim` simulator. In addition, feature enhancements are made to the `axe` simulator, providing traces that are in a format that lends itself to analysis, making statistics analysis less compelling. The changes to `axe` are explained in more detail in § 9.2.

Finally, the analysis process during modelling is improved, such that the simulation can be terminated after the first completion of specified functions of interest. This reduces error, by ignoring instructions that are purely part of the test harness and significantly reduces simulation time, by allowing an *early exit* from the simulation when compared to running and measuring for 0.4 s on the hardware, which would otherwise require potentially thousands of iterations to be simulated.

In summary, these changes, in response to requirements both in other parts of this thesis and in research external to it, necessitate a departure from an analysis centred around statistics, to that of one using instruction traces. These changes incorporate improvements to performance that mitigate the problems leading to the original motivation for statistics based analysis.

7.5.2. Results

Figure 7.10 shows the results of benchmarking with the completed core model. The results are presented in the same format as Figure 7.6, where Figure 7.10a depicts the estimated and actual energy consumption recorded for each benchmark and Figure 7.10b compares the accuracy of the models for each test.

The improved modelling approach, which measures only a single run of each benchmark, is not directly comparable to the energy values provided in Figure 7.6a, as these accumulate the energy over 0.4 s for both hardware and the model. In the improved approach, the hardware energy is determined with respect to a single test kernel iteration by dividing the total energy consumption during the test by the number of times the kernel is executed. The only exception to this is the idle test, which has a fixed run time in both cases. To provide a useful scale in the plot, this test is removed from Figure 7.10a. The error margins in Figure 7.10b include both the original instruction and grouped models, plus the new regression tree based model, for a full comparison and evaluation.

The new model demonstrates an average error of 2.75 %, and a tighter standard deviation, which delivers a strong improvement over the previous models. This incorporates benefits from the regression tree and from additional custom tests to provide more accurate characterisation of memory operations. The worst case error across the benchmarks is less than 10 %, keeping the model within the desirable error margin previously identified in § 3.3. In addition to a tighter error margin, the standard deviation of the error is reduced to 4.61 percentage points, from 7.22 and 7.80 for the grouped and individual instruction models respectively.

The increased complexity required to perform a trace-based ISS is offset by the early-out of the analysis. This means analysis of trace-based simulation, including the simulation itself, typically

Model version	Error (%)	Std. dev. (%)
Original grouped instructions	−16.42	6.91
Original instruction-level	−7.23	7.45
Extended + regression tree	2.67	4.40

Table 7.6: The model error determined from the geometric mean of accuracy, $1 - (\prod_{i=1}^{n} x_i)^{\frac{1}{n}}$, for the three evaluated techniques. Results are presented as percentages, where 0 % is a perfect representation of the hardware energy consumption. The standard deviation is also presented, in percentage points.

completes within less than 10 seconds for the benchmarks used here. This is good when compared to the 40 seconds for a full statistics based simulation performed in `axe`, which simulates a real-time period of 0.4 s. Although `axe` is used for simulation, the slower `xsim` would also benefit from significantly reduced simulation time, using the new improvements to the analysis process.

7.6. Beyond simulation

The efficacy of the multi-threaded energy model described in this chapter has been shown for two possible simulation methods. The data underpinning the model can be used in other contexts as well.

Instruction selection

One possible, but as yet unrealised used, is to aid compiler optimisation. For example, an instruction or series of instructions may have several equivalent forms and equal performance. However, if combined with a cost-function based on the energy of the candidate instruction sequences, a particular sequence may emerge as the most desirable selection. This assumes that the equivalent sequences are relatively straight forward to determine, rather than the subject of a process similar to intensive superoptimization [Mas87].

An example of this is in [Ste+01b], where memory operations are reduced through register pipelining, but additional instructions must be executed for this to take place. Data from an energy model can be used to establish the costs of these trade-offs in order to decide whether to carry out such optimisations.

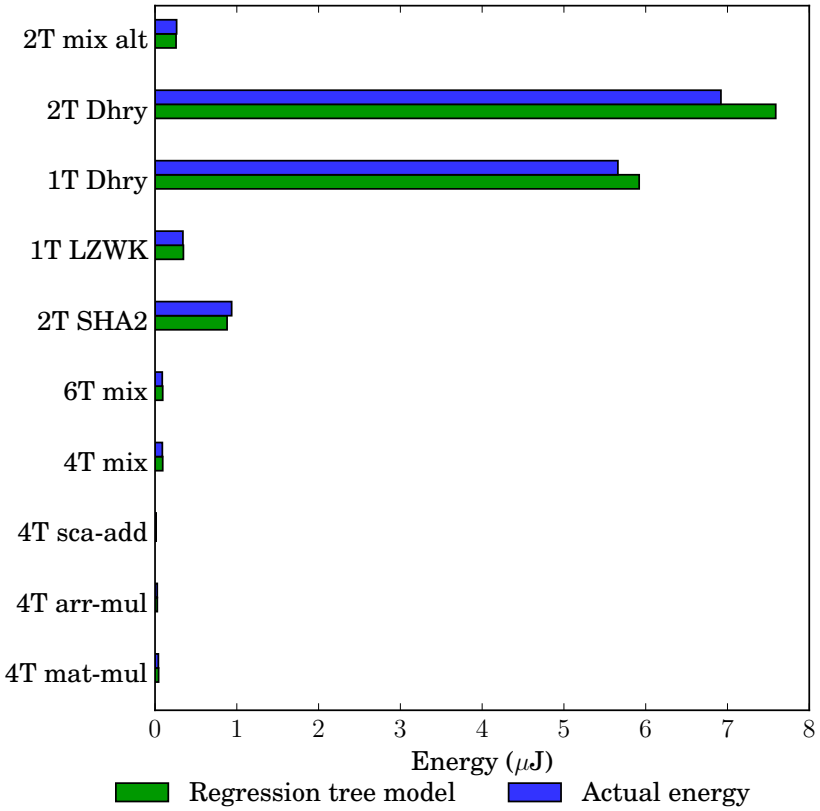
Static analysis

Shortening the process between compilation and energy consumption estimation, static analysis eliminates the need to simulate execution, instead providing an alternative characteristic representation of what a program *would do* if executed. This can then be used to provide energy estimates.

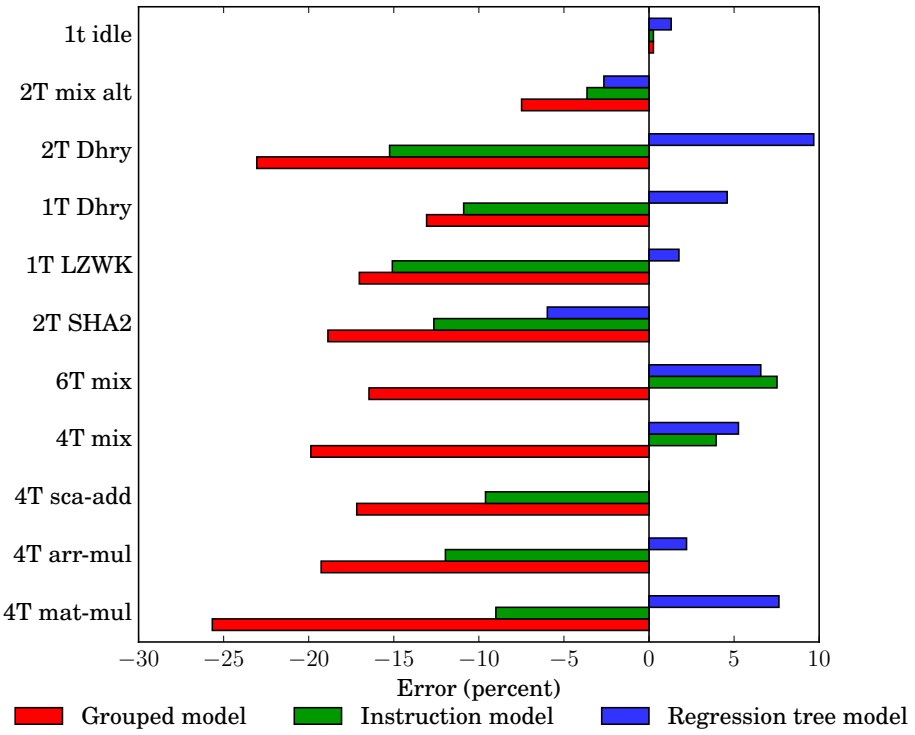
Typically, a static analysis must make certain assumptions about a program's expected behaviour in order to give a range, or bounded result. Static analysis has been used for timing analysis, for example Worst Case Execution Time (WCET), where a more accurate analysis brings the upper bound for execution time closer to that which would be observed during a typical program run. In a real-time embedded system, an assurance that a timing constraint can *never* be broken may be essential. However, from an energy perspective, the average, or typical case may be acceptable.

Several works looking into combining energy modelling and static analysis have already been made. More significantly, the energy models proposed in this thesis form the basis for some of these works.

In [Liq+15], the CiaoPP framework is used to perform resource analysis on a series of functions, establishing an energy consumption bound that can be expressed with respect to the input parameters of the target function. For example, the function `fibonacci(N)`, which calculates the Nth number in the Fibonacci sequence, consumes an amount of energy dependent upon the value of N. XC programs are compiled into XS1 assembly. These are then transformed into Horn Clauses



(a) Results of benchmarks for the regression model vs. actual device measurement.



(b) Accuracy of the three model approaches compared to observed device energy.

Figure 7.10: Completed model benchmark results.

that can be analysed in the Prolog based Ciao tool. The resultant cost functions are built from this analysis combined with the instruction costs provided by our XS1-L energy model.

Energy consumption analysis of the Low Level Virtual Machine (LLVM) toolchain's Intermediate Representation (IR) is demonstrated in [Gre+15]. This is performed for code targeting both XS1-L and ARMv7, with the former using our energy model data.

Further static analysis work is also being performed in [GKE14], taking the concept of WCET adapted to energy, giving Worst Case Energy Consumption (WCEC).

In the above cases, this chapter's *average* energy consumption model is used, so the bounds are not as strict as timing based bounds. However, this is more sensible that absolute or pathological worst case energy consumption, which are less likely than worst case timing paths. A possible compromise to this, however, would be to provide an energy model as a probability distribution rather than a single value. Providing such model data is beyond the scope of this thesis, but is the subject of ongoing work.

7.7. Summary

The energy characteristics of a hardware multi-threaded micro-processor architecture differ from a more traditional micro-processor. This chapter and the previous chapter have identified the effects of this upon software energy modelling. These effects were discussed and subsequently accounted for in a new ISA level software energy model.

This chapter has expanded upon prior work that models software energy on single-threaded architectures, taking into account the more complex behaviors present in a hardware multi-threaded and event-driven architecture, such as idle time, active thread count and the interactions between varying numbers of active threads.

It is shown that the energy cost of individual instructions differs by up to 1.7x. Further, inter-instruction overheads are more complex to predict in this type of architecture and thus a barrier to some instruction level energy modelling techniques. However, it is not necessary to consider them on an individual basis, on account of their low significance when compared to factors such as the specific instructions that are executing and the number of active threads.

Using data extracted with `XMProfile`, execution statistics from simulation runs have been used to form an initial energy model for the XS1-L core. This model was evaluated using a set of benchmarks and demonstrated an average error of $-7.23\,\%$ compared to actual hardware energy measurements. An alternative, grouped instruction model was proposed and demonstrated, but the performance benefits are low compared to the loss of accuracy of several percentage points.

The initial model was applied against execution statistics rather than instruction traces, reducing simulation time by two orders of magnitude. An extended model, enhanced through additional profiling and regression techniques, uses instruction traces rather than statistics, which is intended to allow more flexibility and accuracy in the multi-core work presented in the following chapters. This is achieved by using the fast `axe` simulator, with improvements to its simulation accuracy whilst retaining strong performance. The extended model demonstrates an accuracy of $2.67\,\%$, a compelling improvement compared to the original models and competitive when compared to prior work.

Applications for these models are highlighted through the ongoing work that is using them. A significant portion of the work in this chapter is published in [KE15b].

Future work was proposed, such as how the model can be improved further by refining various terms in the model based on feedback from benchmarks, and by activity in the $3.30\,\mathrm{V}$ domain along with multi-core activities such as communication. Further chapters in this thesis seek to address a number of these points. Finally, the methods presented herein for building an ISA-level energy model of the XS1-L provide a template to aid the construction of energy models for other multi-threaded architectures.

8. Multi-core energy profiling and model design using Swallow

This chapter examines the properties of the multi-core Swallow system that must be understood in order to form an energy model for it. This builds upon the profiling presented in the previous chapters, focusing on communication.

The Swallow boards provide multiple types of communication link that operate at different speeds and have different electrical characteristics. The capacitance of the interconnecting wires contributes to energy consumption [SC00]. For example, the on-chip links between cores have very short bond wires between them that have a different capacitance to the longer, larger tracks connecting chips on the Printed Circuit Board (PCB). These characteristics, combined with a varying number of hops between communicating processors, can be exploited to determine base facts about the cost of communicating over the network.

In high-level terms, the energy consumed by a communication can be considered to accumulate the following:

- The computational cost of issuing any configure, send and receive instructions, consumed by the processor cores involved.
- Communication between the cores and their local switch.
- Transit over the physical links between the communicating cores, on and off-chip.
- Processing cost of the data within each switch through which the message passes.

The sum of these energies gives the cost of the communication itself. However, to adequately account for the full impact of the communication, the latency of communication and energy consumed whilst waiting for communication to take place must also be accounted for.

The remainder of this chapter focuses first upon applying core level modelling across multiple processors, then moves onto profiling the communication costs and other system level properties that affect both energy measurements and model construction.

8.1. Core energy consumption on Swallow

The multi-threaded core model from Chapter 7 remains an essential part of the multi-core model. However, the hardware upon which it is running differs significantly from the single core hardware previously used for profiling and testing. These differences need to be accounted for in order to reconcile hardware energy measurements between the platforms and to identify where additional errors through noise may be introduced.

The main differences in Swallow are:

- Dual-core L2 processor packages instead of a single core L1 package.
- Four 1 V power supplies, each supplying two packages.
- Power sensing instrumented on the 3.30 V output and 5 V output.
- Probe points for measuring the 1 V supply powers, but *not* instrumented for automated sensing.

In § 5.2 more detail is given on these and other differences.

Some of these characteristics, such as sensing on the larger power supplies, allow the measurements taken in Chapter 8 to be performed. However, they also create layers of obfuscation between

the power dissipated in each core, due to the measurement points, power supply efficiency, and the inclusion of other components in measurement, such as board-level clock trees and other signal chains.

To estimate the power at the 1 V supplies, the power delivered to them from the 5 V supply is determined in Eq. (8.1a). This must account for the 3.30 V power and its efficiency, η_{3v3}. η_{1v} is calculated by manually measuring the 1 V output power with a multi-meter for a pair of tests, while η_{3v3} is estimated at 80 % based on the consistent output current observed and inspection of the device datasheet. The power supplies used for both 3.30 and 1 V are NCP1529 devices [ON 10].

Table 8.1 provides a list of measured powers and calculated efficiencies. Comparing the efficiencies to the datasheet, the 1 V supplies are less efficient than stated, but by a small and consistent margin that suggests the method of calculation is sufficiently robust. The power supply efficiency is close to linear in the region of operation that all tests are applied, therefore η_{1v} can be approximated as a linear function of input power, giving Eq. (8.1b) to determine P_{1v}.

$$P_{\text{in\_1v}} = P_{5v} - \frac{P_{3v3}}{\eta_{3v3}} \tag{8.1a}$$

$$P_{1v} = 0.751 \times P_{\text{in\_1v}} - 0.0658 \tag{8.1b}$$

Term	Value	
	Idle	WC lmul
3.30 V output power	177×10^{-3} W	
3.30 V efficiency η_{3v3}	0.8	
5 V power	2.49 W	4.94 W
1 V output power	1.64 W	2.94 W
1 V input power	2.27 W	4.00 W
1 V efficiency η_{1v}	0.723	0.623
1 V efficiency per datasheet (approximate)	0.76	0.66

Table 8.1: Calibration tests for Swallow vs. L1 profiling board.

To test the robustness of the core energy model when applied to Swallow, a set of three tests is performed. These tests are run from one active core up to all sixteen. The tests are:

- An idle test, analogous to the base processor cost test performed for a single core, *idle*.
- A four threaded workload of `add` instructions with 32-bit random input data, *random add*.
- A four-threaded stress test of `lmul` instructions with worst case data (§ 7.2.5), to maximise core power dissipation, *WC lmul*.

The estimated power is determined through Eq. (8.2). $\mathcal{F}$ is the function determining P_{1v} as per Eq. (8.1b). For each test, P_{op} is the model power for the particular activity (e.g. add or worst case lmul) and P_{base} is the core base power established in § 7.2. The number of used cores is N and all unused cores are considered to contribute P_{base} to the total power dissipation.

$$P_{\text{model}} = \frac{P_{3v3}}{\eta_{3v3}} + \mathcal{F}\left(N \times P_{\text{op}} + (16 - N) \times P_{\text{base}}\right) \tag{8.2}$$

The results of the Swallow core tests are presented in Figure 8.1. The idle power is flat, regardless of the number of cores that are utilised. In JTAG boot, all cores must run start-up code in order to configure their local switch, then they become idle. Therefore, each core contributes P_{base} regardless of whether a test is explicitly loaded onto it. The remaining tests increase in power consumption linearly with the number of cores. The geometric mean error is −0.5 %, however the worst observed error is 5.2 %.

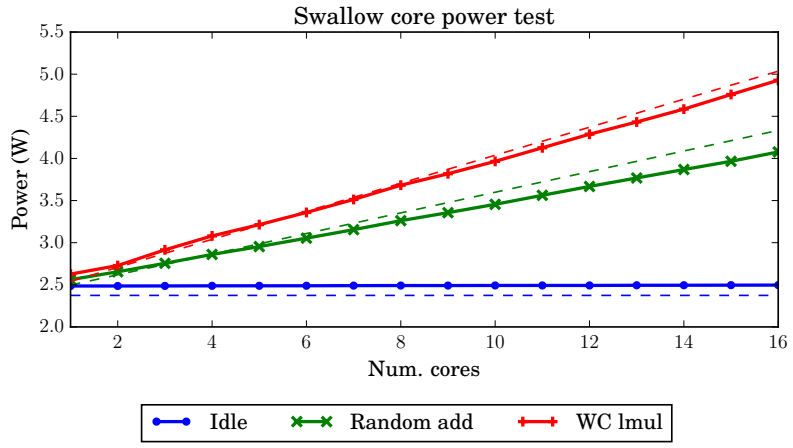


Figure 8.1: Power dissipation on Swallow for 1–16 cores, testing idle, four-threaded random addition and worst case `lmul`. Model predictions are shown as dashed lines.

The Swallow board has some temperature sensitivity, with initial idle tests showing 5 V power dissipation of 2.42 W, increasing to 2.48 W after further testing, stabilising at that level. A high power `lmul` test, when run over a 15 minutes period, demonstrates a power variation of 150 mW, fitting a concave upwards exponential trend with $R^2 = 0.993$. The power increase over time for this test is shown in Figure 8.2.

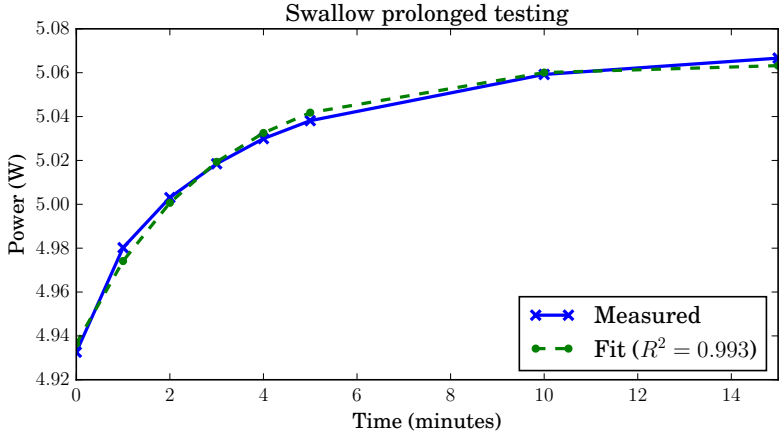


Figure 8.2: Power dissipation during a high-activity `lmul` test, observing an increase in power due to temperature.

The temperature sensitivity is present in both the power supplies and the cores, this particular test demonstrating the worst case. Due to these effects, a model that does not account for temperature may incur an additional error margin over time of up to 7 %, based on conservative observations during testing. Modelling this behaviour does not serve this thesis' goal of enabling analysis at levels above simulation. This work should evaluate the model accuracy giving consideration to this, to assess if the additional error prevents higher level modelling from being useful to the software developer.

8.2. Network communication energy profiling

The communication tests leverage the `XMProfile` energy monitoring software, modified to measure the 5 V and 3.30 V supplies to a Swallow board. This gives energy consumption for the whole board

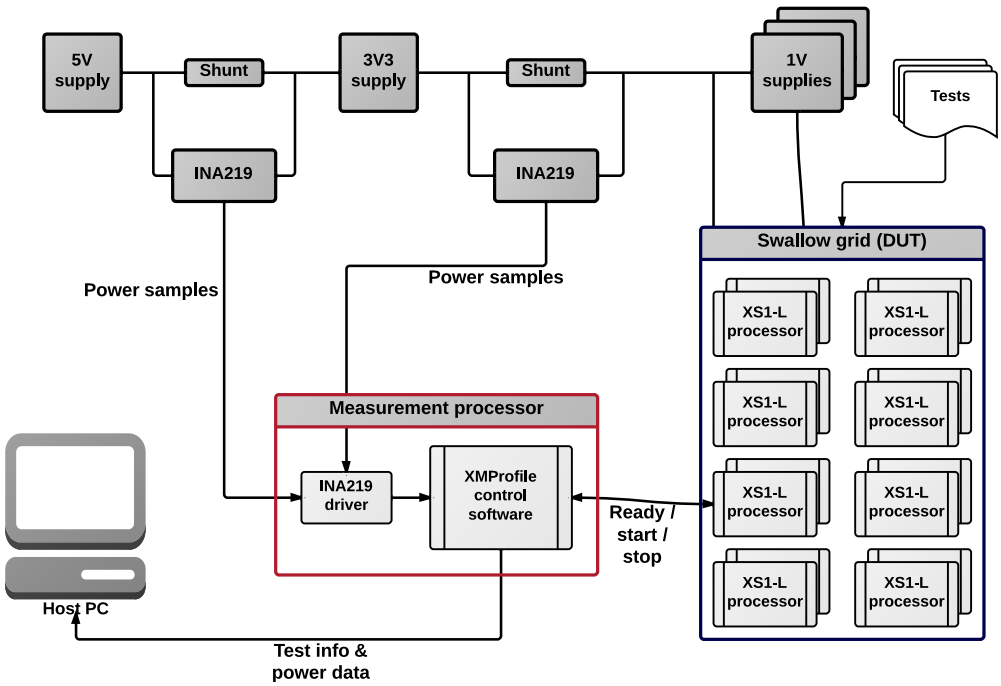


Figure 8.3: Experimental setup of the Swallow hardware and measurement apparatus.

and the board's I/O, respectively. From this, an approximate cost for the 1 V cores can also be extrapolated, using Eq. (8.3), which considers total board power and the power supply efficiency, of the board's 1 V supplies, η_{1v}. This cannot provide as accurate 1 V figures as the dedicated test setup, but can be useful for checking that the results are within expectations.

$$P_{1v} = \eta_{1v} \left(P_{5v} - P_{3v3} \right) \tag{8.3}$$

8.2.1. Physical setup

The test setup shown in § 8.2.1 is similar in structure to that of Figure 6.2. The DUT is now a grid of XS1-L processors instead of a single core. Further, the control path between the measurement processor and the DUT is now a set of GPIO and not an X-Link as used in the single-core setup. Finally, two INA219 sensors are used to sample the 5 V and 3.30 V power supplies.

This configuration requires that software for measurement and software for test are loaded separately, as there is no JTAG chain or X-Link network between the DUT and measurement processor.

8.2.2. Software setup

The 16 cores of the Swallow board are programmed with tests using a combination of `XC` and XS1-L assembly. In a top-level `main` function, tasks are allocated to cores as shown in Listing 8.1. Cores, or *tiles* are indexed numerically. Channels can be declared as `chan` variables or arrays and must be passed to two functions, connecting each *channel end* to form a point-to-point link. The full capabilities of a multi-core top-level main function are described in [Wat09, pp. 29–30].

Using this method, multiple tests are loaded onto sets of cores. One thread on a specific core synchronises the test operations with the test harness, using Swallow's available GPIO (§ 5.2.1) to communicate readiness and test progression with the measurement processor. Each set of tests will wait on a channel end for a start signal before proceeding.

In addition to the tests, all cores, regardless of their participation in the tests, are configured to lower their clock speed when unused, thus lowering the total power dissipation of the Swallow grid and reducing the impact of heat upon the system.

```
int main(void) {
    chan c;
    par {
        on tile[2]: foo(c);
        on tile[8]: bar(c);
    }
}
```

Listing 8.1: XC top-level multi-core allocation example.

8.2.3. Description of tests

The tests used aim to maximise throughput and thus power dissipation. A series of 8192 packets, containing 4000 bytes of random data are sent from one thread to another. The data is sent in 4-byte words, meaning the XS1 ISA instructions `out` and `in` are used [May09b, p. 69, 123], achieving maximum possible throughput vs. processor time.

This test is repeated several times, each time the location of the communicating threads is changed in order to exercise a different communication link or set of links. The combinations include tests of each type of link individually as well as purely local communication on the same core, and multi-hop communication with several switches and types of link traversed. Table 8.2 lists the combination of tests and their network utilisation. Each test uses a switch for every node involved, hence communication that uses three links involves four switches.

Test ID	Description	Cores	V hops	H hops	L hops	Switches
A	Local	1	0	0	0	1
B	Layer (same chip)	2	0	0	1	2
C	Horizontal	2	0	1	0	2
D	Vertical	2	1	0	0	2
E	1 hop in each direction	2	1	1	1	4
F	2 vertical hops	2	2	0	0	3
G	3 hops layer & vertical	2	1	0	2	4
H	4 hops, all directions, 2 layer	2	1	1	2	5
I	6 hops, all directions	2	3	1	2	7
X	*Idle*	—				

Table 8.2: Test combinations for communication power measurements.

8.3. Determining communication costs

The tests described in § 8.2 yield the results shown in Figure 8.4. For the 1 V power calculations, $\eta_{1v} = 0.92$ is used in Eq. (8.3), estimated from the power supply datasheet [ON 10].

Although the I/O power is significantly smaller than the core power, it is observed that the former changes in proportion to the number of network hops taken, whereas the latter is more dependent on the number of awake cores. The physical links operate at 3.30 V, so for modelling purposes, link communication cost only considers the 3.30 V I/O power.

Even at idle or with core-local communication, a significant amount of 3.30 V power is present. This can be attributed to the clock tree, which spans from the oscillator on the board, through various buffers and into each chip, as well as constant power dissipated by various 3.30 V components in the system. It is worth noting, however, that all LEDs were switched off during tests, as these would have skewed readings even further.

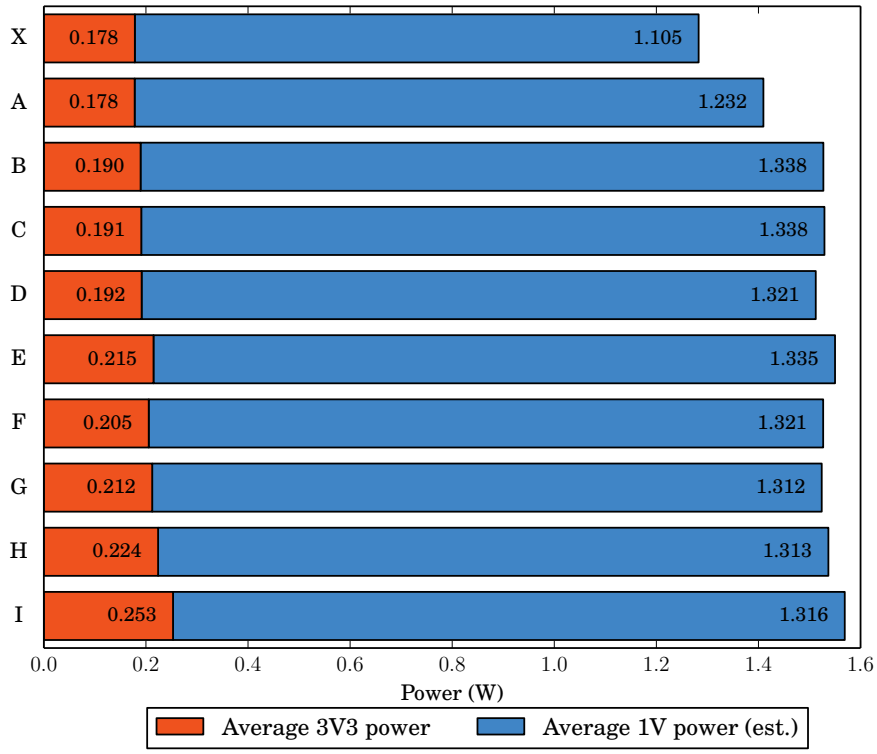


Figure 8.4: Communication power test results, Swallow board, 500 MHz cores, 500 Mbps on-chip links, 400 Mbps off-chip links.

Examining these results, test I, which uses the largest number of links (six), dissipates the most I/O power. Removing the baseline power of 178 mW to give 75 mW, it dissipates approximately twice as much I/O power as test G, which uses three links and dissipates 34 mW. A correlation between number of hops and 1.00 V core power can also be seen, although it is small. This can be attributed to the increased number of switches involved in routing a message. Provision for this can also be made in the model, which will shown in § 9.3.1. The data meets expectations, based on how the Swallow system constructed and the architecture of its processors.

This data is used to extract estimates for the switch power and the link power in each of the three directions, where the 3.30 V power at idle and single hop tests are used in Eq. (8.4) for the directions layer, horizontal and vertical as l, h and v respectively.

$$\begin{aligned} P_{\text{hop}} &= P_{\text{test}} - P_{\text{base}} \\ \begin{pmatrix} \text{l} \\ \text{h} \\ \text{v} \end{pmatrix} &= \begin{pmatrix} 190 \times 10^{-3} \\ 191 \times 10^{-3} \\ 192 \times 10^{-3} \end{pmatrix} - 178 \times 10^{-3} \\ &= \begin{pmatrix} 11.2 \times 10^{-3} \\ 12.6 \times 10^{-3} \\ 13.3 \times 10^{-3} \end{pmatrix} \text{W} \end{aligned} \tag{8.4}$$

Link power can be translated into a dynamic power equation, of the form, $P = CV^2$, given that V is known. This gives a simplified model of the real world behaviour of the transitioning link wires. This approach will be used in Chapter 9.

The multi-hop tests are used to validate these costs, giving an error range of $-5.84\,\%$ to $+2.25\,\%$, shown in Table 8.3. These costs, which focus upon the 3.30 V domain, represent any I/O voltage dissipated by the switches and the wires or tracks connecting them. This is further broken down

Test ID	Num. hops	Power (mW)		Error (%)
		Measured	Estimated	
E	3	36.6×10^{-3}	37.2×10^{-3}	−1.61
F	2	27.2×10^{-3}	26.6×10^{-3}	+2.25
G	3	34.0×10^{-3}	35.8×10^{-3}	−5.01
H	3	45.6×10^{-3}	48.4×10^{-3}	−5.84
I	6	74.8×10^{-3}	75.0×10^{-3}	−0.28

Table 8.3: Communication cost validation, model vs. measurement and error percentage.

in the following section, to produce data compatible with the modelling method. As one might expect, a 6-hop communication on the Swallow grid dissipates approximately six times more power than a 1-hop communication, after subtracting P_{base}.

Considering time

An energy model that exploits this profile data must also consider transmission time. Link speeds will not affect dynamic energy consumption because the same number of transitions will occur. However, the system as a whole may consume more energy whilst waiting for communication to take place. Link speeds are discussed in § 5.2.2 and will be given further consideration in the simulation activities conducted in Chapter 9.

8.4. Summary of Swallow profiling

The profiling of Swallow was conducted to serve three purposes. First, to establish the costs of communication between XS1 processors. Second, to validate the single-core model from Chapter 7 in a multi-core environment, where additional characteristics such as power delivery must be considered. Third, to identify new behaviours that may affect performance of the core model and the new multi-core model.

A measurement framework was presented that builds upon the principles of `XMProfile`, applied within the more complex Swallow system, where multiple power sources and many cores must be measured and programmed. Through this new framework, it was possible to determine power supply efficiencies and factor these into multi-core energy predictions.

The cost of communication was demonstrated in § 8.3, where the different interconnect lengths in each of the network's direction of travel have their own power associated with them. A simple model of this communication power was then proposed, to allow the power of communication over multi-hop, multi-dimensional routes to be considered in a system level model. The maximum error observed in this model was −5.84 %.

Using replicated single-core tests, the core model was then validated in a multi-core environment, showing a worst case error of 5.2 %. This demonstrates that the core model can be adapted to a multi-core system level, providing the foundations upon which a communication aware model can be assembled. Sensitivity to temperature was shown, indicating that over time, additional model errors of up to 7 % could manifest.

The work from this chapter will now be used in Chapter 9 to implement and evaluate a multi-core energy model, comprising ISA simulation, including core and network costs.

9. Implementing and testing a multi-core energy model

In a multi-core system, the interconnection between cores, both in physical terms and those defined by the software, must be considered. This chapter establishes a model that can capture the movement of data between cores in a message passing system. It retains a close relationship to the software, building upon the ISA level modelling presented in Chapter 7.

The multi-core model is capable of modelling arbitrary sized systems of XS1-L processors. The Swallow platform was used in Chapter 8 to obtain communication power costs and so continues to be the subject of study in this chapter. A demonstration of a communicating program is shown and evaluated with respect to absolute accuracy and visualisation of consumption data.

In producing this model, two of the research statements from § 1.1 are addressed. Firstly, the proposed system level, network aware energy model enables the cost of data movement to be modelled. This can then be presented to the software developer in order to give them insight into how communicating is taking place in the system and therefore where energy is being consumed. Secondly, the expectation of absolute accuracy and the usefulness of relative observations are evaluated. This chapter's model encompasses a significantly larger and more complex system than the modelling proposed in Chapter 7. This increases the potential for error in both the simulation and modelling processes. As the error margin grows, the utility of the model is examined.

This chapter is structured as follows. A workflow is defined in § 9.1 that builds upon the flow that was presented in § 7.1. The implementation of timed network communication simulation is presented in § 9.2. The profiling data from Chapter 8 is integrated into a network-level energy model in § 9.3. A demonstration is performed and evaluated in § 9.4. An extension of the current system level model to include arbitrary I/O devices is discussed in § 9.6. Finally, § 9.7 summarises the work presented in this chapter.

9.1. Workflow

The workflow for the multi-core component of this research is built upon that of the single-core tools as described in § 7.1. As such, Figure 9.1 is a modification of Figure 7.1 that introduces multi-core energy modelling capabilities.

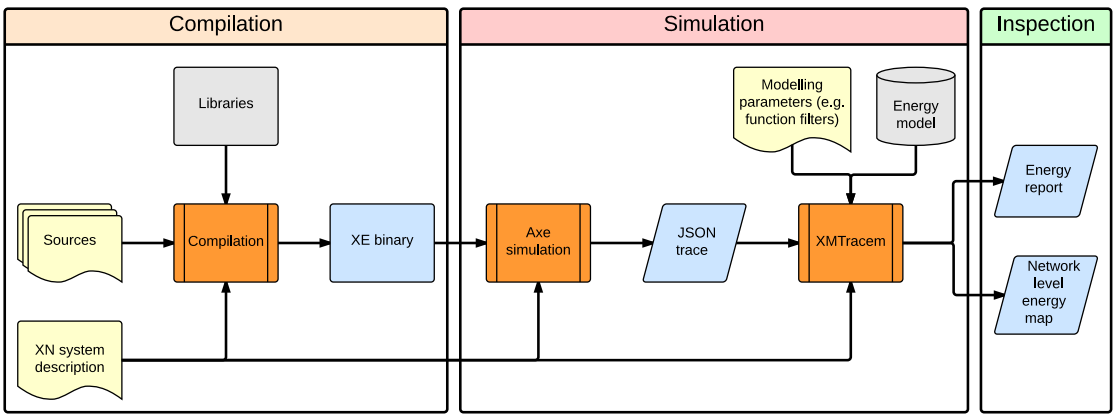


Figure 9.1: `XMTraceM` workflow for a multi-core XMOS system.

The XN system specification file that accompanies a piece of software is integral to the multi-core modelling process. The network structure of a multi-core XMOS system is defined in the XN file, including topology and link parameters. Thus, the XN data is required by `XMTraceM` to model the system at a network level and relate clock cycle information from the simulator to real-world time consumption. The XN file is embedded in the XE binary produced by the compiler and so implicitly available to the simulator. However, `XMTraceM` is supplied with it explicitly for simplicity of file handling. The XN file is depicted explicitly as an input for both simulation and modelling for clarity.

The simulation stage is further refined, with `axe` now the only choice of ISS. The `axe` simulator was modified to provide a network-level timing model and a JavaScript Object Notation (JSON) based trace format, neither of which were available in the closed-source `xsim` that is bundled with the XMOS toolchain.

Finally, the output of `XMTraceM` also includes a network level energy map, where the energy consumption of components in the network can be visualised. This includes cores and switches, along with the possibility to do the same for interconnects and peripherals.

The following sections of this chapter explain the multi-core modelling developments and provides more in-depth explanations of these changes.

9.2. Core and network timing simulation in `axe`

The standard implementation of `axe` is optimised for fast simulation of XS1 programs. It includes features such as Just In Time (JIT) compilation and is flexible with its approach to scheduling execution of threads.

In order to provide a more robust simulator to provide data for a multi-core energy model, the version of `axe` [Ker12a] used in this work contains a number of major modifications.

Stricter thread scheduling

Typically, `axe` will execute instructions from threads in batches, provided there are no dependencies between runnable threads. This behaviour is modified for the version used in this thesis, so that scheduling takes place at the execution of every instruction, thus returning the simulation model to the true round-robin method observed in the XS1-L hardware.

This is necessary in order to effectively use the threading aware energy model defined in Chapter 7. As a result, the timestamps within traces output by this version of `axe` can only progress forwards, whereas previously this was not the case.

Tracing of `FNOP`s

The `FNOP` is important for correct timing and accurate energy modelling. `FNOP` behaviour and the conditions in which is occur was described in § 5.1.1. An `FNOP` model is introduced into `axe` that correctly emulates the fetching and instruction buffering present in the XS1-L. The model was verified by comparing traces to the vendor's official cycle accurate core simulator, `xsim`.

Trace output as JSON

A JSON output format is easily imported into the other Python based tools used in the energy modelling and it is more structured than the human-readable standard trace output. JSON was chosen in preference to Value Change Dump (VCD), despite VCD's common use in lower-level simulators. This is because JSON is trivial to import into the other tools developed in this thesis and the output is compact and still reasonably human readable. A sample trace line is given in Listing 9.1 and the elements detailed in Table 9.1.

The traces contain more data than is currently used by the model. Using this trace data, there are opportunities to extend the modelling in various ways. For example, a Steinke style of ISA model [Ste+01a] could be used that considers switching in the data path as well as the instruction. Statistics on the data characteristics, such as width or number of bits set, should a more data-oriented model be sought. Currently, the function name can be used to apply energy modelling

Name	Type	Description
coreID	int	Network ID of core.
coreName	str	Core name when referenced from XC language, e.g. tile[0].
thread	int	ID of this thread.
nActive	int	Number active threads at this time instance.
fn	str	Name of current function.
fnoffset	int	Byte offset into the function.
pc	int	Program counter value for this thread.
fnop	int	Non-zero indicates the time (clock cycle) at which an FNOP precedes this instruction.
ibuf	int	Number of instructions in the thread's instruction buffer.
time	int	Time (clock cycle) at which this instruction executed.
size	int	Size of instruction (2 or 4 bytes).
instr	str	Instruction name, in the form NAME_encoding, e.g. ADD_3r.
imm	int	The value of the immediate operand, if present.
src	lst(int:int)	List of source register numbers and their values before write-back.
dst	lst(int:int)	List of destination register numbers and their contents before write-back.
write	lst(int:int)	List of registers that were updated during write-back and their new values.

Table 9.1: The elements contained in each line of a JSON trace produced by the modified axe emulator.

```
{
  "coreID": 0, "coreName": "tile[0]", "thread": 0, "nActive": 1,
  "fn": "main", "fnoffset": 42, "pc": 65578, "fnop": 0, "ibuf": 1,
  "time": 77, "size": 2,
  "instr": "ADD_3r",
  "imm": null,
  "dst": [ {"11": 8192} ],
  "src": [ {"11": 8192}, {"10": 65536} ],
  "write": [ {"11": 73728} ]
}
```

Listing 9.1: Example JSON trace line from axe, pretty-printed for improved readability.

only to certain functions or patterns of function names. Energy consumption of each function could also be presented by using this data.

Network delay modelling

Although axe models key parts of the XS1-L's network architecture, including routing and circuit allocation, it does not model timing or credit-based flow control. The modified axe introduces timing into the network simulation, based on a number of parameters:

- Symbol and token rates, defining link speeds, as per the XN platform specification file for a given system.
- Header overheads upon first use (opening) of a circuit.
- Delays introduced by the intermediate switches along a route.
- Buffering of messages when the receiver is full.
- Bandwidth limiting of network tokens governed by the slowest link in a route.

The changes are implemented by including a delay component to simulation for each 8-bit token, which sets the receipt time of a token based on the above parameters. The implementation centres around three main conditions. The simulator determines the route between two channel ends by examining the switch routing tables in line with § 5.2.2, and the transmission speeds of all the links along the route are examined. This then determines D_{tok}, the number of cycles that it will take for a token to traverse the network. At the same time, R_{tok} is calculated, which determines the sustainable transmission rate of tokens, governed by the slowest link along the route.

At the start of a communication a circuit must be opened to the destination. A three token header is sent through the network, its transmission time, T_{hdr}, defined in Eq. (9.1a) will be added to the transmission time of the first data token sent by a channel end. This includes the switch processing delay, D_{switch}, that is incurred when opening the circuit, which is applied for each of the N_{hops} hops along the route.

Eq. (9.1b) represents the transmission time when the receiver is clear to receive. The arrival time at the receiver, T_{rFree}, is the local time T_{local} plus the transmission delay and the transmission rate for the number of tokens being sent, N_{tok}. If T_{rFree} is before the last recorded arrival of a token at the destination, then an alternative method is used, expressed in Eq. (9.1c). In this case, the receive time, T_{rBusy} will be the remote's last token receive time, T_{rRec} added to the transmission rate multiplied by the number of tokens. The ISA contains instructions for single and four token send / receive, therefore N_{tok} is either one or four.

$$T_{\text{hdr}} = 3 \times R_{\text{tok}} + D_{\text{switch}} \times N_{\text{hops}} \tag{9.1a}$$

$$T_{\text{rFree}} = T_{\text{local}} + D_{\text{tok}} + (N_{\text{tok}} - 1) \times R_{\text{tok}} \tag{9.1b}$$

$$T_{\text{rBusy}} = T_{\text{rRec}} + N_{\text{tok}} \times R_{\text{tok}} \tag{9.1c}$$

These changes provide an XS1-L network model that can be used in conjunction with an energy model. However, it does not provide a perfect representation of the real system. In particular, credit based flow control is not implemented and the simple buffering representation can reduce the timing accuracy of the simulation. Further, the switch delay, D_{switch}, is not in any device documentation. For this thesis, it is determined by empirical measurement under a range of test patterns, but may not cover all possible configurations faithfully.

Despite these potential shortfalls, the improvements made to `axe` provide sufficient capabilities for the demonstration of a network level multi-core energy model. Further changes to the simulator, or an alternative network modelling approach, will be discussed in § 11.7.

9.3. Communication aware modelling

Elaborating on the communication costs described in Chapter 8, a communication aware model requires power models for the network switches and communication links, in addition to the processor cores. The processor cores, modelled at the ISA level, can be used to identify the initiation of network events. The characteristics of the network must then be modelled in order for the communicated message to be appropriately costed.

The communication model accounts for communication costs in three ways. Firstly, the `in` and `out` instructions that form the channel communication contribute to the core energy consumption. This was already present in the core model from Chapter 7. In addition to this, the energy consumption of the switches and interconnects are accumulated whenever `XMTraceM` identifies a token or sequence of tokens being transmitted within the `axe` ISS trace.

The communication model follows the same basic principles as the core model. Power is determined as a sum of static and dynamic contributions, which are dependent upon voltage, frequency and capacitance, as in Equations (4.1) and (4.2). Where ISA instructions provide a basis for part of the dynamic power, network tokens must be used as a proxy for the dynamic power of the switch and physical links.

The energy of transmission over a link can be characterised by Eq. (9.2). In this case, only dynamic power is considered, where C is the link capacitance, V the signalling voltage and N the number of transitions that occur. Static power will be accounted for in the core model, where the

Object	Attirbute	Notes
All	Energy	The energy accumulated against this object.
Node	Type	Distinguish between processor cores, switches and potentially other devices such as I/O peripherals.
Node (core)	System frequency	Core clock speed.
	Reference frequency	Timer clock speed.
	Oscillator	Input oscillator frequency.
	Core voltage	
	I/O voltage	
Node (switch)	Switch frequency	Typically the same as the core clock speed.
	Capacitance	Representative capacitance of the switch.
	Voltage	
Edge (link)	Length	
	Capacitance	
	Voltage	

Table 9.2: Graph attributes used by the network-level, multi-core energy model.

time is recorded.

$$E = CV^2N \tag{9.2}$$

The XMOS X-link timing properties were described in § 5.2 and link capacitances estimated from the power profiling in Chapter 8, as discussed in § 8.3. For each link that is used, the appropriate C must be chosen from this profiling data. The C choices for each link type are discussed in § 9.3.2. The number of transitions will depend on the number of tokens sent and whether the communication path was already opened, as was addressed in § 9.2.

9.3.1. Multi-core model structure

The target system is a network of cores, switches, peripherals and interconnects. These are modelled using simple graphs, representing interconnects as edges and all other components as nodes.

The implementation of the graph model uses the `NetworkX` [Net12] Python library. `NetworkX` represents graphs as a set of nodes and edges, both of which can have attributes attached. The attributes can be arbitrary pieces of information, or used to give the graph structure, for example node colours or edge weights.

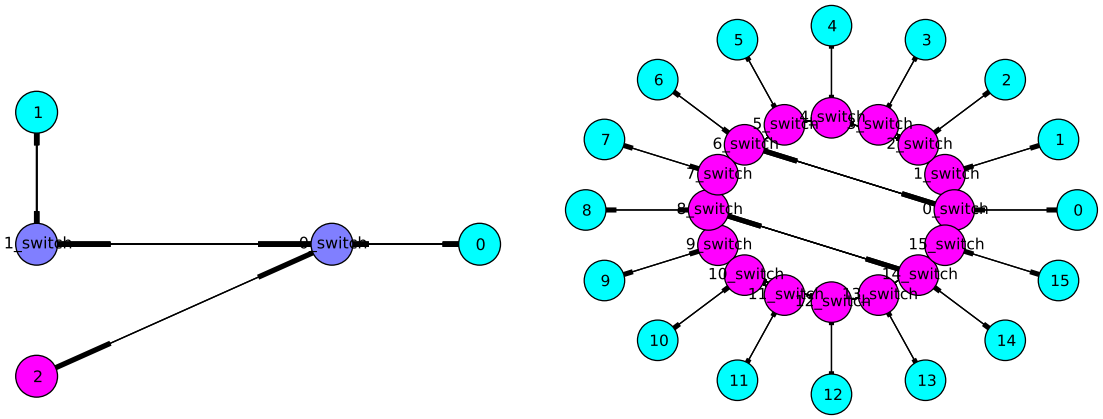
For the multi-core model, a weighted multi-edge directed graph structure is used. This allows the representation of more complex network characteristics. For example the X-Link network can have different outbound and inbound link speeds (requiring weights and directions), and there may be multiple X-Links between two nodes (requiring multiple edges).

Directed, weighted edges represent connections between components. Nodes represent processors, switches or peripherals. The attributes attached to each node or edge depend on the type. Attributes of any kind can be attached to these objects. Table 9.2 details the attributes used in the network level model.

System graphs can be constructed from the high-level system description used by the XMOS compiler toolchain — the XN file format — which describes the number of cores, their configuration and the network topology [XMO13a]. Additional physical attributes such as interconnect lengths, power supplies and peripherals are not present in such files, but can be added programmatically. Similarly, the entire system graph can be built programmatically, where there is a need to do so.

System graphs for arbitrary architectures

This thesis focuses upon the XS1-L processor and networks of such devices. However, the top-level multi-core model, leveraging arbitrary graphing and attributes, can be adapted to support other architectures and system designs. The underlying models of processors, and the behaviour of the network, can be substituted for new models and behaviours.



(a) XS1-U16A dual-core processor with integrated analog peripheral block, labelled as "2".

(b) Swallow 16-core board.

Figure 9.2: Top-level abstraction of components in a modelled multi-core network.

Network examples

Top level diagrams for sample XMOS configurations are shown in Figure 9.2. Figure 9.2a depicts the XS1-U16A processor. This features a dual-core XMOS processor, similar to the XS1-L2 used in Swallow, combined with an analogue peripheral block. The peripheral block uses the X-Link interconnect, so is part of the XMOS network. As such, its own switch connects to one of the processor switches.

Figure 9.2b shows a Swallow board. The physical positioning of the devices in the system is ignored in this simplistic visualisation, but the connectivity is visible, with each core having its own switch. On-chip links are replicated four-fold, but these are all drawn over the same coordinates in `NetworkX`, so not readily visible.

Merging models

Combining the single core model equations from Chapter 7 with the graph structure of the multi-core model, the total power and energy characteristics of the processors, their switches, and interconnects can be represented.

$$E_{\text{sys}} = \sum_{c \in \text{cores}} E_c + \sum_{s \in \text{switches}} E_s + \sum_{l \in \text{links}} E_l + \sum_{e \in \text{ext}} E_e \tag{9.3a}$$

$$E_s = \sum_{\text{t} \in \text{toks}} E_{\text{tok}} \tag{9.3b}$$

$$E_l = \sum_{\text{s} \in \text{syms}} E_{\text{sym}_l} \tag{9.3c}$$

At its top level, the energy for the system, Eq. (9.3a), is simply the sum of the energies consumed by its constituent parts, particularly the cores, switch and interconnects. Additional components, such as power supplies or peripherals, can also potentially be incorporated as well, through $\sum E_e$. This is a similar strategy to that previously used in ISA level models such as the Tiwari model reviewed in § 3.2.2.

Each core uses the extended regression tree model from Chapter 7 in Eq. (7.2) and considering Eq. (8.2) from § 8.1. The switch energy, Eq. (9.3b), is modelled with a fixed cost attributed per token passing through each switch. The static power of the switch is already accounted for in the P_{base} term of the core model. Each link uses Eq. (9.2) to determine the energy used in Eq. (9.3c), considering that each link may have different lengths and therefore different energy costs. In this work, other external effects are not considered.

Instruction	Condition	Purpose
`getr rD,2`	`rD = 0xNNNNCC02`	Channel end is resource type 2. `NNNN` is local node ID, `CC` is local channel end ID.
`setd res[rD],rA`	`rD = 0xNNNNCC02` `rA = 0xNNNNCC{02,c2}`	`rD` must be a local channel end resource, rA must point to another channel end (local or remote), and be a regular channel end (`02`) or a switch endpoint (`c2`).

`out res[rD],rA`
`outt res[rD],rA`
`outct res[rD],{rA,Imm}`
	`rD = 0xNNNNCC02`	Data is sent to the channel end previously set by `setd`. *This triggers network activity.*

`in rD,res[rS]`
`int rD,res[rS]`
`chkct res[rS],{rA,Imm}`
`inct rD,res[rS]`
	`rS = 0xNNNNCC02`	Data is received by the channel end. This consumes network data, but is passive; activity is governed by `out` instructions.

Table 9.3: Resource instructions observed by the core model to trigger system-level network modelling.

Network event modelling

During the modelling process, `XMTraceM` looks for network activity within the instruction trace emitted by `axe`. Table 9.3 lists the instructions that the core model looks for in order to track communication and register activity in switches and network links. The following steps are then taken in order to accumulate the communication cost.

1. Determine source and destination from `rS` and the last `setd` instruction issued against `rS`.

2. Find a path between the source and destination nodes, formed of intermediate nodes (switches) and connecting edges (links).

3. Increment the active energy of each switch and link based on the data size and attached attributes V and C.

The communication costs are annotated against each component, but currently presented as communication cost against the sending core along with each switch along the path. The presentation of this data is described in more detail in § 9.4.

9.3.2. Network model parameters

Using the Swallow link cost data from § 8.3, capacitances for the vertical, horizontal and layer links are estimated. This gives a parameter that can be used in Eq. (9.2).

The average lengths of horizontal and vertical links on Swallow, H and V, are measured using KiCad [KiC15] and the original PCB layout files. The recorded communication powers, $P_{\{v,h,l\}}$ and the ratio $\frac{V}{H}$ are used to estimate the switch power, P_s, in Eq. (9.4). The external track and internal bond capacitances can then be determined in Eq. (9.5), taking into account the data rate of each type of link, F_{elink} for external links, and F_{ilink} for internal links, represented as switching frequency.

$$
\begin{aligned}
V &= 39.1\,\text{mm}, \quad H = 44.2\,\text{mm} \\
\frac{V}{H} &= \frac{P_v - 2P_s}{P_h - 2P_s} \\
P_v &= 13.3 \times 10^{-3}\,\text{W}, \quad P_h = 12.6 \times 10^{-3}\,\text{W}, \\
\frac{V}{H} &= 1.13 \quad \therefore \quad P_s = 3.59 \times 10^{-3}\,\text{W}
\end{aligned} \tag{9.4}
$$

$$
C_{\text{track}} = \frac{P_h - 2P_s}{HV^2F_{\text{elink}}} = 643 \times 10^{-12}\,\text{F m}^{-1}
$$

$$
C_{\text{bond}} = \frac{P_l - 2P_s}{V^2F_{\text{ilink}}} = 1.63 \times 10^{-12}\,\text{F} \tag{9.5}
$$

C_{track} can be used with different lengths of interconnect. However, the length of the bonds within the package are not known, thus it is presented as an absolute capacitance.

Off-board interconnects

An additional test is used to capture a typical capacitance for longer, off-board X-link interconnections. A Swallow board can have its network reconfigured such that data is routed from the top row, out of the top external connections and looped around into the bottom external connections of the same board, using 30 cm ribbon cables. The same method from § 8.3 can be used to obtain power data when this long link is used, thus allowing a new capacitance estimate, C_{offlink} to be used, shown in Eq. (9.6).

$$
\begin{aligned}
L &= 308\,\text{mm} \\
F_{\text{offlink}} &= 19.1\,\text{MHz} \\
C_{\text{offlink}} &= \frac{P_L - 2P_s}{V^2F_{\text{offlink}}} = 2.63 \times 10^{-9}\,\text{F m}^{-1}
\end{aligned} \tag{9.6}
$$

This estimate represents the track between the chips and the external connector, as well as the ribbon connection. This data is not used in the single board experiments performed in this thesis, but can be used in future work.

9.4. Displaying multi-core energy consumption data

As the complexity of a modelled system increases, a single energy consumption figure becomes less useful. Software developers have a better ability to make decisions if they know where energy is spent, rather than just the total energy.

The multi-core extensions to `XMTraceM` provide energy consumption in two ways: a text or csv report, and through plots of system graphs. These two formats form the *inspection* phase of the modelling workflow from Figure 9.1.

9.4.1. Energy consumption reporting

The report output of `XMTraceM` can be given in pre-formatted text or a Comma Separated Value (CSV) file. If function filtering is enabled, a report can optionally be emitted after each completed filter. For example, if a filter is applied against a function that is called several times during the course of a program, several reports can be emitted.

The report, an abridged sample of which is shown in Listing 9.2, provides various pieces of information. The recorded time is shown, both in terms of total simulation time and the amount of time that energy consumption data was recorded. If no function filtering is performed, these will be equal. Then, for each core in the system, the energy consumption is given, separated into static, dynamic and communication energy, followed by a total. The instruction count and number of `FNOPs` is also given. Idle cores may show no instructions executed, but still contribute to energy consumption. Finally, the total energy consumption is summarised and the total power also given.

```
Time (Wall | Recorded): 1.00e+00 S | 336.23e-06 S
Core 0 (0x0000):
  Energy      (static  | dynamic      | comms          | total):
           8.58e-06 J | 6.34e-06   J | 145.07e-09   J | 15.06e-06    J
  Total instructions: 9377, FNOPS: 20

...

Core 14 (0x0304):
  Energy      (static  | dynamic      | comms          | total):
           8.58e-06 J | 5.50e-06   J | 0            J | 14.08e-06    J
  Total instructions: 0, FNOPS: 0

Core 15 (0x0306):
  Energy      (static  | dynamic      | comms          | total):
           8.58e-06 J | 6.25e-06   J | 0            J | 14.83e-06    J
  Total instructions: 8348, FNOPS: 16

Total Energy (static   | dynamic      | comms          | total):
         137.24e-06 J | 89.62e-06  J | 145.07e-09   J | 227.00e-06   J
Total Power  (static   | dynamic      | comms          | total):
         408.17e-03 W | 266.54e-03 W | 431.48e-06   W | 675.14e-03   W
```

Listing 9.2: `XMTraceM` report in text format.

9.4.2. Graph visualisation

A graph model of the system is useful not just for modelling, but also for visualisation. In § 9.3.1, an abstract representation of the network layout is presented, generated within the same framework as the model. This on its own can be useful for the programming in visualising how work could sensibly be assigned to the available cores on the network. However, with the addition of energy modelling, the visualisations can serve an additional purpose.

A possible visualisation of energy consumption is presented in Figure 9.3, where a colour is applied to each node and edge depending upon its energy consumption determined by the model. The colour scale can be arbitrarily defined, although a scale representing heat is most intuitive.

Eq. (9.7) determines a colour value, C, for each node or edge i, of type, t. The energy consumption, E, is used in this example, although other desirable metrics could also be used, such as power or time active, provided these are recorded in the network as attributes by the model.

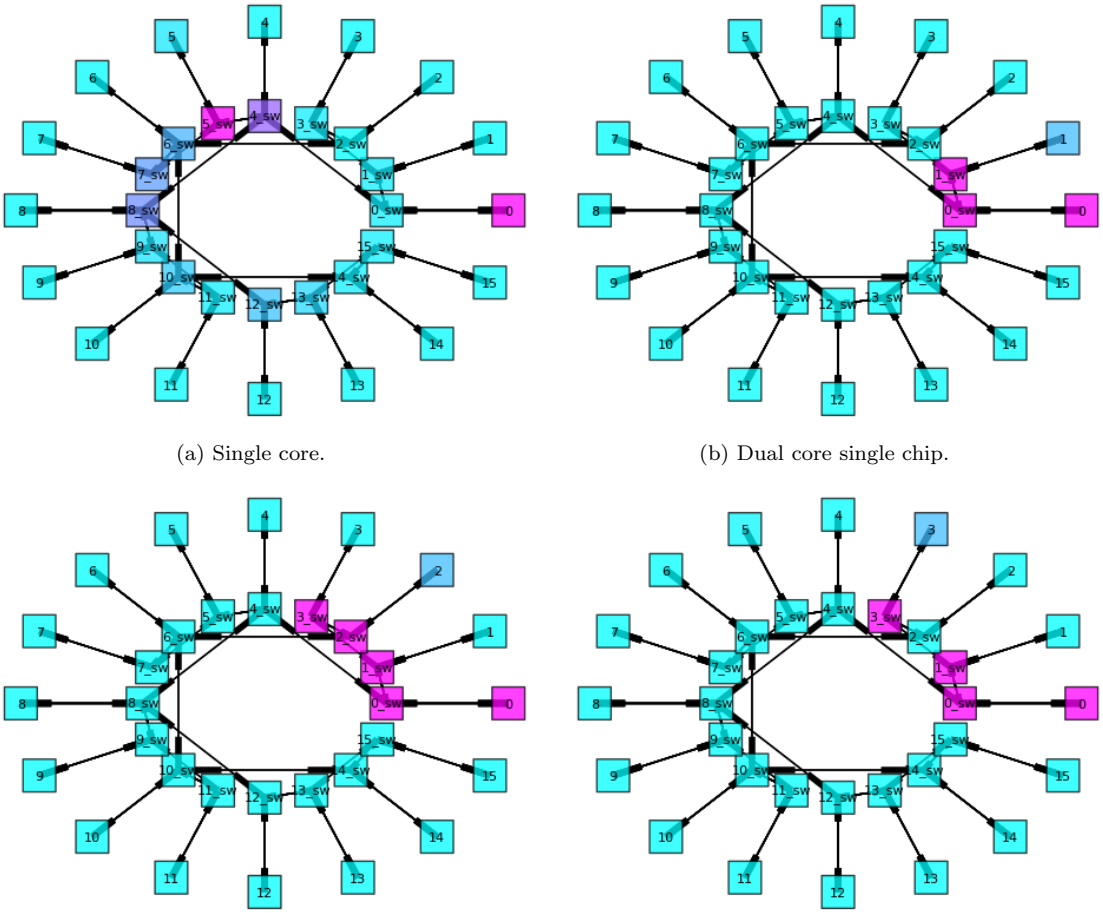
$$\forall\, t \in \{\text{core}, \text{switch}, \ldots, \text{link}\} \colon \forall\, i \in t \colon C_{t,i} = \frac{E_{t,i} - \min(E_t)}{\max(E_t) - \min(E_t)} \tag{9.7}$$

Each type is separately normalised onto the colour scale in order to avoid scaling from obfuscating lower energy components. For example, the interconnects consume significantly less energy than processor cores. Performing this segmented colour scaling may in some circumstances be undesirable, for example where the true "hot-spots" in the system are sought in absolute terms. Further, when comparing multiple versions of a piece of software in order to identify bugs in a particular version, or the one with the best energy usage profile, it may be preferable to bound the scale based on the system requirements, giving easier comparison between the modelling runs. In any case, these preferences could easily be configured and do not introduce any novel visualisation problems.

Example: Simple pipeline

The example given in Figure 9.3 shows four test cases for a three-stage pipelined program run on the Swallow platform. The program is very simple, intended for illustrative purposes only. In Figure 9.3a all three pipeline stages are executed on a single core and so only one core presents a significant amount of energy consumption. The remaining sub-figures split the workload between two cores. Figure 9.3b shows the split on a single package, where two cores consume more energy than the rest, with the core running two threads consuming the most. The switch energy is more obvious in this case, as data is carried between the pipeline stages over the network.

Figures 9.3c and 9.3d allocate the work between cores in different packages, increasing the number of switches that must be traversed. The core energy distribution is effectively the same



(a) Single core.

(b) Dual core single chip.

(c) Dual core two chip.

(d) Dual core two chip, better allocation.

Figure 9.3: Network graphs coloured by core, and switch energy consumption. Each "ring" is scaled independently, where the inner ring is switches and the outer ring is cores.

for all of the dual core examples. However, fewer switches are traversed in Figure 9.3d, thus less energy is consumed by network communication. The performance impact of this may be of little importance, for example to due latency hiding within the algorithms of the program. However, clearly, upon seeing energy consumption information presented in this way, a developer is able to choose a more energy efficient multi-core program layout.

9.5. Demonstration and evaluation

This section uses a simple demonstration program to show how the network level model works and provide an initial evaluation of its performance. The scope will be core-local, dual-core (same chip) and dual-core (two chips), to evaluate the models ability to handle different communication costs due to changing network speeds and wire lengths.

9.5.1. Test description

In this test, a sequence of randomly generated data is sent from one thread to another. Two parameters are explored during this test:

- Message length, where longer messages have a lower overhead from synchronisation and header tokens. Messages of $1, 2, 4, 8, 16, 32, 64, 128, 256$ and 512 words are used.

- Thread location, where the communicating threads are placed a different number of hops apart. They will be placed on the same core, the same package, or a neighbouring package. Thus, the number of network hops is zero, one or two, but the cost of a hop differs between tests.

To allow comparison between the hardware and simulation, each test is run for $10,000$ iterations in hardware. The execution time and power are then recorded, so that the energy consumption of the test or a single iteration can be determined. `XMTraceM` then estimates a single iteration, which can be compared to the hardware readings. The length of tests varies from 0.04 s to 5.22 s, depending on the message length and communication distance.

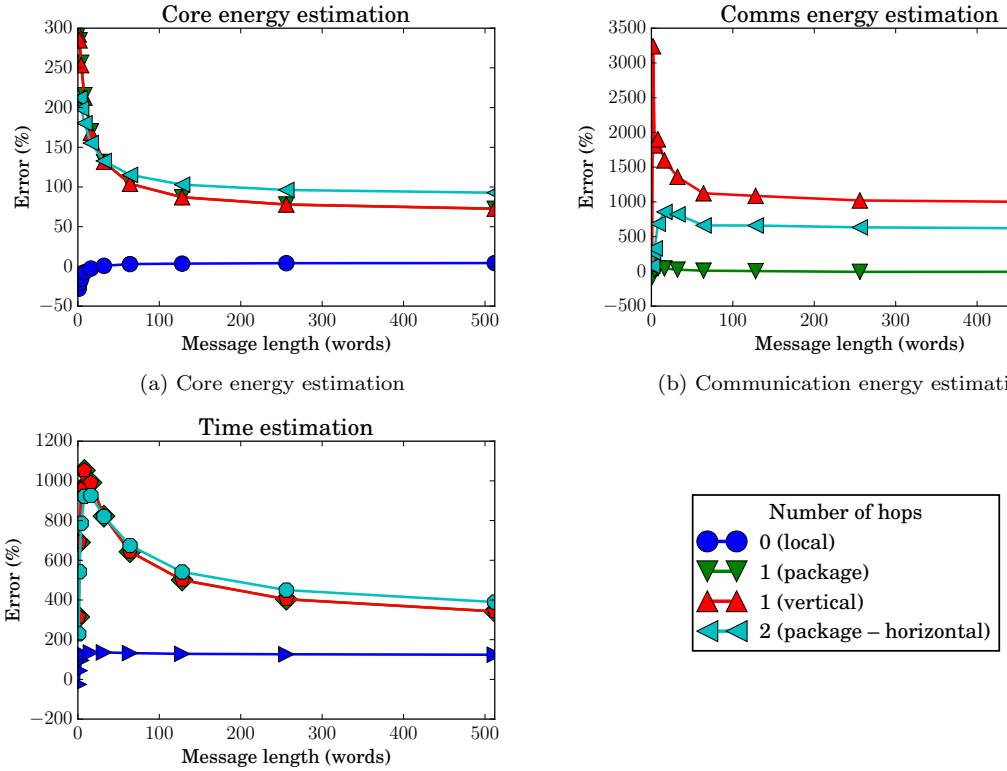
The power readings for Swallow are taken at the 5 V and 3.30 V supplies, as in § 5.2. Therefore, the power supply losses are used to reconcile `XMTraceM`'s core energy estimates against the measurements for Swallow, as was detailed in § 8.1.

9.5.2. Accuracy

The accuracy is presented in Figure 9.4. They are presented in terms of estimations of core energy (Figure 9.4a), communication energy (Figure 9.4b) and time consumption (Figure 9.4c). This data shows that the absolute accuracy of the multi-core modelling process is currently low, but provides some insight into where the accuracy is lost.

There exist both inaccuracies in the power and timing estimation, which leads to error in energy estimation for both core and communication. As the message size increases, the test harness and channel synchronisation become insignificant, thus the error margins stabilise. However, the overall mis-calculation of transit time for messages remains the largest contributor to error. The energy consumption of the communication activities themselves is relatively small, but the wait time for communication results in a significant mis-prediction in how long the program will execute for. As a result, the effect of modelling excess static and dynamic idle core energy consumption gives rise to the largest proportion of error. The error in on-chip (package) and off-chip (vertical or horizontal) communication work in opposing directions with respect to communication energy estimation. However, the impact on overall execution time is largely the same, regardless of the number of hops or their direction.

Figure 9.5 shows the hardware measured energy versus that of the model for both core and communication energy. This data shows that, whilst the accuracies presented in Figure 9.4 are problematic, the relative changes are representative of the system structure. For example, the most costly communication in these test cases uses one layer hop (within the package) and one horizontal hop, thus increasing communication power and time taken compared to all other cases. In turn, this increases 1 V and 3.30 V energy. This increase can be seen in the model data, but it does not fit well with the actual measurement. Nevertheless, to a developer, the penalty for longer communication runs is visible. A single vertical hop can also be observed as more costly than a hop within the package. Intuitively, this communication takes more time due to the slower link speed, and there is a larger wire capacitance as well, due to significantly longer wire length.

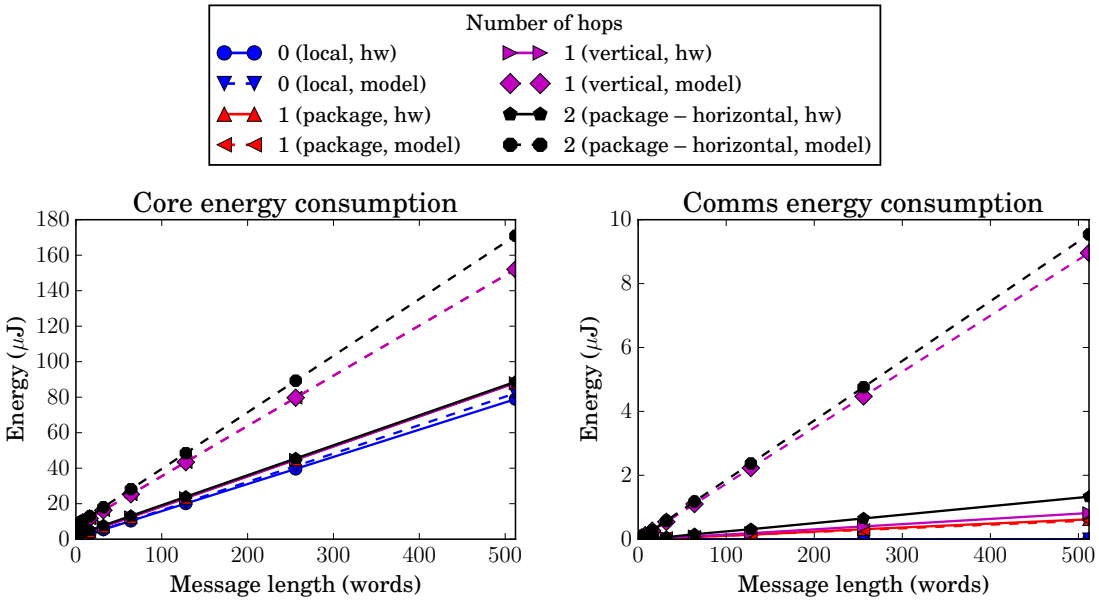


(a) Core energy estimation

(b) Communication energy estimation

(c) Execution time estimation

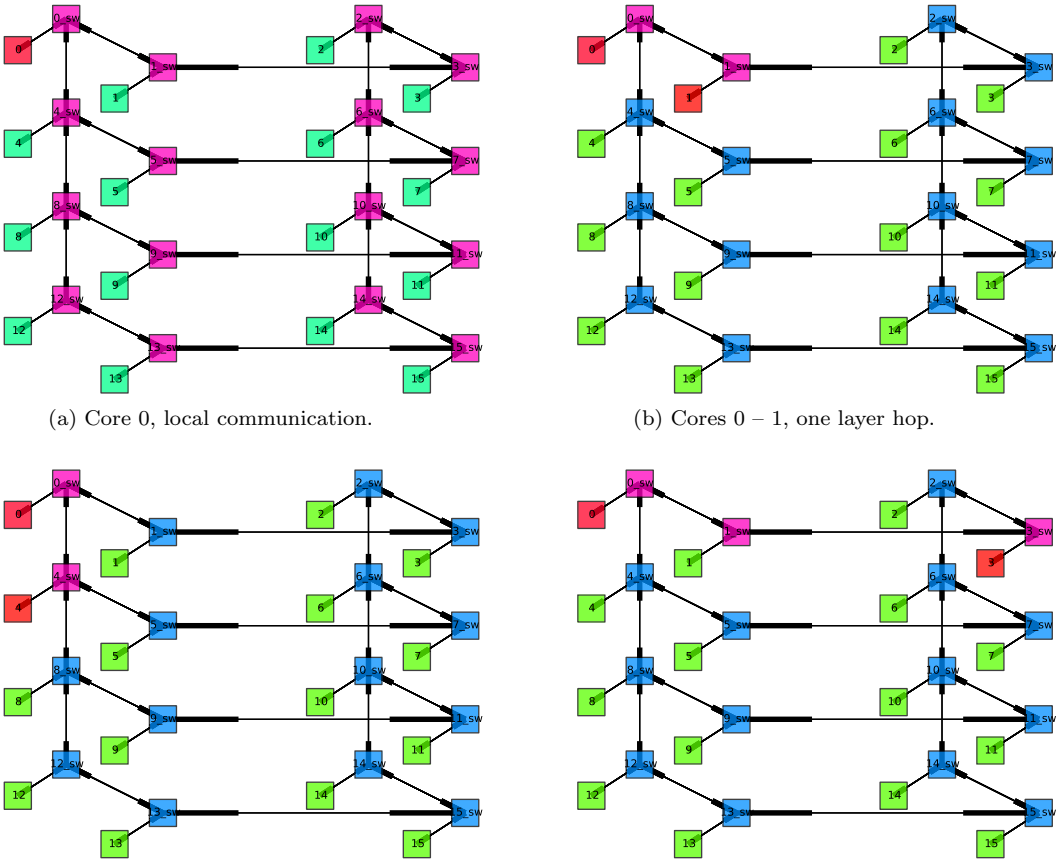
Figure 9.4: Multi-core modelling accuracy.



(b) Core energy

(c) Communication energy

Figure 9.5: Measured (hw) and estimated (model) energy consumption.



(a) Core 0, local communication.

(b) Cores 0 – 1, one layer hop.

(c) Cores 0 – 4, one vertical hop.

(d) Cores 0 – 3, one layer hop, one vertical hop.

Figure 9.6: Refined modelling visualisation for Swallow.

9.5.3. Using modelling for visualisation

The previous subsection evaluated the model in terms of the number of hops, and the type of hop over which communication took place in the test application. A developer has the choice of which cores to allocate threads onto, and so visualisation of both the network topology and the impact of that allocation with the assistance of the energy model, is potentially valuable.

Refining the visualisation examples given in § 9.4.2 in the context of the tests performed in this section, the impact of thread placement can be seen. In Figure 9.6, the graphs have been custom formatted to approximate the Swallow system layout.

In this layout, the active cores and switches along with the connectivity between them is more visible. For example, it clear that sharing data between cores 0 and 3, which uses a single vertical hop, is more expensive than sharing data between cores 0 and 4, which uses an in-package hop and then a horizontal hop. The switches and cores use separate colour scaling. High/low energy cores are red/green, whilst high/low energy switches are pink/blue.

This visualisation, combined with the textual reporting shown previously in Listing 9.2, can be used to compare thread allocation strategies in order to find optimal solutions. This strategy forms part of the recommendations that are made in § 11.6.

9.6. I/O as an adaptation of the network model

The proposed multi-core network model implements an approximation of the XS1 network. It does not rigidly follow the underlying routing and control strategies in the architecture, but can still provide useful information to the software developer.

This flexible implementation can be exploited for other means. For example, additional components that are not compute nodes can be modelled in a similar fashion. Figure 9.2a (§ 9.3.1) demonstrates this possibility by including the analog peripheral block of a processor in the network model. In this example, the peripheral block forms part of the XS1 X-link network, thus it is connected via a network switch. With a suitable model for the peripheral component, communication to and from that component, along with energy costs, can be implemented.

Although this example depicts a peripheral block forming part of the XS1 network, this is not necessary. A peripheral may be connected directly to a processor core, rather than a network switch, or share connection to a core on a bus, as is typical in traditional memory hierarchies with memory-mapped peripheral devices.

Providing this functionality in the model is relatively straightforward. The more significant effort is in providing a simulated version of an I/O device that `axe` can use. This precludes giving a concrete demonstration in this thesis, but could be the subject of future work.

9.7. Summary

In this chapter, a multi-core energy model was proposed and tested. It uses the energy profiling data for Swallow from Chapter 8 and extends the `XMTraceM` framework to work for multi-core system. Although the accuracy of the network model is not comparable to the single-core multi-threaded model, energy consumption visualisations have been presented that provide insight for a software developer.

The implementation of the multi-core model involved significant changes to the `axe` simulator. This included new trace data formatting in JSON, that is rich in information that can be exploited by this and potentially other models. Further, an approximate network timing model was introduced to `axe`. Without implementation of the switch buffering and flow control, however, accuracy of timing is limited.

This chapter creates a number of opportunities for future work. Improvements to the network simulation could be implemented and evaluated. I/O could also be abstracted through the same framework that has been contributed here. Looking beyond simulation, static information about communication costs could also be obtained from this framework, by interrogating the properties of the network. This would de-couple the efficacy of the model and framework from the accuracy of the underlying multi-core simulation, creating new ways to use the framework.

10. Beyond the XS1 architecture

MTMC can be implemented in many ways, as was outlined in § 2.3. The main body of this research has used the XMOS XS1-L as the subject of profiling and modelling, and as such has dealt with the multi-threading and inter-chip communication mechanisms present in that particular architecture.

This chapter explores how the relevant aspects of this research can be transferred to other architectures, by analysing the architectural differences, identifying what changes would need to be made, and suggesting the most effective ways of making these changes. In some cases, the differences may be significant enough that transferability of techniques is limited. This chapter also gives consideration towards the ease with which software energy consumption can be modelled against architectures, principally within the techniques explored by this research.

The structure of this chapter is comprised of an introduction to several other architecture, of which some are embedded and others are not. Each is discussed in turn, relating characteristics that lend themselves to the style of energy models proposed in this thesis, as well as identifying characteristics that work against these approaches. The architectures that will be discussed are Adapteva Epiphany (§ 10.1), the Intel Xeon Phi (§ 10.2), various ARM implementations (§ 10.3) and the EZChip Tile processor (§ 10.4). A brief summary of the main discussion points is then given in § 10.5.

10.1. Epiphany

The Adapteva Epiphany processor [Ada13] is a multi-core floating-point architecture intended to be high performance, low power and scalable. Epiphany is described by the vendor as a "clean slate architecture" [Ada15]. The processor is designed to fit into a heterogeneous environment, working alongside other processing devices, such as what might be termed more conventional ARM or x86 processors as well as FPGAs.

Epiphany implements a Network on Chip (NoC) comprising three networks — one for read operations on chip, one for write operations on chip, and one for any transaction involving off-chip components. The memory model maps the local memory of all cores into the global address space, so that it is possible to read and write the memory of other cores. Additional memory may be mapped into the address space. This can be implemented as a DRAM controller attached to the external network.

There are no cache memories in current implementations of the Epiphany architecture. However, the local, remote and off-chip memory accesses can be considered analogous to a multi-level memory hierarchy. These three modes of memory access need to be profiled and modelled in order for data movement in the Epiphany architecture to be represented in a similar way to that of the channel modelling demonstrated for the XS1. The Epiphany's pipeline is somewhat different to the XS1, not just in that its includes FPU operations, but also in that it is variable length [Ada13, pp. 62–67].

10.1.1. Memory map and network

The Epiphany G3 architecture provisions the 12 most significant bits for node addressing, allowing up to 4096 node addresses, `0x000`*`nnnnn`*–`0xfff`*`nnnnn`*. Each node has 32KiB of memory, accessible either combined with the node address or aliased into `0x00000000`–`0x00007fff`. Processor registers are also memory mapped into `0x`*`nnn`*`f`*`nnnn`* for all nodes. This topology creates a Partitioned Global Address Space (PGAS) where memory is distributed between all nodes, but accessible globally.

Local memory access

When the most significant 12 bits of the memory address match the local node ID, or they are zero, then a local memory operation will be performed. As such, operations on these addresses can be modelled based on profiled local memory access costs and latencies. A core model, implemented in a similar way to Chapter 7 or prior methods from Chapter 3, can capture this type of access.

Remote memory access

If the node address bits are non-zero and do not equal the local address, but reside within the same chip, then a remote on-chip access will be performed. This can be modelled based on memory access costs in addition to network transaction costs. The 2D, dimension-order routed network allows for simple hop calculations based on the difference between the source and destination node addresses, as shown in Eq. (10.1), where N is a node ID, representing the upper 12 bits of the memory map and D is the distance, or number of hops between the two nodes.

$$\begin{aligned} D = &|N_{\text{src}}[11:6] - N_{\text{dst}}[11:6]| \\ &+ |N_{\text{src}}[5:0] - N_{\text{dst}}[5:0]| \end{aligned} \tag{10.1}$$

Write operations are 8 times faster than reads, so the type of network transaction will also dictate latency and therefore static power consumption. If there is congestion in the network, then the increased latency will have an effect on total static power consumption. This is largely analogous to the network model implemented for the XS1-L, with a different set of performance constraints that must be represented during simulation or abstracted by the model.

Off-chip memory access

Accesses off-chip can be handled in a similar way to those of remote memory accesses, but there may be additional components and topological considerations to take into account.

If the off-chip activity is to another Epiphany chip, then the external hop cost must be considered, in addition to all internal hops in both chips. The external hops can be considered by extracting the most significant bits of the row and column addresses, dependant on the number of cores per chip. In the 16-core variant, address bits 27–26 are used for the local row and bits 21–20 for the local column, if the upper bits match. Bits 31–28 and 25–22 will be for the chip row and column, respectively. Eq. (10.1) can then be extended to give two hop costs, dependant on whether they are external or internal, as in Eq. (10.2).

$$\begin{aligned} D_{\text{int}} = &|N_{\text{src}}[7:6] - N_{\text{dst}}[7:6]| \\ &+ |N_{\text{src}}[1:0] - N_{\text{dst}}[1:0]| \\ D_{\text{ext}} = &|N_{\text{src}}[11:8] - N_{\text{dst}}[11:8]| \\ &+ |N_{\text{src}}[5:2] - N_{\text{dst}}[5:2]| \end{aligned} \tag{10.2}$$

An off-chip Epiphany access could be modelled using the same set of principles as shown in Chapter 9. However off-chip accesses may not necessarily be to other Epiphany cores. Therefore, if an address range is allocated to a device such as a DRAM, the device's read/write behaviour should be profiled and utilised in the model whenever necessary. This is a similar modification to what would be needed to model arbitrary I/O devices in the XS1-L. However, DRAM controllers are particular complex devices, so this may be a barrier to rapid model development unless existing DRAM models can be integrated.

10.1.2. Summary

This section has described Epiphany, a highly parallel processor. It differs from the XS1-L in three major ways: its pipeline, the network and its memory model.

Epiphany is multi-core, however it is not multi-threaded like the XS1. Each core has a super-scalar pipeline, the effects of which cannot be accounted for without a model that determines how

the pipeline will be utilised. The core model targeting the XS1 has no mechanism to account for this.

There are three distinct NoCs that use different flow control techniques to the XS1-L. The routing method is relatively simple dimension-order routing; simpler than the Swallow system. Therefore similar network modelling techniques could be applied to the Epiphany.

Finally, memory mapping and shared memories are used at the architectural level, in stark contrast to the message passing using in the XS1-L. The memory map is very clearly defined, meaning that in simulation, memory activities that invoke network communication will be easy to identify. The lack of explicit channel resources may make higher level analysis, such as static analysis, harder to perform.

Giving consideration to these properties, there is a good case for transferring the style of model presented in this thesis onto the Epiphany. This would require additional work, particularly in modelling the Epiphany core and potentially an external memory.

10.2. Xeon Phi

Intel's Xeon Phi processor is an accelerator product intended to co-exist with a host processor and provide high performance, highly-parallel compute capabilities. It bears some similarities to a GPU with GP-GPU functionality, although its heritage is in the x86 architecture, rather than bespoke 3D graphics architectures. It is a significant departure from traditional x86 processors, with only a small proportion of the processor dedicated to x86 logic. Xeon Phi introduces a large number of new vector processing units.

Comparison with the XS1-L or any other embedded architecture is not straightforward, because the Phi is a different class of processor. It is built for HPC applications, not embedded real-time. It is perhaps more similar to the Adapteva Epiphany architecture than the XS1-L, although still vastly different. Nevertheless, there are a number of characteristics that suggest some tractability with respect to using the type of model described in this thesis. There is also an energy model for the Phi [SB13], which was identified in Chapter 3 that has some similar properties to the XS1-L model. It will be discussed in more detail in this section.

10.2.1. Architecture details and discussion

The Xeon Phi implements a NoC of processor cores, caches, tag directories and memory controllers, with the aim of providing high bandwidth memory that minimises interruption of parallel processing tasks. The construction and use of these are discussed in this subsection and related to the proposed energy model, giving consideration to the effort required to account for them using such an approach.

Processor cores

Each core within the Phi has hardware support for four threads, with the hardware multi-threaded implemented as a front-end to an in-order, dual-pipeline super-scalar back-end. At the time of writing the largest available Xeon Phi contains 61 cores, giving up to 244 threads. Despite the in-order approach to the pipeline implementation, the Phi's super-scalar cores are still significantly more complex than the XS1's round-robin 4-stage pipeline.

Memory hierarchy

There is a 512KiB L2 cache per core. The cached addresses of all L2 caches within the Phi are maintained by a set of tag directories. When queried by an L2 cache, these directories can either find the required data in a neighbouring cache, or perform a memory access by forwarding a request to one of several Graphics Double Data Rate (GDDR)-5 memory controllers connected to the Phi's onboard memory.

Attempting to model the cache behaviour for the purposes of providing energy data is troublesome, as the behaviour of the caches must be modelled, as well as the forwarding requests to the tag directories, memory controllers and memory.

Network topology

The cores, caches, tag directories and memory controllers are interconnected by a bi-directional ring network, that is separated into three components — data, addressing and acknowledgement. The data ring is 64 bytes wide. There are twice as many address and acknowledgement rings (four in total) as there are data rings (two in total) in order to maximise the bandwidth usage of the area-expensive data rings.

The network further complicates the process of modelling memory accesses, as there is potential interaction between caches, memory controllers and tag directories from multiple cores simultaneously, depending on the access pattern of the application.

10.2.2. An existing energy model for the Xeon Phi

The Xeon Phi, whilst a relatively new architecture, has received attention from research into software energy costs. Shao and Brooks [SB13] performed an energy characterisation of the Phi with the aim of producing an instruction level energy model for it.

The method of exercising the processor is similar to that of prior work and indeed the work in this thesis. A set of micro-benchmarks are used to exercise specific features of the processor, measuring the power dissipation in order to characterise that feature's contribution towards energy consumption. In the case of the Phi, the memory hierarchy and cache layers make a significant contribution towards total energy consumption. For example, a memory operation without pre-fetch, for a single-core, single-threaded case, requires in the region of 230 nJ of energy, whereas in-register costs are in the order of 1 nJ.

Hardware performance counters are used to guide the characterisation process, indicating the utilisation of each level in the cache hierarchy, number of pre-fetches and so on. Vector and scalar operations are profiled, as well as the `vprefetch0` and `vprefetch1` commands, which explicitly pre-fetch data into the L1 and L2 caches respectively.

The Phi's multi-threaded in-order cores share some similarities to the XS1-L with respect to power dissipation at different threading levels, which are presented in § 7.2.2. The instruction cost for single-threaded operation is sub-optimal versus two or four threaded operation; single-threaded scalar and vectors operations uses 67 % more energy. Although the Phi and XS1-L architectures are different in many ways, this demonstrates that under-utilisation of cores designed for multi-threaded computation, is undesirable if maximum energy efficiency is sought. As with the XS1-L core model, the unique energy behaviour at different threading level has to be considered in the energy model of the Phi.

The resultant model defines Energy Per Instruction (EPI), based on the type of operation, threading level in the core, and the location of operands in the memory hierarchy. The model is integrated with the Intel VTune performance profiling tool and used to predict program energy with the performance counter predictions available through this tool. The accuracy of this approach is claimed to be within 5 % of actual energy consumption. The usefulness of the model is demonstrated with a ported, performance-tuned version of the Linpack benchmark, in which 10 % of energy consumption is shown to be due to redundant pre-fetch operations. These would not necessarily harm performance, but clearly consume energy unnecessarily.

10.2.3. Summary

The Xeon Phi operates in a considerably different application space to the XS1-L. This section has highlighted the significant differences between the two processors, particularly the core complexity and ring based memory hierarchy of the Phi. However, much like the XS1-L, the Phi implements multi-threading and its energy consumption scales in a similar way as the number of threads per core increases.

The similarities between some of the behaviours in the Phi and XS1-L, and the common observations in this thesis and the work of Shao and Brooks lend credence to the overall techniques of profiling multi-threaded processors in this way. In particular, the thread utilisation of a core is important for energy modelling, which was not required in previous instruction level energy models for single-threaded cores.

Given that this thesis focuses on software for embedded systems, the modelling techniques presented here may not map well onto the Phi. However, at least some commonality has been shown, suggesting that energy modelling of software in any application space can be tackled in similar ways.

10.3. Multi-core ARM implementations

ARM architectures are used within embedded systems in their billions. However, the implementation of ARM based processors is far more fragmented than the other architectures reviewed in this chapter. Therefore, modelling one ARM processor does not necessarily cover the many implementations that exist.

There are many versions of the ARM instruction set, the current versions being the 32-bit ARMv7 and the 64-bit ARMv8. These architectures are integrated into devices designed by different companies and produced by different manufacturers. There are different System on Chips (SoCs), different physical layouts and different process technologies.

This chapter focuses on implementations that deliver multi-core capabilities in order to reflect on how their energy consumption can be modelled. It gives consideration to the heterogeneity of the ARM market, but does not consider the entire spectrum as this would be significantly beyond the scope of this thesis. Two produce ranges are considered, the Cortex-A series and the Cortex-M. An important multi-core energy saving technology, named big.LITTLE, is also examined.

10.3.1. Multi-core Cortex-A processors

There are a large number of multi-core Cortex-A based processors in simulations, the most notable application area being smartphones. These have significantly higher performance than deeply embedded systems such as those served by the XS1-L.

The Cortex-A processors are super-scalar, out-of-order processors and can be used in multi-core configurations. A comparison between the Cortex-A9 and other multi-core processors is made by Blake et al. [BDM09]. Cache coherency and the use of ARM's various interconnect technologies, such as Advanced High-performance Bus (AHB), do not match well with the style of model presented in this thesis.

10.3.2. The big.LITTLE philosophy

ARM's big.LITTLE technology broadens the range of power and performance capabilities of a single device by incorporating two processor implementations. One processor type (big), delivers the highest performance, but its DVFS curve limits the power savings at the lower ends of performance. The second processor type (LITTLE), has a lower operating power and can scale performance down further than the big core. There is a small overlap between the performance profiles of the two devices [Gre11, p. 5].

The first big.LITTLE implementation used a Cortex-A15 combined with a Cortex-A7 [Gre11]. Both of these cores implement the ARMv7 ISA, so binaries are compatible between them. A workload can be migrated between the cores during runtime if a change in operating point is deemed beneficial, for example to save energy whilst performing a low performance task. The transition requires a significant amount of state to be transferred between the cores; no more than 20,000 cycles according to the vendor. Cache coherency must also be considered in the system's L2 cache.

Examples of uses of big.LITTLE include the use of eight cores in various Samsung Exynos processors, where four A7 and four A15 cores are present [CCK12]. A number of these products allow all eight cores, or combinations of the cores, to be used, whilst earlier versions only permit either A7 or A15s to be used at any one time, due to cache coherency. ARM's 64-bit A57 and A53 processors can also be combined into big.LITTLE SoCs [ARM12].

Modelling these types of processors, particularly with their cache hierarchies, is somewhat different to the XS1-L and the models proposed in this thesis. Each of the cores and their DVFS behaviour must be modelled, as well as the transition process between big and LITTLE cores whenever it is invoked. Chapter 3 reviews various energy models, including those that target

ARM processors of a single core nature. A big.LITTLE processor has been energy modelled in the context of web page rendering [ZR13]. Page characteristics are identified in order to estimate the rendering effort and choose an appropriate operating point for the processor without exceeding a cut-off latency for rendering. The scope of this is outside of the types of embedded systems discussed in this thesis, but demonstrates that energy models for this type of ARM processor are possible and can be exploited. In this case, web page rendering could be done with 83 % energy savings.

10.3.3. Multi-core Cortex-M processors

In embedded computing, the Cortex-M series is widely used. Many embedded controllers include a Cortex-M and embedded systems may be comprised of several of these. However, they tend to be programmed independently, resulting in heterogeneous systems of processors with exclusive programming models. The M-series use the ARM Thumb instruction set, a more compact ISA than the traditional 32-bit ARM form. They also feature no cache controller, so tend to execute directly out of flash or RAM and access the RAM regularly.

Multi-core M-series systems, are possible however. ARM has published a white paper on the subject [YJ13], that highlights a number of key design decisions. The main consideration is main memory access, as the architecture follows shared memory paradigms. Without caches on the cores, the memory hierarchy is seemingly simpler, however arbitration must be provided and the performance cost of RAM access considered. A system level cache can optionally be used.

This design approach is significantly different to the network approach examined in this thesis. As such, simulators and models that exploit existing ARM processors on Gem5, may be a better fit, with less effort to extend to support whatever multi-core M-series implementations arise.

One possible opportunity lies in a heterogeneous multi-core system comprising an ARM Cortex-M3 and an XMOS XS1-L series that XMOS has introduced [XMO14b]. The biggest challenge in this scenario would be to simulate both architectures simultaneously, whilst handling communication between them as a network level activity. Both ARM and XS1-L models have been used in work that builds upon this thesis [Gre+15] but only in single device scenarios.

10.3.4. Summary

The many ISAs and SoCs that can be called “ARM cores”, do not appear to represent systems onto which the models proposed herein could easily be transferred. Significant work into ARM energy modelling exists, however, so the contribution of transferring the work in this thesis would not necessarily be significant.

10.4. EZChip Tile processors

The EZChip (formerly Tilera) Tile processor is perhaps the most similar to the XS1 architecture of those discussed in this chapter. Several iterations of the processor exist, following the same multi-core design principles. The devices target low latency processing of data send and received over Ethernet [EZC09].

In a Tile processor each core is single-threaded, but implements a VLIW pipeline. This can be represented in a similar way to the *concurrency level* described by the XS1 model as shown in § 7.2.2. The concurrent behaviour of the pipeline is more predictable than the XS1-L because of the VLIW implementation. The single-threaded in-order execution gives certainty in the sequence of instructions progressing through the pipeline. Other models for VLIW processors may also have characteristics that could be transferred to the Tile processor.

There are five networks in the Tile processor [EZC13, pp. 29–32], divided into two classes. The first are user accessible and the latter only system accessible. The five networks are user messages, I/O messages, memory transfers, cache coherency messages, inter-tile messages. This is significantly more granular than the XS1-L or Epiphany (§ 10.1 network implementations. Like the Epiphany, shared memory is used as the programming model. Caching is also provided, hence the presence of a network for cache coherency signalling. The Tile’s processors form a 2D grid.

The separation of networks helps simplify any modelling effort. However, following the methods presented in this thesis, each network and its associated activities must be profiled and model parameters determined for them. The memory hierarchy also poses modelling complexity. The individual cores may therefore prove significantly easier to model than the rest of the Tile, where both network and memory structure introduce potential barriers to ISA level modelling.

10.5. Summary of model transferability

In this chapter, a selection of processor architectures were surveyed. The purpose was to highlight where the models proposed in this thesis may be transferable, and where this may prove impractical. A brief summary is given in Table 10.1, with further thoughts in this closing section.

This chapter has shown that processors in significantly different performance brackets, such as the Xeon Phi, can still share model properties with those that operate in the embedded space. Other many core devices, such as the Epiphany and Tile processors, implement grid-like networks that bear some similarities to the network capabilities of the XS1-L and the lattice implemented in Swallow. However, there are sufficient differences that a large number of changes would need to be made for a network level model of the nature presented in Chapter 9 to be applicable. For example, both Tile and Epiphany processors use multiple networks, each for a different purpose.

The most varied range of processors discussed were ARM based devices, which proliferate many markets at different performance points, and for which there are a great variation in implementations. Although there are recommendations from ARM on multi-core embedded Cortex-M series processor implementations, the higher performance Cortex-A devices tend to use multi-core more readily. These are already served by energy models such as those based on Gem5. ARM's big.LITTLE technology also presents further modelling challenges due to heterogeneity and cache coherency. However, these devices do come with the potential reward of better energy efficiency through more performance/power trade-off choices in a single device. It is interesting to observe that efforts to save energy in hardware designs may increase the difficulty or reduce the amount of detail available in modelling energy consumption from a software perspective.

Processor	Similarities	Differences
Adapteva Epiphany	2D grid NoC, similar structurally to Swallow.	Variable-length pipeline.
Intel Xeon Phi	In-order multi-threaded pipeline. Existing energy model bears similarities.	Large ring network with complex memory hierarchy.
ARM big.LITTLE	The "little" cores have simpler micro-architectures, closer to XS1-L than their "big" counterparts.	Cache hierarchy and cache coherency mechanisms, different DVFS behaviour and modelling requirements in big and little cores.
ARM M-series multi-core	Simpler cores that can be deeply embedded. A sipmle M-series model has been used in static analysis alongside the XS1-L model.	Multi-core implementation significantly different from XS1-L network, including caches and arbitration.
EZChip Tile	VLIW pipeline, with some transferable techniques from XS1-L thread modelling. 2D grid NoC, similar structurally to Swallow.	Multiple networks, including caches and cache coherency signalling.

Table 10.1: Summary of key similarities and differences between XS1-L and other architecture, considering applicability of energy modelling.

11. Conclusions

This chapter presents a conclusion and evaluation of the complete contributions of this thesis. It begins by re-stating the research question and thesis declarations posed in Chapter 1. The contributions to those thesis points are then highlighted in turn, summarising the work put forward in the previous chapters. An evaluation is then presented, addressing how this work can be used in future efforts to further the state of the art through possible alternative approaches, improvements to the presented methods, or follow-on research.

11.1. Review of thesis contributions

This work has made several contributions to the state of the art in energy modelling of software for MTMC systems. A profiling framework was constructed with precise instruction schedule guarantees, in order to collect multi-threaded instruction energy data. This data was then used to create a new energy model, and that model completed with the use of a regression tree technique. A large multi-core embedded system was then used to produce multi-core communication profile data, which was then used in a multi-core energy model that provides a network-level view of where energy is consumed in the system.

The formative statements of this thesis were defined in § 1.1. These are re-stated below, then reflected upon with respect to the contributions that were made towards them throughout this work.

Effective energy estimates for modern embedded software must consider multi-threaded, multi-core systems. The motivation for this was made in Part I, where parallelism and concurrency were explored in Chapter 2 and existing energy modelling techniques reviewed in Chapter 3. The present state of processor tecnhology and energy saving techniques was reviewed in Chapter 4, demonstrating that concurrency in the form of multi-threading and multi-core is then new means through which performance increases are made. Combined, these chapters make the case that energy modelling of software must be capable of considering these new characteristics of hardware in order to remain useful.

Energy modelling at the instruction set level provides good insight into the physical behaviour of a system whist preserving sufficient information about the software. Chapters 6 and 7 address this subject by demonstrating that an ISA can be used to form a core energy model that can estimate software energy consumption with good accuracy. This model can also be raised to higher levels of abstraction, such as static analysis, still underpinned by the ISA level energy consumption data.

Energy saving and energy modelling techniques are placed under greater constraints in the embedded space. Chapter 4 presents how DVFS, a hardware feature for energy saving, is constrained when real-time requirements are considered. Chapter 5 describes the XS1-L processor, which is designed to allow hardware interfaces to be written in software. This gives a firm example of where these constraints apply, increasing the value in exposing more information on energy consumption to the software developer, giving them more design exploration options.

Multi-threaded and multi-core devices introduce new characteristics that must be considered in energy models. Both Chapter 6 and Chapter 10 show properties of Multi-Threaded and Multi-Core (MTMC) processors that are not present in single threaded, single core counterparts. In the case of Chapter 6, these characteristics are accounted for in the models proposed in the rest of Part II.

Energy models that do not rely on hardware counters provide greater flexibility for multi-level analysis. Several prior energy models, discussed in Chapter 3, rely on hardware performance counters, in order to provide energy estimates for reporting or optimisation. Chapters 6 and 7 have shown that in the absence of these and through profiling to build an ISA level model, the estimation process can be de-coupled from the hardware and applied at various levels of abstraction.

Both absolute accuracy and relative indicators provide useful information to a developer. The multi-core Swallow system, described in § 5.2, then profiled and modelled in Chapters 8 and 9 respectively. It is shown that when the modelling techniques in this thesis deviate from the actual hardware energy, the relative measures they present still have value in identifying where energy consumed in the system, at the behest of the developer's software.

Movement of data costs energy, no matter the form that movement takes. The channel communication model of parallelism, implemented in the XS1-L and on Swallow, dispenses with shared memory. However, it is shown in this thesis that the movement of data is still costly in terms of energy, particularly with respect to time.

Energy models for different architectures can have elements in common. Transferability of energy models was discussed in Chapter 10. Several challenges were outlined that must be overcome if energy models are to be made portable between architectures. However, common properties were identified that will can at least contribute to the more rapid development of models for other architectures. The profiling techniques demonstrated in Chapters 6 and 8 can also be re-tooled for other systems.

11.2. Building a multi-core platform for energy modelling research

The majority of the modelling, profiling and evaluation contributed in this thesis would not have been possible without the significant effort that was put into providing usable hardware platforms. In the case of single core profiling, this was largely a software contribution on the part of the author and was described in Chapter 6.

For the multi-core Swallow system (§ 5.2 and Chapter 8), the starting point was a device with no means of fully exploiting its interconnection network. A significant contribution undertaken during this thesis was enabling research to be conducted on this platform, through an effort to bring-up the board and provide tools for programming and interfacing with it, as well as monitoring the energy consumption of its various power supplies.

Now that this work has been carried out, the scope of what can be researched on Swallow is yet to be fully explored. Swallow was used to the benefit of this thesis in producing network level energy models, and also for other energy efficiency themed research [HMM15].

11.3. ISA-level energy modelling for a multi-threaded embedded processor

This thesis has shown that multi-threaded energy models can be built for embedded processors that yield single-digit percentage errors (2.67 % on average), in a range of single- and multi-threaded software benchmarks. For applications that require concurrency within the scope of a single processor, this work provides a means for the developer to estimate the energy consumption of their code without taking hardware measurements. This work was shown with the XS1-L processor.

The production of the energy model was made possible by first constructing a flexible profiling framework, `XMProfile`. This framework allows tightly controlled instruction sequences to be issued through the pipeline of the XS1-L processor. Combined with power measurement equipment, this allowed a portion of the ISA to be automatically profiled for its energy consumption characteristics. Further instructions were profiled through more directed testing, still operating within the `XMProfile` framework.

The profiling effort revealed energy behaviours specific to the XS1-L's architecture. In particular, its energy consumption with respect to the number of active threads scales non-linearly. In addition, the underlying *base cost* of the system must be defined without reference to instructions, due to the processor's hardware scheduling and event-driven properties.

This data was then used to construct an energy model, capturing program energy consumption with respect to the instructions executed and the amount of parallelism present. Various methods were used to represent the energy of un-profiled instructions. A regression-tree based method was shown to be the most accurate. It utilises a set of characteristic features of each instruction, combined with the recorded energy of instructions with similar features, in order to estimate the energy where it is not known.

The estimation process presented in this work focuses on Instruction Set Simulation (ISS) traces, against which the proposed energy models are evaluated. These shorten the loop from software development to evaluating energy consumption, particularly where the developer does not have the means to instrument their system for energy measurements. The simulations themselves are not as fast as hardware by approximately two orders of magnitude. Optimisations to the trace process have been shown that mean entire program runs do not necessarily have to be simulated in order to extract useful energy consumption information. One must also consider that in the face of not being able to acquire hardware energy measurements, simulation is a desirable alternative, and ISS is orders of magnitude faster than the lower level simulations performed in hardware design space exploration.

External work has also been shown [Liq+15; Gre+15] that has successfully exploited these core models in static analysis. Thus, the work of this thesis is not constrained solely to simulation based modelling. The models provide sufficient characterisation of the underlying hardware in a way that is not dependent upon characteristics that must be directly measured or simulated, for example hardware performance counters.

11.4. Multi core software energy modelling from a network perspective

Raising an energy model from the single core ISA level up to a network of cores poses many challenges. Profiling, modelling and presenting the results are all tasks that require significant effort. These three areas have been addressed in this thesis in Chapters 8 and 9.

An XS1-L core energy model was first extended to work in multiplicity, demonstrated through profiling of the multi-core Swallow system. It was evaluated with single core benchmarks replicated over the system's lattice of processors. It was shown that in this more complex system, the error margins increase as considerations such as heat sensitivity and power supply efficiency have a larger influence over measurements. At the same time, however, the core model is shown to have good robustness with intuitive extensions to account for the most significant differences with larger scale systems.

The communication costs of the Swallow system were profiled through a series of tests that exercised both on- and off-chip links, as well as switches. This allowed a cost per transmitted token to be established with respect to the number of switches used to route each token, combined with the links used during traversal of the route. The model represents the energy in terms of switching activity on capacitative wires, therefore other link lengths or interconnects can be integrated.

This new data was then used to perform additional modelling that was applied on top of the core model, forming a system level model that accounts for networked communication between cores. The system was represented as a graph structure, comprising nodes of cores, switches and potentially external peripherals, connected by edges representing links. The core models could then be attached to the core nodes, and the other model parameters attached as attributes to the other network elements. The `axe` simulator was extended to support a timing model of network communication. The XS1 ISA's dedicated communication instructions allow message passing to be identified in traces and communication energy accumulated accordingly.

The system graph was used to provide visualisation of energy consumption for multi-core code. This allows easy inspection of certain properties of the software, such as which components in the

system do the most work, and within what parts of the system significant communication takes place. Text based energy reports are also provided, identifying for each core, where energy is being consumed. Features of the single-core model, such as function filtering, can be applied across cores for more focused modelling and faster simulation.

The accuracy of these network level models was shown to degrade as the complexity of the programs increases, particularly with respect to the amount of communication taking place. However, the information that can be obtained from the modelling runs retains usefulness in relative terms, giving visual and textual indication to the software developer as to where task allocation or communication strategies could be improved in their software. Improving the accuracy would increase value to the developer by giving them absolute figures that they can expect from their hardware with a measurable degree of confidence. However, just as Swallow is a custom system of XS1-L chips, any other bespoke system would also require further profiling effort in order to establish its particular characteristics.

11.5. The transferability of multi-threaded, multi-core models

An idealised energy model would be effortlessly transferable between hardware architectures, easily accounting for any changes that arise. Similarly, the ideal model could also be exercised at multiple levels of abstraction, from detailed instruction set simulation up to source-code or even abstract software component levels. Existing efforts, including the work presented in this thesis, must be more pragmatic in their implementation and capabilities. However, it is still possible for some degree of transferability to be achieved, both in terms of architecture and abstraction levels. This thesis has demonstrated energy models that possess some opportunities in both of these dimensions of design-space exploration, with active exploitation of some of these already taking place in surrounding work.

Exploring architectures

The first challenge of constructing an energy model is to make that model a good fit to a given piece of hardware, for example a particular processor architecture. The next challenge is to make the model flexible enough to be easily transferred between other architectures and system implementations.

Chapter 10 reviewed a number of other architectures with this second goal in mind. A number of similarities between the XS1-L were identified, as well as differences that would need to be overcome in order to successfully model these processors using the methods that are presented in this work. Further work would need to address these difference if the presented methods of modelling were to be used.

Traversing the abstraction layers

In § 11.3 of this chapter, the raising of the XS1-L core energy model up beyond ISS level was highlighted. In § 11.4, system level energy modelling was added on top of the core model. These are two areas that demonstrate the flexibility of this thesis' modelling approaches with respect to traversing abstraction layers.

As with abstraction from a software engineering perspective, as the view becomes further removed from the underlying detail, so too does the accuracy and degree of understanding that can be obtained. However, this work has shown areas where sufficient information is retained to be useful to a software developer, giving them some degree of transparency through the software and hardware system stack, without requiring them to work at an abstraction level that would not fit well to their needs.

Further work needs to be undertaken in order to improve and expand the traversal of the system stack. This will continue to improve the benefits and insights that an embedded software developer can benefit from.

11.6. Writing energy efficient multi-threaded embedded software

Drawing upon the recommendations made in prior work, as discussed in § 4.1, and building upon the insight gained throughout this thesis, particularly in Part II, a new set of recommendations can be made. These recommendations define steps to develop energy efficient software for MTMC embedded systems.

Choose an algorithm that is appropriate for the target platform

This remains of key importance; one must start with an algorithm that is a good fit to the hardware, otherwise further energy saving efforts will yield minimal returns. However, in a MTMC embedded system, algorithm choice may be different to those chosen historically or in other types of system, for example where cache hierarchies are present or parallelism is not available. Where the system's memory hierarchy is flat and fully predictable, such as in the XS1-L, the ability to express the algorithm concurrently becomes the most important aspect. Not all algorithms lend themselves to concurrency, but in a suitably large program or system, sufficient concurrency may be created through the composition of tasks.

Fully exploit the available parallelism on a core

With the concurrency established above, ensure that a core's available parallelism is utilised to maximum efficiency. In the case of the XS1-L, for example, this means ensuring at least four threads are active, otherwise processor cycles are wasted. One should consider that threads may not be permanently active, and so allocating more than four threads may be necessary to keep the device fully utilised. Only once a core is fully utilised should additional cores then be used.

Place communication intensive tasks close together in the network

Analogous to ensuring that frequently accessed data is kept in cache wherever possible, communicating tasks need to have sufficient bandwidth and a low latency between them to avoid increasing the execution time. The first means of achieving this is through locating these tasks on the same core. However, the number of tasks or threads in the software may require that communication takes place over multiple cores. In this case, ensure that these communications take place over the shortest path possible. Tasks communicating less frequently may incur higher latencies as a result, but this should have a lower impact on the run-time of the program. If communication is used to synchronise multiple tasks, then effort should be put into minimising the latency of this, to avoid long waits, particularly if processor cores would be under-utilised during these waits.

Turn off unused cores and voltage/frequency tune those that remain

Where adequate knowledge of timing requirements can be obtained, cores can have voltage and frequency scaling applied to maximise their energy efficiency whilst still meeting deadlines. If DVFS is available, it may be beneficial to adjust these scaling parameters over the course of execution, provided transitions between them is fast enough. Of course, if a system has more cores than is required for a particular program, then unused cores should be turned off, otherwise voltage and frequency must be scaled as aggressively as possible.

These steps are, for the most part, expressed in order of importance. However, it is the responsibility of programmers to understand their programs as well as target systems and prioritise accordingly. Tools such as the energy modelling software demonstrated in this work can be used to explore design options if there is uncertainty in particular choices, or to validate decisions that have been made.

11.7. Future work

The domain of energy modelling of software possesses many opportunities for future work, including exploitation of this thesis and improvements or extensions to the methods that have been presented. The most compelling of these are explained in this section.

Exploring a wider range of message passing parallel programs

At the multi-core level, this model has used a selection of simple programs that highlight typical communication patterns. To extend this work further, larger programs could be used with more complex communication patterns and real world applications.

The first challenge in undertaking this will be selecting appropriate programs. The ParMiBench benchmark suite [ILG10] features parallel implementations of various embedded benchmarks. However, these no not necessarily map onto the message passing programming model presented by the XMOS software stack. It may be necessary to construct a benchmark suite that is more appropriate, assembling a collection of appropriate existing samples as has been done with BEEBS [PHB13] for single threaded embedded applications, or re-implementing new benchmarks.

With a suitable extended benchmark in place, there are more opportunities for refinement of the modelling techniques, demonstration of their effectiveness and comparison to other methods. This will also allow for more in-depth case studies to be performed in order to give detailed illustrations of working techniques for using software energy modelling to reduce energy consumption in a concurrent embedded system.

Comparing precise network simulation to approximate methods

The network level modelling presented in Chapter 9 abstracts away some of the underlying details of the XS1 architecture's flow control and routing. This incurs a cost to model accuracy, but is traded off against a simpler network simulation implementation.

The simulation framework of `axe` does not currently implement cycle accurate network simulation. Implementing this would be a significant new undertaking. However, doing so would allow a comparison to be drawn between the current simulation and modelling framework and a more precise framework. It can be reasoned that the accuracy of energy predictions from a more precise simulation would also be higher. However, the more important research question in this case, is to establish whether that difference has any significant impact on the usefulness of the modelling itself. The added complexity of the rest of the system may conspire to prevent anything beyond relative energy consumption comparisons being made. The further work would of course need to establish whether this is the case.

Incorporating the XS1-L energy model into other simulation frameworks

Other existing frameworks such as Gem5 [Bin+11] support several architectures, as was discussed in § 3.1.2. The relevant portions of the XS1-L processor behaviour and energy model presented in this thesis could be ported to a system such as Gem5. This would allow a more direct comparison between other architectures implemented in the same framework.

The most significant challenge in this would be addressing multi-core. The multi-core systems typically demonstrated on Gem5 use memory hierarchies and cache coherency mechanisms to provide shared memory programming models and inter-processor communication. A channel based model that is more appropriate for the XS1-L would need to be contributed in order for similar multi-core tests to be evaluated.

The main contribution of this effort would be broader scope for energy-aware design space exploration. It would allow a developer to sample many system types and configuration, enabling them to find a suitable target system.

In effect, this endeavour could turn one of the key research questions of this thesis around. Instead of asking "how do I make my good software a good fit to the hardware?" a developer could ask "what is the best hardware to suit the structure of the software that I am developing?"

Static analysis of network communication

The core model presented in this thesis has already been demonstrated in static analysis (§ 11.3). However, there remains opportunity to raise the analysis of the multi-core network aware modelling in the same manner.

Static analysis at the multi-core level would need to be supplied various cost parameters on top of the existing ISA costs. This would include the communication cost between any connected pair of channel ends in the target software. The programming model for the XS1 in XC simplifies this, with clear allocation of code to cores and visibility of which threads are connected through channels. The underlying model would need to be provided with sufficient data to extract a route through the target system (described in an XN platform description file). Then configuration data for the links and established interconnect costs, it could provide a cost function in terms of data packets sent, where shorter packets incur higher relative costs due to the overhead of cut-through switching setup and header tokens.

In addition to this, network contention may also need to be considered. However, static analysis can assist in determining which channels may contend for particular routes in the system, in order to further parameterise the communication cost functions in terms of the likelihood of delays caused by congestion.

Further analysis may also be able to propose improvements to the task layout across the available cores, or changes to the communication structure. This extends beyond analysis and design space exploration, and into optimisation.

Case study

A suitably large software project could be examined using the tools and techniques presented in this work. The project could be new, in which case these tools would aid in the design exploration process, or it could be existing and this work used to seek to improve the code. Exercising the work on this scale would be both a good test of its effectiveness as well as a demonstrator for how to use it on real-world software.

Application of techniques to other systems

Although some of the findings in this work are specific to the platforms examined, a number of techniques were used or further developed that could be reapplied to other types of system or architecture. For example, the instruction level profiling framework `xmprofile`, could form the basis for constrained test generation of other devices. Similarly, the model construction techniques, involving a combination of directly profiled instructions and regression-tree solved instruction costs, could be applied elsewhere and its accuracy evaluated across a wider range of systems.

11.8. Concluding remarks

This thesis has formed a collection of background research, proposed new techniques and performed subsequent experimentation and evaluation, all with the aim of enabling software developers to better understand the energy consumption of software in modern embedded systems.

Through this work, it has been shown how multi-threaded, multi-core processors can be profiled in order to determine energy consumption information in relation to their ISA. This enables the construction of ISA level energy models, several iterations of which are then presented in this thesis. The models were subsequently extended to incorporate system level elements, specifically network communication, which is identifiable within the channel communication model present in the ISA.

These energy models have then been used in conjunction with Instruction Set Simulation (ISS) to provide energy estimates for single-threaded, multi-threaded and multi-core embedded benchmarks. Error margins at the core level were shown to be an average of 2.67 %. At a multi-core system level, error increases, but useful observations were still able to be seen in with these models, giving a software developer information to help them refine the energy efficiency of their design.

The models have been shown to have uses beyond ISS. Static software analysis is being applied against the presented core level models in follow-on work, and future work was outlined to perform

static analysis at the network level. Additional future work in other areas of interest was also laid out, including porting the model to other analysis frameworks that incorporate collections of architectures, and implementing a more precise network model in order to increase accuracy of multi-core modelling, particularly in larger networks.

In closing, given that the impact of ICT and energy consumption on our lives and our planet is clear, we must continue in our efforts to find intelligent, responsible ways to lessen it. Embedded systems are the most prolific computing devices on the planet, so their impact is significant and ever-growing. If developers of software for these systems can be given more control over the energy consumption of their code, then they have the tools that they need to affect a positive impact on global ICT energy consumption. Software developers can then take an active role in a community of engineers that want to bring more efficient devices into the world. This thesis has provided some means of enabling this. It is the will of the author that this effort continue with vigour.

List of acronyms

ADC Analog-to-Digital Converter. 64

AHB Advanced High-performance Bus. 139

ALU Arithmetic Logic Unit. 34

AVX Advanced Vector Extensions. 35

CAN Controller Area Network. 30

CMOS Complementary Metal Oxide Semiconductor. 51

CPU Central Processing Unit. 31, 32, 71

CRC Cyclic Redundancy Check. 53

CSP Communicating Sequential Processes. 31, 63

CSV Comma Separated Value. 126

DDR Double Data Rate. 41

DFS Dynamic Frequency Scaling. 66

DMA Direct Memory Access. 62

DRAM Dynamic Random Access Memory. 41, 65, 67, 135, 136

DSP Digital Signal Processor. 35, 42, 61

DUT Device Under Test. 80–82, 114

DVFS Dynamic Voltage and Frequency Scaling. 36, 41, 48, 50, 51, 53, 54, 56, 57, 139, 141, 143, 147

EDP Energy Delay Product. 40

EPI Energy Per Instruction. 138

FNOP "fetch no-op". 61, 84, 101, 102, 120, 121, 126

FPGA Field Programmable Gate Array. 31, 60, 135

FPU Floating Point Unit. 34, 35, 39, 135

FU Functional Unit. 34, 35, 49

Gbps Gigabits per second. 63, 70

GDDR Graphics Double Data Rate. 137

GP-GPU General Purpose GPU. 35, 47, 137

GPIO General Purpose Input/Output. 60, 67, 114

GPU Graphics Processing Unit. 31, 35, 137

HPC High Performance Computing. 44, 137

I²C Inter-Integrated Circuit. 30, 60, 67

I/O Input/Output. 20, 30, 32, 53, 56, 59, 60, 62, 65–67, 72, 82, 88, 114–116, 119, 123, 133, 136, 140

ICT Information and Communication Technology. 17, 18, 150

ILP Instruction Level Parallelism. 34

IoT Internet of Things. 17

IPC Instructions Per Clock. 33–35, 41

IR Intermediate Representation. 109

ISA Instruction Set Architecture. 21–23, 35, 39, 41, 42, 45, 49, 59–64, 68, 71, 72, 75, 77, 83–85, 100, 101, 104, 109, 117, 119, 120, 122, 124, 139–141, 143–145, 149

ISR Interrupt Service Routine. 32, 56

ISS Instruction Set Simulation. 87, 104, 106, 120, 122, 145, 146, 149

ITRS International Technology Roadmap for Semiconductors. 55

JIT Just In Time. 120

JSON JavaScript Object Notation. 120, 133

JTAG Joint Test Action Group. 21, 65–68, 112, 114

KB Kilobyte. 67

LED Light Emitting Diode. 66, 115

LLVM Low Level Virtual Machine. 109

LoA List of Acronyms. 23

MB Megabyte. 67

Mb Megabit. 67

Mbps Megabits per second. 53, 70, 116

McPAT Multi-core Power Analysis and Timing. 40

MII Media Independent Interface. 53, 60

MIMD Multiple Instruction Multiple Data. 29, 35

MIPS Million Instructions Per Second. 61

MPI Message Passing Interface. 31

MPSoC Multi Processor System on Chip. 42, 56

MTMC Multi-Threaded and Multi-Core. 21, 23, 29, 36, 39, 57, 59, 64, 135, 143, 147

NoC Network on Chip. 135, 137, 141

NTV Near-Threshold Voltage. 55

NUMA Non-Uniform Memory Architecture. 49

OLS Ordinary Least-Squares. 102–104

OpenCL Open Compute Language. 31

OpenMP Open Multi-Processing. 31

OS Operating System. 29–32, 35, 36, 43, 47, 56, 59, 60

PCB Printed Circuit Board. 111, 125

PGAS Partitioned Global Address Space. 135

PHY Physical layer. 64, 67

PLL Phase Locked Loop. 88

POSIX Portable Operating System Interface. 30–32

RAM Random Access Memory. 48, 140

RISC Reduced Instruction Set Computer. 61

ROB Re-Order Buffer. 34

RTL Register Transfer Logic. 79

RTOS Real-Time Operating System. 30, 32, 48, 56, 62

SIMD Single Instruction Multiple Data. 29, 35

SISD Single Instruction Single Data. 29, 35

SoC System on Chip. 139, 140

SoP Start of Packet. 53

SPI Serial Peripheral Interface. 60, 67

SRAM Static Random Access Memory. 42, 67, 80

SSE Streaming SIMD Extensions. 35

STV Sub-Threshold Voltage. 52, 55

TAP Test Access Port. 67, 68

TDM Time Division Multiplexing. 36

TDP Thermal Design Power. 36

TFTP Trivial File Transfer Protocol. 21

TLM Transaction Level Modelling. 41, 42

USB Universal Serial Bus. 60

VCD Value Change Dump. 120

VLIW Very Long Instruction Word. 35, 42, 140, 141

WCEC Worst Case Energy Consumption. 109

WCET Worst Case Execution Time. 107, 109

XML eXtensible Markup Language. 40

Bibliography

[Ada13] Adapteva. *Epiphany Architecture Reference*. Tech. rep. 2013.

[Ada15] Adapteva. *Ephiphany Introduction*. 2015. URL: http://www.adapteva.com/introduction/ (visited on 03/30/2015).

[Adl10] Mark Adler. *Pigz: A parallel implementation of gzip for modern multi-processor, multi-core machines*. 2010. URL: https://github.com/madler/pigz.

[Amd67] Gene M Amdahl. "Validity of the single processor approach to achieving large scale computing capabilities". In: *AFIPS spring joint computer conference*. Vol. 34. 4. ACM, 1967, pp. 483–485. DOI: 10.1145/1465482.1465560. URL: http://dl.acm.org/citation.cfm?id=1465560.

[AND07] Rabie Ben Atitallah, Smail Niar, and Jean-luc Dekeyser. "MPSoC power estimation framework at transaction level modeling". In: *2007 Internatonal Conference on Microelectronics*. IEEE, Dec. 2007, pp. 245–248. ISBN: 978-1-4244-1846-6. DOI: 10.1109/ICM.2007.4497703. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4497703.

[ANG08] H Amur, Ripal Nathuji, and M Ghosh. "IdlePower: Application-aware management of processor idle states". In: *Proceedings of MMCS* (2008). URL: http://www.cc.gatech.edu/grads/h/hamur3/mmcs08_angsl.pdf.

[ARM12] ARM. *ARM Launches Cortex-A50 Series, the World's Most Energy-Efficient 64-bit Processors*. 2012. URL: http://www.arm.com/about/newsroom/arm-launches-cortex-a50-series-the-worlds-most-energy-efficient-64-bit-processors.php (visited on 03/31/2015).

[ARM14] ARM. *NEON*. 2014. URL: http://www.arm.com/products/processors/technologies/neon.php.

[AS83] Gregory R. Andrews and Fred B. Schneider. *Concepts and Notations for Concurrent Programming*. 1983. DOI: 10.1145/356901.356903.

[Aus02] Todd Austin. "SimpleScalar: An Infrastructure for computer system modeling". In: *IEEE Computer* February (2002), pp. 59–67.

[Bat+09] Luis Angel D. Bathen, Yongjin Ahn, Sudeep Pasricha, and Nikil D. Dutt. "A Methodology for Power-aware Pipelining via High-Level Performance Model Evaluations". In: *2009 10th International Workshop on Microprocessor Test and Verification*. Ieee, Dec. 2009, pp. 19–24. ISBN: 978-1-4244-6479-1. DOI: 10.1109/MTV.2009.19. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5460786.

[BC05] Daniel Bovet and Marco Cesati. *Understanding The Linux Kernel*. Oreilly & Associates Inc, 2005. ISBN: 0596005652.

[BDM09] Geoffrey Blake, Ronald G. Dreslinski, and Trevor Mudge. "A survey of multicore processors: A review of their common attributes". In: *IEEE Signal Processing Magazine* 26.6 (2009), pp. 26–37. ISSN: 10535888. DOI: 10.1109/MSP.2009.934110.

[Bin+06] N.L. Binkert, R.G. Dreslinski, L.R. Hsu, K.T. Lim, A.G. Saidi, and S.K Reinhardt. "The M5 simulator: Modeling networked systems". In: *IEEE Micro* 26.4 (2006), pp. 52–60. URL: https://heterogenous-thread-assignment-sim.googlecode.com/files/01677503.pdf.

[Bin+11] Nathan Binkert, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, David A. Wood, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, and Tushar Krishna. *The gem5 simulator*. Aug. 2011. DOI: 10.1145/2024716.2024718. URL: http://dl.acm.org/citation.cfm?doid=2024716.2024718%20http://dl.acm.org/citation.cfm?id=2024718.

[Blu+07] H Blume, D Becker, L Rotenberg, M Botteck, J Brakensiek, and T Noll. "Hybrid functional- and instruction-level power modeling for embedded and heterogeneous processor architectures". In: *Journal of Systems Architecture* 53.10 (Oct. 2007), pp. 689–702. ISSN: 13837621. DOI: 10.1016/j.sysarc.2007.01.002. URL: http://linkinghub.elsevier.com/retrieve/pii/S1383762107000161.

[Boh07] Mark Bohr. *A 30 Year Retrospective on Dennard's MOSFET Scaling Paper*. 2007. DOI: 10.1109/N-SSC.2007.4785534.

[BR06] Paulo Francisco Butzen and Renato Perez Ribas. *Leakage current in sub-micrometer cmos gates*. Tech. rep. Universidade Federal do Rio Grande do Sul, 2006. URL: http://www.inf.ufrgs.br/logics/docman/book_emicro_butzen.pdf.

[Bre+84] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, 1984. ISBN: 978-0412048418.

[BSS07] Giovanni Beltrame, Donatella Sciuto, and Cristina Silvano. "Multi-Accuracy Power and Performance Transaction-Level Modeling". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 26.10 (Oct. 2007), pp. 1830–1842. ISSN: 0278-0070. DOI: 10.1109/TCAD.2007.895790. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4305240.

[BTM00] David Brooks, Vivek Tiwari, and Margaret Martonosi. *Wattch: A Framework for Architectural-Level Power Analysis and Optimizations*. May 2000. DOI: 10.1145/342001.339657. URL: http://portal.acm.org/citation.cfm?doid=342001.339657.

[Bur+00] Thomas D. Burd, Trevor A. Pering, Anthony J. Stratakos, and Robert W. Brodersen. "Dynamic voltage scaled microprocessor system". In: *IEEE Journal of Solid-State Circuits* 35.11 (Nov. 2000), pp. 1571–1580. ISSN: 00189200. DOI: 10.1109/4.881202. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=881202.

[But+11] Michael Butler, Leslie Barnes, Debjit Das Sarma, and Bob Gelinas. "Bulldozer: An approach to multithreaded compute performance". In: *IEEE Micro*. Vol. 31. 2. Mar. 2011, pp. 6–15. ISBN: 0272-1732 VO - 31. DOI: 10.1109/MM.2011.23. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5751937.

[But+12] Anastasiia Butko, Rafael Garibotti, Luciano Ost, and Gilles Sassatelli. "Accuracy evaluation of GEM5 simulator system". In: *ReCoSoC 2012 - 7th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip, Proceedings*. IEEE, July 2012, pp. 1–7. ISBN: 9781467325721. DOI: 10.1109/ReCoSoC.2012.6322869. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6322869.

[CCK12] Hyun-duk Cho, Kisuk Chung, and Taehoon Kim. *Benefits of the big.LITTLE Architecture*. Tech. rep. 2012, None. URL: http://www.samsung.com/global/business/semiconductor/minisite/Exynos/data/benefits.pdf.

[CH10] Aaron Carroll and Gernot Heiser. "An analysis of power consumption in a smartphone". In: *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*. USENIXATC'10. Berkeley, CA, USA: USENIX Association, 2010, p. 21. URL: http://portal.acm.org/citation.cfm?id=1855840.1855861.

[CLH09] Da-Ren Chen, Tasi-Duan Lin, and Shu-Ming Hsieh. "A Transition-Aware DVS Method for Jitter-Controlled Real-Time Scheduling". In: *2009 International Conference on Parallel and Distributed Computing, Applications and Technologies* (Dec. 2009), pp. 34–41. DOI: 10.1109/PDCAT.2009.20. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5372824.

[CM05] Gilberto Contreras and Margaret Martonosi. "Power prediction for Intel XScale processors using performance monitoring unit events". In: *ISLPED '05. Proceedings of the International Symposium on Low Power Electronics and Design.* (2005), pp. 221–226. ISSN: 15334678. DOI: 10.1109/LPE.2005.195518. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1522767.

[CSP04] Kihwan Choi Kihwan Choi, R. Soma, and M. Pedram. "Dynamic Voltage and Frequency Scaling based on Workload Decomposition". In: *Proceedings of the 2004 International Symposium on Low Power Electronics and Design* (2004), pp. 174–179. DOI: 10.1109/LPE.2004.1349330.

[Cza+12] Tomasz S Czajkowski, Utku Aydonat, Dmitry Denisenko, John Freeman, Michael Kinsner, David Neto, Jason Wong, Peter Yiannacouras, and Deshanand P. Singh. "From opencl to high-performance hardware on FPGAS". In: *22nd International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, Aug. 2012, pp. 531–534. ISBN: 978-1-4673-2256-0. DOI: 10.1109/FPL.2012.6339272. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6339272.

[Dan+12] Andrew Danowitz, Kyle Kelley, James Mao, John P. Stevenson, and Mark Horowitz. "CPU DB: Recording Microprocessor History". In: *ACM Queue* 10 (2012), p. 10. ISSN: 15427730. DOI: 10.1145/2181796.2181798.

[DGY74] Robert H. Dennard, F.H Gansslen, and H-N. YU. "Design of Ion Implanted MOSFET's with very small physical dimensions". In: *IEEE Journal of Solid-State Circuits* SC-9.5 (1974), p. 256. ISSN: 0018-9200. DOI: 10.1109/JSSC.1974.1050511.

[DKC08] Tien Van Do, Udo R. Krieger, and Ram Chakka. "Performance modeling of an Apache Web server with a dynamic pool of service processes". In: *Telecommunication Systems* 39.2 (June 2008), pp. 117–129. ISSN: 1018-4864. DOI: 10.1007/s11235-008-9116-y. URL: http://link.springer.com/10.1007/s11235-008-9116-y.

[DL06] Clara Dismuke and Richard Lindrooth. "Ordinary least squares". In: *Methods and Designs for Outcomes Research* (2006), pp. 93–104.

[DM98] Leonaxdo Dagum and Ramesh Menon. "OpenMP: an industry standard API for shared-memory programming". In: *IEEE Computational Science and Engineering* 5.1 (1998), pp. 46–55. ISSN: 10709924. DOI: 10.1109/99.660313. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=660313.

[DPT03] Andrew Duller, Gajinder Panesar, and Daniel Towner. "Parallel Processing the picoChip way". In: *Communicating Processing Architectures* (2003), pp. 299–312. ISSN: 13837575.

[Dre07] Ulrich Drepper. "What every programmer should know about memory". In: *Red Hat, Inc* 3 (2007), p. 114. ISSN: 0361526X. DOI: 10.1.1.91.957.

[Ead11] Douglas Eadline. *May's Law and Parallel Software*. 2011. URL: http://www.linux-mag.com/id/8422/ (visited on 02/10/2015).

[EZC09] EZChip Semicondutor. *TILE-Gx72 Processor*. Tech. rep. 2009, pp. 1–2.

[EZC13] EZChip Semicondutor. *Overview of the TilePro Series Tile Processor Architecture*. Tech. rep. 2013.

[FDF98] Paolo Farahoschi, Giuseppe Desoli, and Joseph A. Fisher. *Latest word in digital and media processing*. Mar. 1998. DOI: 10.1109/79.664698. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=664698.

[Fly72] Michael J. Flynn. "Some Computer Organizations and Their Effectiveness". In: *IEEE Transactions on Computers* C-21.9 (Sept. 1972), pp. 948–960. ISSN: 0018-9340. DOI: 10.1109/TC.1972.5009071. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5009071.

[Gem14] Gem5. *Gem5*. 2014. URL: http://gem5.org/.

[GKE14] Kyriakos Georgiou, Steve Kerrison, and Kerstin Eder. *A Multi-level Worst Case Energy Consumption Static Analysis for Single and Multi-threaded Embedded Programs*. Tech. rep. University of Bristol, 2014. URL: http://www.cs.bris.ac.uk/Publications/pub_master.jsp?id=2001701.

[GLP07] Olga Golubeva, Mirko Loghi, and Massimo Poncino. "On the energy efficiency of synchronization primitives for shared-memory single-chip multiprocessors". In: *GLSVLSI '07: Proceedings of the 17th ACM Great Lakes symposium on VLSI*. 2007, pp. 489–492. ISBN: 978-1-59593-605-9. DOI: http://doi.acm.org/10.1145/1228784.1228900.

[Gre+15] Neville Grech, Kyriakos Georgiou, James Pallister, Steve Kerrison, Jeremy Morse, and Kerstin Eder. "Static analysis of energy consumption for LLVM IR programs". In: *Proceedings of the 18th International Workshop on Software and Compilers for Embedded Systems*. SCOPES '15. Sankt Goar, Germany: ACM, 2015. DOI: 10.1145/2764967.2764974.

[Gre11] Peter Greenhalgh. "big.LITTLE Processing with ARM Cortex-A15 & Cortex-A7". In: *ARM White Paper* September 2011 (2011), pp. 1–8.

[GT90] Gary Graunke and Shreekant Thakkar. "Synchronization algorithms for shared-memory multiprocessors". In: *Computer* 23.6 (1990), pp. 60–69. ISSN: 00189162. DOI: 10.1109/2.55501.

[Han14] James W Hanlon. "Scalable abstractions for general-purpose parallel computation". PhD thesis. University of Bristol, 2014. URL: http://www.jwhanlon.com/thesis.php.

[Hei+12] Wim Heirman, Souradip Sarkar, Trevor E Carlson, Ibrahim Hur, and Lieven Eeckhout. "Power-aware multi-core simulation for early design stage hardware/software co-optimization". In: *Proceedings of the 21st international conferencge hardware/software co-optimizatione on Parallel architectures and compilation techniques - PACT '12*. New York, New York, USA: ACM Press, 2012, p. 3. ISBN: 9781450311823. DOI: 10.1145/2370816.2370820. URL: http://dl.acm.org/citation.cfm?doid=2370816.2370820.

[HK15] Simon J. Hollis and Steve Kerrison. "Overview of Swallow — A Scalable 480-core System for Investigating the Performance and Energy Efficiency of Many-core Applications and Operating Systems". In: *arXiv* (2015).

[HMM15] Simon J Hollis, Edward Ma, and Radu Marculescu. *nOS: a nano-sized distributed operating system for resource optimisation on many-core systems*. Tech. rep. 2015.

[Hoa78] C. a. R. Hoare. *Communicating sequential processes*. 1978. DOI: 10.1145/359576.359585.

[Hol12] Simon J. Hollis. *Swallow many-core research project*. 2012. URL: http://www.cs.bris.ac.uk/Research/Micro/swallow.jsp (visited on 02/24/2015).

[HP 14] HP Labs. *CACTI*. 2014. URL: http://www.hpl.hp.com/research/cacti/.

[HP06] John L Hennessy and David a Patterson. *Computer Architecture, Fourth Edition: A Quantitative Approach*. 2006, p. 704. ISBN: 0123704901. DOI: 10.1.1.115.1881. URL: http://portal.acm.org/citation.cfm?id=1200662.

[ILG10] Syed Muhammad Zeeshan Iqbal, Yuchen Liang, and Håkan Grahn. "ParMiBench - An open-source benchmark for embedded multiprocessor systems". In: *IEEE Computer Architecture Letters* 9.2 (Feb. 2010), pp. 45–48. ISSN: 15566056. DOI: 10.1109/L-CA.2010.14. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5550920.

[Int03a] Intel Corporation. *Intel Hyper-Threading Technology Technical User's Guide*. 2003. URL: http://cache-www.intel.com/cd/00/00/01/77/17705_htt_user_guide.pdf.

[Int03b] Intel Corporation. "The Intel Pentium M Processor: Microarchitecture and Performance". In: *Intel Technology Journal* 07.02 (2003).

[Int11] Intel Corporation. *Intel 64 and IA-32 Architectures Software Developer's Manual, Combined Volumes*. December. 2011, p. 3463.

[Int15] Intel Corporation. *Intel Turbo Boost Technology 2.0*. 2015. URL: http://www.intel.com/content/www/us/en/architecture-and-technology/turbo-boost/turbo-boost-technology.html (visited on 03/01/2015).

[IRF08] Mostafa E. a. Ibrahim, Markus Rupp, and Hossam a. H. Fahmy. "Power estimation methodology for VLIW Digital Signal Processors". In: *2008 42nd Asilomar Conference on Signals, Systems and Computers*. 1. IEEE, Oct. 2008, pp. 1840–1844. ISBN: 978-1-4244-2940-0. DOI: 10.1109/ACSSC.2008.5074746. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5074746.

[IRH08] MEA Ibrahim, Markus Rupp, and SED Habib. *Power consumption model at functional level for VLIW digital signal processors*. Tech. rep. 1. 2008, pp. 2–7. URL: http://www.researchgate.net/publication/228947933_Power_consumption_model_at_functional_level_for_VLIW_digital_signal_processors/file/e0b49521c2bc72bd43.pdf.

[JGL09] Kirsten Jacobs, Huw Geddes, and Mark Lippett. *XSIM User Guide*. 2009.

[Joh89] William M Johnson. *Super-Scalar Processor Design*. Tech. rep. June. Stanford University, 1989. DOI: 10.1.1.16.3573.

[Kah13] Andrew B Kahng. "The ITRS design technology and system drivers roadmap". In: *Proceedings of the 50th Annual Design Automation Conference on - DAC '13*. New York, New York, USA: ACM Press, 2013, p. 1. ISBN: 9781450320719. DOI: 10.1145/2463209.2488776. URL: http://dl.acm.org/citation.cfm?doid=2463209.2488776.

[Kam10] Poul-Henning Kamp. *You're doing it wrong*. 2010. DOI: 10.1145/1785414.1785434.

[KAO05] Poonacha Kongetira, Kathirgamar Aingaran, and Kunle Olukotun. "Niagara: A 32-Way Multithreaded Sparc Processor". In: *IEEE Micro* 25.2 (Mar. 2005), pp. 21–29. ISSN: 0272-1732. DOI: 10.1109/MM.2005.35. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1453485.

[Kau+12] Himanshu Kaul, Mark Anders, Steven Hsu, Amit Agarwal, Ram Krishnamurthy, and Shekhar Borkar. "Near-threshold voltage (NTV) design: opportunities and challenges". In: *49th Annual Design Automation Conference*. 2012, pp. 1153–1158. ISBN: 9781450311991. DOI: 10.1145/2228360.2228572. URL: http://dl.acm.org/citation.cfm?id=2228572.

[KE14] Steve Kerrison and Kerstin Eder. "Measuring and modelling the energy consumption of multi-threaded, multi-core embedded software". In: *ICT Energy Letters* (July 2014), pp. 18–19. URL: http://www.nanoenergyletters.com/files/nel/ICT-Energy_Letters_8.pdf.

[KE15a] Steve Kerrison and Kerstin Eder. "A software controlled voltage tuning system using multi-purpose ring oscillators". In: *arXiv* (2015). arXiv: 1503.05733. URL: https://arxiv.org/abs/1503.05733.

[KE15b] Steve Kerrison and Kerstin Eder. "Energy modelling of software for a hardware multithreaded embedded microprocessor". In: *Transactions on Embedded Computer Systems (TECS)* (2015).

[Ker12a] Steve Kerrison. *AXE (An Xcore Emulator) fork*. 2012. URL: https://github.com/stevekerrison/tool_axe/tree/axe_json (visited on 03/30/2015).

[Ker12b] Steve Kerrison. *Swallow ethernet loader*. 2012. URL: https://github.com/stevekerrison/sw_swallow_etherboot (visited on 03/27/2015).

[Ker14] Steve Kerrison. *Swallow XN generator for XMOS v13+ tools*. 2014. URL: https://github.com/stevekerrison/tool_swallow-gen-xn (visited on 03/28/2015).

[KiC15] KiCad. *KiCad EDA Software Suite*. 2015. URL: http://www.kicad-pcb.org/ (visited on 04/03/2015).

[Kim+03] NS Kim, T Austin, D Baauw, and T Mudge. "Leakage current: Moore's law meets static power". In: *Computer* (2003), pp. 68–75. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1250885.

[KK06] Ranjith Kumar and Volkan Kursun. "Reversed temperature-dependent propagation delay characteristics in nanometer CMOS circuits". In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 53.10 (2006), pp. 1078–1082. ISSN: 10577130. DOI: 10.1109/TCSII.2006.882218.

[Kow88] Janusz Kowalik. *ACTORS: A Model of Concurrent Computation in Distributed Systems (Gul Agha)*. 1988. DOI: 10.1137/1030027.

[Kuh09] Kelin J. Kuhn. "Moore's law past 32nm: Future challenges in device scaling". In: *Proceedings - 2009 13th International Workshop on Computational Electronics, IWCE 2009*. 2009, pp. 1–4. ISBN: 9781424439270. DOI: 10.1109/IWCE.2009.5091124.

[LE94] Michael Luby and Wolfgang Ertel. "Optimal parallelization of Las Vegas algorithms". In: 89. 1994, pp. 461–474. ISBN: 3-540-57785-8. DOI: 10.1007/3-540-57785-8_163. URL: http://link.springer.com/chapter/10.1007/3-540-57785-8%5C_163%20http://link.springer.com/10.1007/3-540-57785-8%5C_163.

[LEM01] Sheayun Lee, Andreas Ermedahl, and Sang Lyul Min. "An Accurate Instruction-Level Energy Consumption Model for Embedded RISC Processors". In: *ACM SIGPLAN Notices* 36.8 (Aug. 2001), pp. 1–10. ISSN: 03621340. DOI: 10.1145/384196.384201. URL: http://portal.acm.org/citation.cfm?doid=384196.384201.

[Li+09] Sheng Li, Jung Ho Ahn, Richard D Strong, Jay B Brockman, Dean M Tullsen, and Norman 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*. c. New York, New York, USA: ACM Press, 2009, p. 469. ISBN: 9781605587981. DOI: 10.1145/1669112.1669172. URL: http://portal.acm.org/citation.cfm?doid=1669112.1669172.

[Liq+15] Umer Liqat, Steven Kerrison, Serrano Alejandro, Kyriakos Giorgiou, Pedro Lopez-Garcia, Neville Grech, Manuel V. Hermenegildo, and Kerstin Eder. "Energy Consumption Analysis of Programs based on XMOS ISA-Level Models". In: *23rd International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR'13)*. Springer, Sept. 2015.

[Lom11] Chris Lomont. *Introduction to Intel Advanced Vector Extensions*. Tech. rep. Intel, 2011, p. 21. URL: http://www.obpm.org/download/Intro_to_Intel_AVX.pdf.

[Mar+05] Milo MK Martin, Daniel J Sorin, Bradford M Beckmann, Michael R Marty, Min Xu, Alaa R Alameldeen, Kevin E Moore, Mark D Hill, and David A Wood. "Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset". In: *ACM SIGARCH Computer Architecture News* 33.September (2005), pp. 92–99. URL: http://dl.acm.org/citation.cfm?id=1105747.

[Mar11] Peter Marwedel. *Embedded System Design*. Dordrecht: Springer Netherlands, 2011. ISBN: 978-94-007-0256-1. DOI: 10.1007/978-94-007-0257-8. URL: http://link.springer.com/10.1007/978-94-007-0257-8.

[Mas87] Henry Massalin. *Superoptimizer: a look at the smallest program*. 1987. DOI: 10.1145/36205.36194.

[Mat10] Nick Mathewson. *Fast portable non-blocking network programming with Libevent*. Online, 2010. URL: http://www.wangafu.net/~nickm/libevent-book/.

[May+08] David May, Ali Dixon, Ayewin Oung, Henk Muller, and Mark Lippett. *XS1-L System Specification*. 2008.

[May09a] David May. *The XMOS XS1 Architecture*. 2009. ISBN: 9781907361012.

[May09b] David May. *XMOS XS1 Instruction Set Architecture*. 2009.

[Mcc02] Dave Mccracken. "POSIX Threads and the Linux Kernel". In: *Ottowa Linux Symposium*. 2002, pp. 330–337.

[Mic12] Microsoft Corporation. *About Processes and Threads (Windows)*. 2012. URL: http://msdn.microsoft.com/en-us/library/windows/desktop/ms681917(v=vs.85).aspx.

[Mic14] Microsoft Corporation. *Windows Products Support Lifecycle FAQ*. 2014. URL: https://support.microsoft.com/en-gb/gp/lifewinfaq#Microsoft-Windows-Embedded (visited on 04/06/2015).

[Moo65] G E Moore. "Cramming more components onto integrated circuits". In: *Electronics* 38.8 (1965), pp. 114–117. ISSN: 1098-4232. DOI: 10.1109/N-SSC.2006.4785860. URL: papers3://publication/uuid/8E5EB7C8-681C-447D-9361-E68D1932997D.

[Net12] NetworkX Developers. *NetworkX*. 2012. URL: http://networkx.lanl.gov/ (visited on 03/30/2015).

[NL13] Jose Nunez-Yanez and Geza Lore. "Enabling accurate modeling of power and energy consumption in an ARM-based System-on-Chip". In: *Microprocessors and Microsystems* 37.3 (May 2013), pp. 319–332. ISSN: 01419331. DOI: 10.1016/j.micpro.2012.12.004. URL: http://linkinghub.elsevier.com/retrieve/pii/S0141933113000021.

[ON 10] ON Semiconductor. *NCP1529 Low Ripple , Adjustable Output Voltage Step-down Converter*. 2010. URL: http://www.onsemi.com/pub_link/Collateral/NCP1529-D.PDF.

[OrB14a] Zvi Or-Bach. *Intel vs. Intel*. 2014. URL: http://www.eetimes.com/author.asp?doc_id=1323497 (visited on 03/27/2015).

[OrB14b] Zvi Or-Bach. *Moore's Law has stopped at 28nm*. 2014. URL: http://electroiq.com/blog/2014/03/moores-law-has-stopped-at-28nm/ (visited on 03/27/2015).

[Osb11] Richard Osborne. *AXE (An Xcore Emulator)*. 2011. URL: https://github.com/xcore/tool_axe (visited on 05/04/2015).

[Pat85] David a. Patterson. "Reduced instruction set computers". In: *Communications of the ACM* 28.1 (Jan. 1985), pp. 8–21. ISSN: 00010782. DOI: 10.1145/2465.214917. URL: http://portal.acm.org/citation.cfm?doid=2465.214917.

[PEH14] James Pallister, Kerstin Eder, and Simon Hollis. "Optimizing the flash-RAM energy trade-off in deeply embedded systems". 2014. URL: http://arxiv.org/abs/1406.0403.

[PHB13] James Pallister, Simon Hollis, and Jeremy Bennett. "BEEBS: Open Benchmarks for Energy Measurements on Embedded Platforms". 2013. URL: http://arxiv.org/abs/1308.5174.

[Phi04] Philip A. Laplante. *Real-time Systems Design and Analysis*. 2004, p. 529. ISBN: 3175723993. DOI: 10.1002/0471648299. URL: http://www.springerreference.com/.

[PHZ11] Abhinav Pathak, Y Charlie Hu, and Ming Zhang. "Bootstrapping energy debugging on smartphones". In: *Proceedings of the 10th ACM Workshop on Hot Topics in Networks - HotNets '11*. New York, New York, USA: ACM Press, 2011, pp. 1–6. ISBN: 9781450310598. DOI: 10.1145/2070562.2070567. URL: http://doi.acm.org/10.1145/2070562.2070567.

[Pic+08] Mario Pickavet, Willem Vereecken, Sofie Demeyer, Pieter Audenaert, Brecht Vermeulen, Chris Develder, Didier Colle, Bart Dhoedt, and Piet Demeesterl. "Worldwide energy needs for ICT: The rise of power-aware networking". In: *2008 2nd International Symposium on Advanced Networks and Telecommunication Systems, ANTS 2008*. December. 2008, pp. 15–17. ISBN: 1424436001. DOI: 10.1109/ANTS.2008.4937762.

[Pin98] Keshav Pingali. *Parallel Programming Languages*. Tech. rep. Cornell University, 1998, pp. 1–24.

[PW99] Massoud Pedram and Qing Wu Qing Wu. "Design considerations for battery-powered electronics". In: *Proceedings 1999 Design Automation Conference (Cat. No. 99CH36361)*. 1999, pp. 861–866. ISBN: 1-58113-092-9. DOI: 10.1109/DAC.1999.782166.

[Qu+00] Gang Qu, Naoyuki Kawabe, Kimiyoshi Usami, and Miodrag Potkonjak. "Function-level power estimation methodology for microprocessors". In: *Proceedings of the 37th conference on Design automation - DAC '00* (2000), pp. 810–813. DOI: 10.1145/337292.337786. URL: http://portal.acm.org/citation.cfm?doid=337292.337786.

[RA06] Jason Roberts and Shameem Akhter. *Multi-Core Programming: Increasing Performance through Software Multi-threading*. Intel Press, 2006, p. 22. ISBN: 0976483246.

[Rei99] Edwin D Reilly. "Memory-mapped I/O". In: *Encyclopedia of Computer Science*. 4th ed. Chichester: John Wiley and Sons, 1999, p. 1152. ISBN: 0-470-86412-5.

[Ret+14] Santhosh Kumar Rethinagiri, Oscar Palomar, Rabie Ben Atitallah, Smail Niar, Osman Unsal, and Adrian Cristal Kestelman. "System-level power estimation tool for embedded processor based platforms". In: *6th Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools*. 2014. URL: http://dl.acm.org/citation.cfm?id=2555491.

[RJ00] Glen Reinman and Norman P Jouppi. *CACTI 2.0: An Integrated Cache Timing and Power Model*. Tech. rep. 2000, p. 24. URL: http://arch.cs.utah.edu/cacti/cacti2.pdf.

[RJ97] Kaushik Roy and Mark C. Johnson. "Software design for low power". In: *Low power design in deep submicron electronics*. Kluwer Academic Publishers, 1997. Chap. 6, pp. 433–460. ISBN: 0-7923-4569-X. URL: http://dl.acm.org/citation.cfm?id=265902.

[RJ98] Jeffry T. Russell and Margarida F. Jacome. "Software power estimation and optimization for high performance, 32-bit embedded processors". In: *Proceedings International Conference on Computer Design. VLSI in Computers and Processors (Cat. No.98CB36273)*. IEEE Comput. Soc, 1998, pp. 328–333. ISBN: 0-8186-9099-2. DOI: 10.1109/ICCD.1998.727070. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=727070.

[Rob94] G.D. Robinson. "Why 1149.1 (JTAG) really works". In: *Proceedings of ELECTRO '94*. Vol. 1. 1994. ISBN: 0-7803-2630-X. DOI: 10.1109/ELECTR.1994.472649.

[RPK00] Srinivas K. Raman, Vladimir Pentkovski, and Jagannath Keshava. "Implementing streaming SIMD extensions on the Pentium III processor". In: *IEEE Micro* 20.4 (2000), pp. 47–57. ISSN: 02721732. DOI: 10.1109/40.865866. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=865866.

[Sak88] T Sakurai. "CMOS inverter delay and other formulas using alpha -power law MOS model". In: *Computer-Aided Design, 1988. ICCAD-88. Digest of Technical Papers., IEEE International Conference on*. 1988, pp. 74–77. DOI: 10.1109/ICCAD.1988.122466.

[Sam+02] Mariagiovanna Sami, Donatella. Sciuto, Cristina Silvano, and Vittorio Zaccaria. "An instruction-level energy model for embedded VLIW architectures". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 21.9 (Sept. 2002), pp. 998–1010. ISSN: 0278-0070. DOI: 10.1109/TCAD.2002.801105. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1028101.

[SB13] Yakun Sophia Shao and David Brooks. "Energy characterization and instruction-level energy model of Intel's Xeon Phi processor". In: *International Symposium on Low Power Electronics and Design (ISLPED)*. November. IEEE, Sept. 2013, pp. 389–394. ISBN: 978-1-4799-1235-3. DOI: 10.1109/ISLPED.2013.6629328. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6629328.

[SB77] Herbert Sullivan and T R Bashkow. "A large scale, homogeneous, fully distributed parallel machine, I". In: *Proceedings of the 4th annual symposium on Computer architecture - ISCA '77*. New York, New York, USA: ACM Press, May 1977, pp. 105–117. DOI: 10.1145/800255.810659. URL: http://portal.acm.org/citation.cfm?doid=800255.810659.

[SC00] P.P. Sotiriadis and A. Chandrakasan. "Low power bus coding techniques considering inter-wire capacitances". In: *Proceedings of the IEEE 2000 Custom Integrated Circuits Conference (Cat. No.00CH37044)*. Vdd. 2000, pp. 507–510. ISBN: 0-7803-5809-0. DOI: 10.1109/CICC.2000.852719.

[Sci15] Scikit-Learn. *Scikit-Learn Decision Trees*. 2015. URL: http://scikit-learn.org/stable/modules/tree.html (visited on 03/18/2015).

[SGS10] John E Stone, David Gohara, and Guochun Shi. "OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems." In: *Computing in science & engineering* 12.3 (May 2010), pp. 66–72. ISSN: 1521-9615. DOI: 10.1109/MCSE.2010.69. URL: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=2964860%5C&tool=pmcentrez%5C&rendertype=abstract.

[Shi12] Anand Lal Shimpi. *The AMD FX (Bulldozer) Scheduling Hotfixes Tested*. 2012. URL: http://www.anandtech.com/show/5448/the-bulldozer-scheduling-patch-tested.

[Sim04] Sim-Panalyser. *Sim-Panalyser 2.0 Reference Manual*. 2004, pp. 1–54.

[Smi81] James E Smith. "A study of branch prediction strategies". In: *8th annual symposium on Computer Architecture*. 1981, pp. 135–148. DOI: 10.1.1.219.3575.

[Sni98] Marc Snir. *MPI–the Complete Reference: The MPI core*. Vol. 1. MIT press, 1998.

[Sta12] Stanford University. *CPU DB*. 2012. URL: http://cpudb.stanford.edu/ (visited on 02/17/2015).

[Ste+01a] Stefan Steinke, Markus Knauer, Lars Wehmeyer, and Peter Marwedel. "An accurate and fine grain instruction-level energy model supporting software optimizations". In: *Proc. of PATMOS*. Citeseer, 2001. DOI: 10.1.1.115.3528. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.115.3528.

[Ste+01b] Stefan Steinke, Rüdiger Schwarz, Lars Wehmeyer, Peter Marwedel, Register Pipelining, and Ruediger Schwarz. *Low Power Code Generation for a RISC Processor by Register Pipelining*. Tech. rep. 2001. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.21.8168.

[Sys14] Igor Sysoev. *Nginx*. 2014. URL: http://wiki.nginx.org/Main.

[Taf14] S. Tucker Taft. *Alternatives to C/C++ for system programming in a distributed multicore world*. 2014. URL: http://www.embedded.com/design/programming-languages-and-tools/4428704/Alternatives-to-C-C--for-system-programming-in-a-distributed-multicore-world (visited on 02/24/2014).

[Tex11] Texas Instruments. *Zero-Drift, Bi-Directional Current/Power Monitor with I2C Interface*. Tech. rep. 2011. URL: http://www.ti.com/lit/ds/symlink/ina219.pdf.

[Tiw+96] Vivek Tiwari, Sharad Malik, Andrew Wolfe, and Mike Tien-Chien Lee. "Instruction level power analysis and optimization of software". In: *Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology* 13.2-3 (1996), pp. 223–238. ISSN: 0922-5773. DOI: 10.1007/BF01130407. URL: http://www.springerlink.com/index/10.1007/BF01130407.

[TMW94a] Vivek Tiwari, Sharad Malik, and Andrew Wolfe. "Compilation techniques for low energy: An overview". In: *Low Power Electronics, 1994. Digest of Technical Papers., IEEE Symposium*. IEEE, 1994, pp. 38–39. ISBN: 0780319532. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=573195.

[TMW94b] Vivek Tiwari, Sharad Malik, and Andrew Wolfe. "Power analysis of embedded software: a first step towards software power minimization". In: *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* 2.4 (1994), pp. 437–445. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=335012.

[Tsa07] Dan Tsafrir. "The context-switch overhead inflicted by hardware interrupts (and the enigma of do-nothing loops)". In: *Proceedings of the 2007 workshop on Experimental computer science - ExpCS '07*. June. New York, New York, USA: ACM Press, 2007, p. 4. ISBN: 9781595937513. DOI: 10.1145/1281700.1281704. URL: http://dl.acm.org/citation.cfm?doid=1281700.1281704.

[TT09] Su Lim Tan and Bao Anh Tran Nguyen. "Survey and performance evaluation of real-time operating systems (RTOS) for small microcontrollers". In: *IEEE Micro* (2009), pp. 1–14. ISSN: 0272-1732. DOI: 10.1109/MM.2009.56. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5210078.

[WA12] David Wolpert and Paul Ampadu. *Managing Temperature Effects in Nanoscale Adaptive Systems*. New York, NY: Springer New York, 2012, pp. 15–34. ISBN: 978-1-4614-0747-8. DOI: 10.1007/978-1-4614-0748-5. URL: http://link.springer.com/chapter/10.1007/978-1-4614-0748-5_2%20http://link.springer.com/10.1007/978-1-4614-0748-5.

[Wat09] Douglas Watt. *Programming XC on XMOS Devices*. 2009.

[Wei84] Reinhold P Weicker. "Dhrystone: a synthetic systems programming benchmark". In: *Communications of the ACM* 27.10 (1984), pp. 1013–1030. ISSN: 00010782. DOI: 10.1145/358274.358283. URL: http://portal.acm.org/citation.cfm?id=358283.

[Wel84] Terry A. Welch. "A Technique for High-Performance Data Compression". In: *Computer* 17.6 (June 1984), pp. 8–19. ISSN: 0018-9162. DOI: 10.1109/MC.1984.1659158. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1659158.

[Wir95] Niklaus Wirth. "Plea for lean software". In: *Computer* 28 (1995), pp. 64–68. ISSN: 00189162. DOI: 10.1109/2.348001.

[WJ96] S.J.E. Wilton and N.P. Jouppi. "CACTI: an enhanced cache access and cycle time model". In: *IEEE Journal of Solid-State Circuits* 31.5 (May 1996), pp. 677–688. ISSN: 00189200. DOI: 10.1109/4.509850. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=509850.

[WWA01] G. D. Wilk, R. M. Wallace, and J. M. Anthony. "High-κ gate dielectrics: Current status and materials properties considerations". In: *Journal of Applied Physics* 89.10 (2001), p. 5243. ISSN: 00218979. DOI: 10.1063/1.1361065. URL: http://scitation.aip.org/content/aip/journal/jap/89/10/10.1063/1.1361065.

[XMO10] XMOS. *XMOS Timing Analyzer Whitepaper*. Tech. rep. 2010, pp. 1–9.

[XMO12] XMOS Ltd. *XS1-L02A-QF124 Datasheet*. 2012. URL: https://www.xmos.com/en/published/xs1-l2-124qfn-datasheet?secure=1.

[XMO13a] XMOS. *XN Specification*. 2013. URL: https://www.xmos.com/xn-specification?secure=1.

[XMO13b] XMOS Ltd. *XS1-L Active Energy Conservation*. Tech. rep. 2013.

[XMO14a] XMOS. *USB 2.0 Audio Multichannel U16 Platform*. 2014. URL: http://www.xmos.com/products/reference-designs/multichannel (visited on 04/03/2015).

[XMO14b] XMOS. *XS1-XAU8A-10-FB265 Datasheet*. Tech. rep. 2014.

[XMO15] XMOS. *xCORE General Purpose sliceKIT*. 2015. URL: https://www.xmos.com/support/boards?product=15825%5C&secure=1 (visited on 03/05/2015).

[Yak11] Alexandre Yakovlev. "Energy-modulated computing". In: *2011 Design, Automation & Test in Europe*. IEEE, Mar. 2011, pp. 1–6. ISBN: 978-3-9810801-8-6. DOI: 10.1109/DATE.2011.5763216. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5763216.

[YJ13] Joseph Yiu and Ian Johnson. *Multi-core microcontroller design with Cortex-M processors and CoreSight SoC*. Tech. rep. ARM, 2013.

[Zha+09] Bo Zhai, Sanjay Pant, Leyla Nazhandali, Scott Hanson, Javin Olson, Anna Reeves, Michael Minuth, Ryan Helfand, Todd Austin, Dennis Sylvester, and David Blaauw. "Energy-efficient subthreshold processor design". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 17.8 (2009), pp. 1127–1137. ISSN: 10638210. DOI: 10.1109/TVLSI.2008.2007564.

[ZR13] Yuhao Zhu and Vijay Janapa Reddi. "High-performance and energy-efficient mobile web browsing on big/little systems". In: *Proceedings of the International Symposium on High-Performance Computer Architecture*. 2013, pp. 13–24. ISBN: 9781467355858. DOI: 10.1109/HPCA.2013.6522303.