Power Modelling and Analysis on Heterogeneous Embedded Systems

A Systematic Approach

By

KRIS NIKOV

Department of Electrical and Electronic Engineering
UNIVERSITY OF BRISTOL

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of DOCTOR OF PHILOSOPHY in the Faculty of Engineering.

MARCH 2018

Word count: forty-one thousand five hundred

Abstract

The IoT and mobile industries are predicted to grow exponentially over the next few decades. Battery technologies, which are essential to embedded devices, are developing at a much slower pace. This results in increases in battery area and charge time in order to keep up with rising device energy usage. In order to maintain the practicality of mobile devices and reduce their environmental impact, a great improvement needs to be made to increase the energy efficiency of consumer devices. Increasing demand in the mobile industry for very efficient hardware has seen the rise of more Heterogeneous Systems(HS), which combine multiple specialised Processing Units(PU) on a single Chip. This allows faster and more energy-efficient execution of particular tasks on the optimised PUs. A very good example of this is the ARM *big.LITTLE* System-on-Chip(SoC), which combines a low-power processor with a high-performance one. The system allows for specific tasks to be migrated across the two compute units in a way to maximize performance, while minimizing energy consumption. This research aims to further explore the potential benefits of this configuration in terms of energy management.

The primary aim of this work it to develop a flexible and stable methodology for developing power models on *big.LITTLE* that use the hardware event counters from the on-board Performance Monitoring Unit(PMU). For these purposes several hardware platforms have been used, with the majority of the results being obtained on the HARDKERNEL ODROID-XU3 development board. Several directions for choosing the hardware events are explored, with machine learning proving to be the most successful method. The resulting models are compared against other work and report higher accuracy.

Since *big.LITTLE* is a multi-core system, both single-thread and multi-thread models are explored. The final single-thread models have less than 3% prediction error for both CPU types on the HARDKERNEL ODROID-XU3 board, making them capable of accurately capturing system behaviour and distinguishing between the different energy levels. The multi-thread models, however, have 5% error on the *LITTLE* processor and 9% error on the *big* processor, which makes the *big* model unusable for accurate runtime prediction. This work identifies the key limitations that PMU event based models suffer from on multi-core systems, which are often overlooked. The multi-thread case is explored in much greater detail than any other published work.

The most significant contribution are the fully heterogeneous power models, which use PMU events from one processor type to derive an accurate power prediction of the other. The main purpose is to predict the system behaviour when trying to migrate a task from one processing cluster to the other. The models demonstrate less than 1% error for the single-thread case and less than 3% error for the multi-thread case. These models have much lower error than the individual processor models due to coarse prediction granularity. This work is unique for *big.LITTLE* and truly explores the benefits of the HS for the purposes of energy management.

Dedication and acknowledgements

First and foremost I need to express my endless gratitude to my supervisor Dr Jose Nunez-Yanez. He has been my academic guide for more than 8 years - from the moment I started my MEng in Computer Science and Electronics at University of Bristol up until the end of my PhD. I am certain that without his insight and almost monastic patience and faith in me I would have failed long ago.

Also a big thank you to my industrial supervisor Dr Matthew Horsnell for providing me with an excellent work environment, during my brief visits to ARM in Cambridge. His vast knowledge has helped me immensely during my research. I would also like to thank Dr Kerstin Eder for also being an excellent guide and a constant provider of support.

Thank you to Mr Eric Van Hensbergen for talking to me from across the world and helping me with my first experiments. Thank you as well to Dr Stephan Diestelhorst, Mr Rene De Jong and all the other colleagues at ARM Research for the welcoming and supportive environment.

A special thank you to Dr Samuel Xavier-de-Souza for our collaborative work. It came right at the time where I should have been solely focused on my thesis, but I am glad we managed to carry it out and produce some excellent results.

Big thanks to my friends in Bristol Dr Plamen Proynov and Mr Rosen Rachev for our impromptu lunches and our entertaining conversations. Sharing all our experiences of the academic life really brightened my mood.

My most sincere thanks to my family Polina, Slavka and Kiril. They have been a constant stream of support and I would not have completed this journey without them. I hope I have made them proud.

Finally, I would like to thank EPSRC and ARM Ltd. for funding my PhD and giving a start to my academic career.

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: .....

Table of Contents

List of Tables

List of Figures

TABLE OF ACRONYMS

Acronym	Explanation
ACPI	Advanced Configuration and Power Interface
ASIC	Application Specific Integrated Circuit
BBV	Block Vectors
CCI	Cache Coherent Interconnect
CM	Cache Misses
CMT	Cluster Migration
CPI	Cycles-per-Instruction
CPU	Central Processing Unit
CR	Cache References
DCT	Dynamic Concurrency Throttling
DEM	Dynamic Energy Management
DSP	Digital Signal Processing
DVFS	Dynamic Frequency and Voltage Scaling
ES	Embedded Systems
FLPA	Functional Level Power Analysis
GPU	Graphics Processing Unit
GTS	Global Task Scheduler
HASS	Heterogeneous-Aware Signature Supported
HC	Heterogeneous Computing
HCS	Heterogeneous Computing Systems
HDD	Hard Disk Drive
HES	Heterogeneous Embedded Systems
HMP	Heterogeneous Multi-Processing
HS	Heterogeneous Systems
RMS	Root Mean Square
IKS	In-Kernel Switcher
ILPA	Instructional Level Power Analysis
IPC	Instructions-per-Cycle
ISA	Instruction Set Architecture
MPPA	Massively Parallel Processor Array
OLS	Ordinary Least Squares
PMU	Performance Monitoring Unit
PU	Processing Units
RAM	Random Access Memory
SoC	System-on-Chip

List of Publications

[1] Nikov, Krastin, Jose L. Nunez-Yanez, and Matthew Horsnell. "Evaluation of hybrid run-time power models for the ARM big. LITTLE architecture." Embedded and Ubiquitous Computing (EUC), 2015 IEEE 13th International Conference on. IEEE, 2015.

[2] Xavier-de-Souza, Samuel, Krastin Nikov, Jose L. Nunez-Yanez, and Kerstin Eder. "The energy consumption benefits of DynamIQ for heterogeneous parallel workloads." ARM Research Summit, 2017. [Poster. Work currently being prepared for submission to conference.]

[3] Nikov, Krastin and Jose L. Nunez-Yanez. "Real-Time Power Modelling on Heterogeneous Embedded Systems." [Journal paper. Submitted to ACM Transactions on Architecture and Code Optimization (TACO) December 2017.]

CHAPTER 1

INTRODUCTION

Mobile Computing is arguably the fastest growing consumer technology area in recent history and it is a major part of the semiconductor industry. As a testament, the Ericsson Mobile Report estimates 31.6 billion connected devices by the year 2023 [1]. However, recent effects like the deviation from Moore's law and the emergence of Dark Silicon [2] make it harder for industry to keep up with consumer demand for improved hardware. In this thesis a possible solution to this problem is explored, namely the use of Heterogeneous Systems coupled with power-aware Resource Management. A methodology for developing accurate power models for a Heterogeneous System on a modern development platform has been developed. It is argued that accurate power models are absolutely necessary to develop energy-aware systems. The work presented in this thesis can be extended as the foundation of developing an Energy Management system in the form of an advanced power-aware system software scheduler for Embedded System platforms. This Chapter introduces the topic and key terminology.

Section 1.1 introduces the concept of Heterogeneous Computing in the context of Embedded Systems and gives examples of available configurations. Section 1.2 presents the focus platform of this work, namely the ARM *big.LITTLE* System-on-Chip and describes its importance to the semiconductor industry. The key purpose and objectives of this work along with the principles used to achieve them are described in Section 1.3. The final Section 1.4 outlines the thesis structure and contents.

1.1 Heterogeneous Computing in Embedded Systems

In general Heterogeneous Computing (HC) refers to a platform, which has more than one type of Processing Unit (PU) a.k.a. *cores*. Usually, each PU is optimised for a specific set of

applications. This allows for increased performance at the cost of increased complexity, due to the need to assign the appropriate tasks to the various processing *cores*. Nowadays there are a large amount of different PU. This diversity is dictated by increases in consumer and industry demands for optimisations targeting specific software applications and also advances in technology. Here are a few examples that have been developed over the years and have proven their usability in the industry:

- Central Processing Unit (CPU), the baseline type of PU capable of performing most tasks.
- Graphics Processing Unit (GPU), a specialized PU for computing graphical output and more recently other highly parallels tasks.
- Digital Signal Processing (DSP), another specialized PU for analysing signals (speech, radar, sensor arrays, digital images, communications, etc.).
- Vision Processing Unit (VPU), a more recent specialised PU designed for accelerating artificial neural networks for computer vision and other machine learning related tasks.
- Other specific coprocessors, supervised by the CPU and helping with specific tasks (floating point arithmetic, string processing, encryption, I/O interfacing, etc.).

Examples of various Heterogeneous Computing Systems (HCS) are shown in Figure 1.1. In those examples the CPU is aided by one or many additional Coprocessors, optimised for specific tasks.

HC is applied across all ranges of electronic products, from high performance servers to home desktops to various Embedded Systems (ES). An ES combines both hardware and software in order to perform a specific set of tasks. It lacks the flexibility of a General Purpose computer, but is smaller in size, faster and consumes less power, because it is optimized for a limited amount of tasks. It could be incorporated into bigger systems, or it could be a standalone solution. ES have a wide variety of applications, ranging from consumer electronics, such as mobile phones and cameras to the automotive industry as well as aviation and defence systems. The hardware components of an ES provide the performance capability and are usually packaged in a System-on-Chip (SoC). This means that all components including the main processor, the various co-processors, the memory, peripherals, interconnect, etc. are put on a single chip. This saves area and cost, which is crucial for most ES. The main processing unit can be a dedicated application processor/a micro-controller, a flexible structure of programmable hardware, or both. Using a dedicated piece of hardware to do computation or an Application Specific Integrated Circuit (ASIC), results in longer development time, testing and complexity, however it achieves better performance and less area.

The very competitive mobile industry in particular is trending towards including even more specialized PU on the SoC die in order to satisfy consumer demands for specific content.

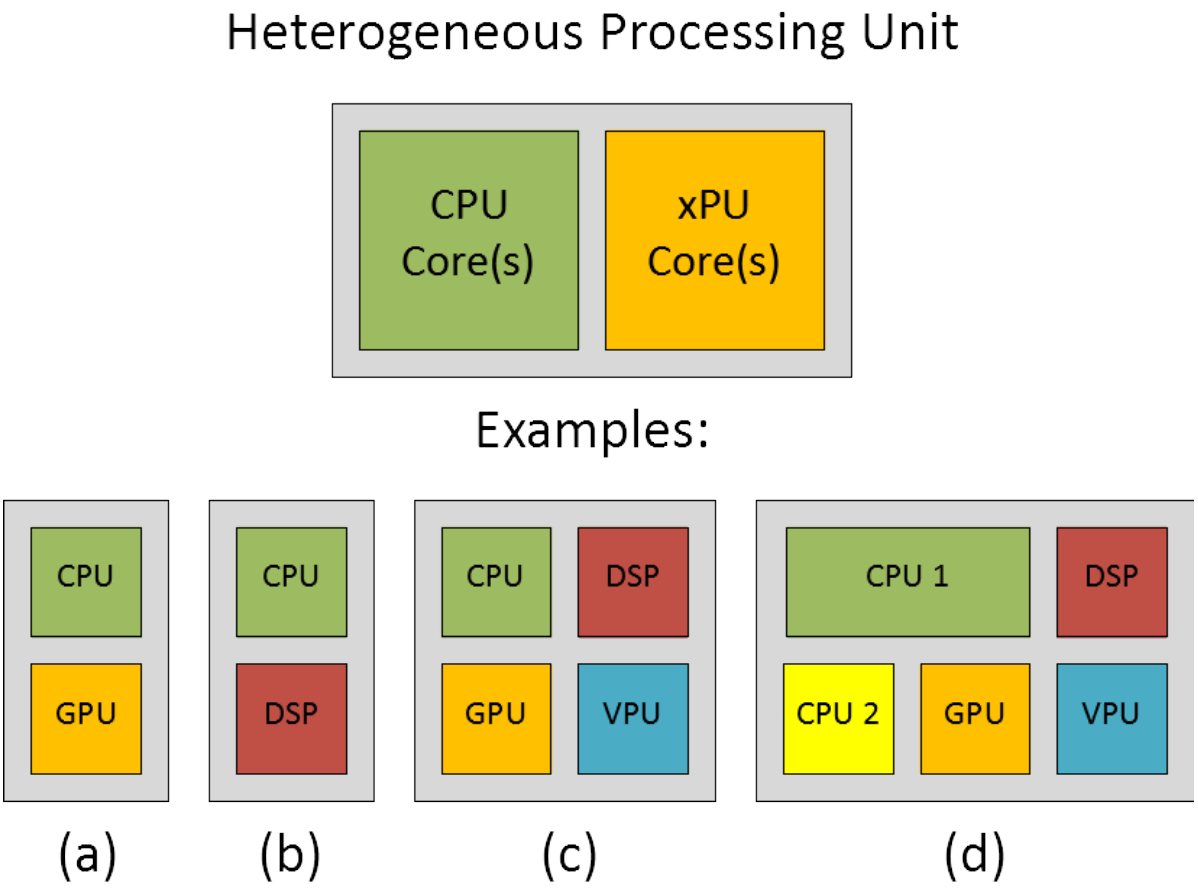


Figure 1.1: **Examples of Heterogeneous Computing Systems:**

a) Variation 1 - System with two Processing Units a CPU and GPU

b) Variation 2 - System with a CPU and a specialised DSP Coprocessor

c) Variation 3 - Modern system with multiple Coprocessors

d) Variation 4 - System with two CPU variants and multiple Coprocessors

There is also a big movement in using existing established co-processors like GPUs and DSPs to assist the CPU in more generic tasks in order to increase performance. For example there are several programming models which allow GPUs to be able to do more generalized parallel arithmetic and not just compute pixels like OpenCL [3] and CUDA [4] [5]. This can be used when the GPU is done with its primary task and the CPU can offload some workloads for it. This General Purpose GPU or GPGPU trend started from trying to increase the performance of high-performance clusters and now it's finding its way into mobile devices due to increased consumer demand for more advanced content and also as a means for different competitors to differentiate themselves.

1.2 The ARM *big.LITTLE* SoC

An example of a commercially successful mobile HC is the *big.LITTLE* SoC [6] developed by ARM. The system was first announced in 2011 [7] and has quickly gained popularity. The system combines a high-performance processor with a power efficient processor in a configurable combination. The two processors use the same Instruction Set Architecture (ISA) so are able to execute the same compiled code [8]. The aim is to achieve better power efficiency by using the heterogeneity of the system to direct tasks towards the processor type they are more optimized for. An example overview of the system is presented in Figure 1.2.

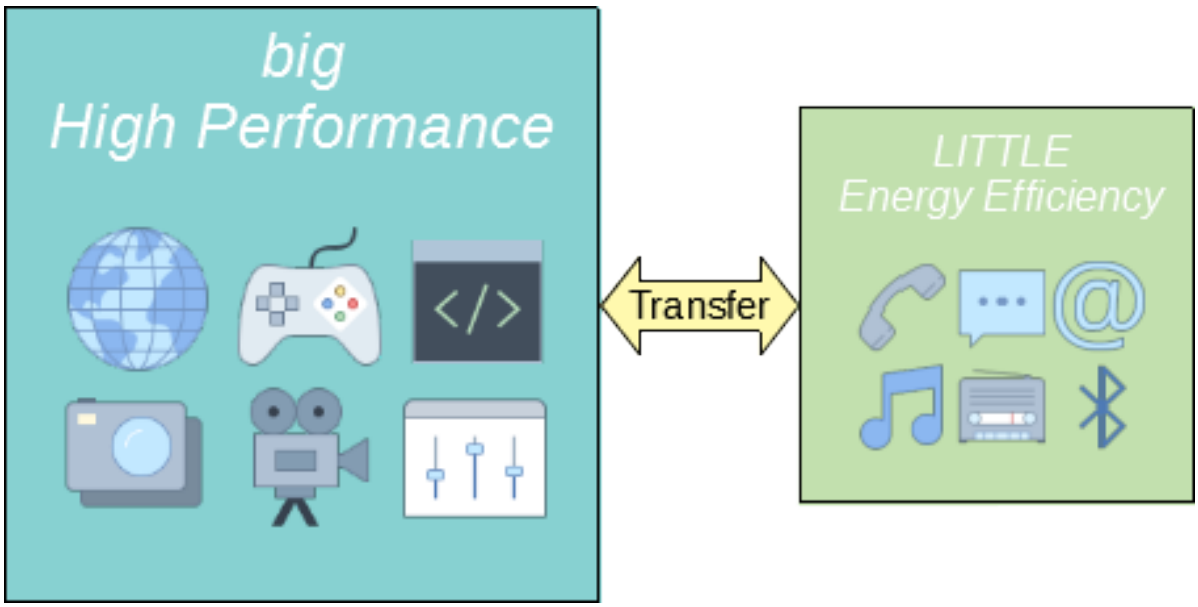


Figure 1.2: **ARM *big.LITTLE* SoC concept.** The *big* processor is more suitable for computationally demanding tasks like web browsing, gaming, video filming, etc. The *LITTLE* processor is optimised for less intensive tasks like calls, sms, email, etc. Tasks are transferred between the two processors dynamically based on user demand.

It has two processor types with binary compatible processing cores, which differ in terms of complexity. The more powerful processor is shown as *big* and the more power-efficient one as *LITTLE*. Examples of the target applications of the two processors is given in Figure 1.2. The *big* is aimed towards high performance demanding applications like web surfing, multimedia and gaming. In contrast the **LITTLE** is aimed towards low energy demand applications, like phone calls, messaging, email and listening to music.

Kumar et al. [9] demonstrate the capabilities of such single ISA systems. They have tested various hardware configurations and scheduling policies and have concluded that Heterogeneous ISA Systems achieve overall an 18%-30% improvement in power efficiency over same area homogeneous architecture depending on how advanced the scheduler is.

Another work that supports the use of Single ISA HC is Borkar et al. [2]. They predict that

Moore's Law will continue, but the power budget will no longer stay the same, resulting in the effects of Dark Silicon. As transistors get smaller, device thermal dissipation will increase up to the point where not all the components on the chip can be powered at the same time, hence the term *Dark* Silicon.w They propose the use of heterogeneous specialized/configurable cores to overcome this issue on a micro-architectural level. They claim that energy will be the key limiter to performance and the aggressive use of accelerators (for small subset of problems) can be key to increasing performance within the limiting power budget.

Hill at al. [10] perform a comprehensive study on extending Amdahl's law [11] in the multi-core era. They explore *symmetric*, *asymmetric* and *dynamic* multi-core designs. *Symmetric* has either all cores parallel or grouped up in bigger units, according to Pollack's rule. The rule states that performance increase is proportional to the square root of the increase in microarchitecture complexity [12]. An interpretation of this rule is that the most performance per area can be obtained by using large quantities of low-complexity processing cores in a Massively Parallel Processor Array (MPPA). In contrast, *Asymmetric* has 1 large combined Core and many small ones in parallel. *Dynamic* assumes the hardware can be configured to have all available simple cores in parallel and be able to dynamically combine them for optimal execution. Their research also makes a compelling argument for the superiority of asymmetric systems compared to symmetric ones in terms of performance increase and energy efficiency.

Woo et al. [13] highlight the applicability of a heterogeneous design with 1 big core and several smaller cores in terms of performance per watt and performance per joule. Some of the assumptions are also based on Pollack's rule as in Hill et al. [10].

It is evident that Heterogeneous ISA Systems like ARM's *big.LITTLE* could provide a way for the mobile market to keep up with consumer demand and can be the way forward for the entire industry. However, due to the increased complexity of such systems and their broader energy usage variation, extra attention needs to be paid to the software side and particularly the Energy Management policies.

1.3 Scope of this Thesis

This thesis is a response to the growing popularity of Heterogeneous Embedded Systems (HES) and the need to address the energy demands of the growing Mobile market. The primary aim is to evaluate if such systems can be used to improve performance and lower the energy usage of consumer products. A large body of related work is examined with the conclusion that in order to fully utilise the heterogeneous hardware, new energy management solutions at the software level need to be developed with increased complexity. The current solutions use CPU utilization data and other performance-measurement techniques to distribute the workload on HS. This thesis proposes that power-aware schedulers can be better for energy management by using power models that closely capture hardware energy usage.

In order to facilitate the development of such solutions, a methodology for generating accurate run-time power models has been researched. The full step-by-step development progress is described. Several platforms implementing the ARM *big.LITTLE* SoC have been used for development, namely the Hardkernel ODROID XU+E [14], the ARM Versatile Express Motherboard [15] [16] with ARM CoreTile TC2 Daughterboard [17] and the Hardkernel ODROID XU3 [18]. The pitfalls and challenges have been closely studied. Factors, such as platform variability and memory system characteristics, that greatly influence system power have been explored and analysed. As the methodology software has matured, the produced models have demonstrated increased accuracy.

The models use system state information from the hardware performance counters in the on-board PMU to dynamically predict power consumption. Several single-thread and multi-thread workloads have been explored and compared, with cBench [19] and PARSEC [20] being used to fit and test the models. The methodology is configured so that the workload can easily be replaced, so that niche applications can also be targeted for power modelling. In addition multiple techniques to choose the PMU events have been investigated. Custom automatic search algorithms, have been developed to identify the best PMU events. This makes the methodology potentially portable to other platforms.

A breakthrough achieved during this research is the modification of the methodology to produce individual models for every CPU frequency level of the development platform. These models are appropriately named *per-frequency* level models in this document. This is done by reducing the amount of information the power model is trying to capture by limiting the model data to the data points at each frequency level independently, thus producing a unique set of model coefficients for every frequency level. This technique results in a much better mathematical fitting over the data and lower model error, compared to a model fitted over the full set of data points across all frequency levels of the development platform. These models are shown to greatly improve accuracy and model system behaviour more closely. This is shown to be a unique insight into power modelling and a contribution to the knowledge in the field in Nikov et al. [21].

At several key points during development the methodology has been used to validate other published work and compare the research against their power models with the produced models achieving significantly lower error that the published models. Model accuracy can vary greatly between different system configurations and workloads used in the experiments as is shown in the results of the comparison. Therefore, this research has focused not only on producing models of highest accuracy, but also a robust, adaptable methodology, capable of generating models tailored to specific scenarios.

The final produced single-thread models demonstrate a 2.5% to 2.9% average error for the *big* and **LITTLE** processors respectively. This is shown to be accurate enough to capture the full behaviour of the target platform, thus these models can be used as the basis of a

power-aware scheduling solution. Extra attention has been paid to the multi-thread case, since the octa-core ODROID-XU3 development platform has a very broad energy consumption spectrum. The multi-thread models exhibit a 9% to 5.7% average error for the *big* and for the **LITTLE** processors respectively and are shown to be unable to capture system behaviour accurately with relation to the minimum accuracy requirement for real-time use. The causes of this are examined and the complexities of the multi-thread case are documented.

As a final contribution this thesis describes the developed full heterogeneous models for the system. The models use PMU events from one processor type to try and predict the average power of the other. This is done using a mathematical scaling technique for the events of the origin processor. They can then be used with the power model equation of the target processor to obtain an estimate of the average power if the workload is executed on it. The reported prediction error is 2.3% to 1.6% when going from **LITTLE** to *big* and *big* to **LITTLE** respectively. Because of the event scaling method, the models can be used to predict between any two frequency levels of the processors. Due to the differences between the two processor types, only PMU events common for both of them can be used in this type of model. Using the automatic search algorithms unique models have been developed for the different scenarios. This technique can only predict average power at slower intervals, but can be used as an accurate basis to form advanced power-aware schedulers. This work, including the finalized single and multi-thread models has been submitted to ACM TACO and is awaiting review.

Finally the resulting work from a collaboration with the Universidade Federal do Rio Grande do Norte Department of Computer Engineering and Automation is described in the thesis. A model capable of capturing the performance and energy usage behaviour of all 8 cores has been developed. The unique characteristic is that the model uses training data only from single-thread data of the **LITTLE** processor and mathematical methods to scale the model for any system configuration. The model also considers the level of workload concurrency and can be used to explore potential energy savings for differing configurations. Validation is done on an ODROID-XU3 development board using the final developed methodology, demonstrating error rates lower than 1.6% and 4.6% for performance and energy consumption, respectively. This work has been presented as a poster at ARM Research Summit 2017 [22].

1.4 Thesis Structure

This thesis continues in 7 other Chapters, organized as follows:

Chapter 2 Background gives an in-depth overview of current Energy Management Solutions for ES, with specific details for the ARM *big.LITTLE* SoC. It also provides a breakdown and comparison of published power modelling methodologies and helps identify the research focus.

Chapter 3 Development Platforms showcases the three systems used to develop the methodology. A particular focus is placed on the HARDKERNEL ODROID-XU3 board, which is used for the majority of the research.

Chapter 4 Methodology describes the software tools developed for generating accurate power models. These include the system configuration and data collection, the experiment workloads, the processing of the experiment data and how the power models are built and validated.

Chapter 5 Single-Thread Models demonstrates the research in developing power models for the single-thread scenario. Starting initially with coarse-grained models for just one frequency for each processor type and extending them to highly accurate run-time power models capable of capturing the entire CPU energy usage with under 3% error. Along the way key stages of the methodology are discussed and unexpected factors that affect system and model stability are investigated, such as the type of flash memory used on the development board.

Chapter 6 Multi-Thread Models continues from the experiments in Chapter 5 and details how the methodology is extended to the multi-thread case. New search algorithms and optimisation criteria are developed in order to improve model accuracy. This chapter demonstrates that the multi-thread case is too complex to be captured with the limited number of PMU events available on the ODROID-XU3.

Chapter 7 Heterogeneous Models details the final part of this research. It involves researching comprehensive models that truly capture the behaviour of the heterogeneous system. Building on some techniques from Chapter 5, power models which can predict average power between the two processor types using PMU event scaling are developed. This chapter also details the recent collaboration with Federal University of Rio Grande do Norte. The collaborative research involved using the developed methodology on the HARDKERNEL ODROID-XU3 platform to fit and validate a different set of mathematical multi-core heterogeneous models for CPU power and performance. At the time of writing this document, this work is still ongoing and the initial results are being prepared for submission to a conference.

Chapter 8 The final chapter gives a brief overview of the research objectives and summarises the most important contributions of this thesis. It also discusses unresolved problems and areas for future exploration.

CHAPTER 2

BACKGROUND

The initial phase of this research involved an in-depth analysis of the current state of the art in energy management and power modelling techniques for computing systems. A thorough understanding of the developed trends was required in order to identify an area that that can be improved upon. The researcher's comprehensive literature review on the topic is presented in this Chapter.

The investigation starts in Section 2.1 by describing the various strategies for Energy Management in Computer Systems that have been developed over the years, with a focus on Embedded Systems and the ARM *big.LITTLE* SoC in particular.

The different methods for power modelling in the industry are summarized Section 2.2, focusing on other CPU hardware-event-based models, including other research directed at the ARM *big.LITTLE* SoC. Details of the power models used in the validation and comparison steps of the methodology are also given in this Section.

The third Section 2.3 gives a more detailed overview of the ARM *big.LITTLE* SoC. The various system configurations and implementations are explored. Details are also given about the two processor types, as well as the available hardware events and counters. Details of the existing energy management scheduler, which migrates tasks between the processor types, are presented.

The final Section 2.4 concludes this Chapter with a short summary, outlying the key research papers from Sections 2.1 and 2.2 and unique characteristics of the ARM *big.LITTLE* SoC from Section 2.3.

2.1 Energy Management

Most electronic systems are designed to deliver peak performance at the lowest energy cost and environmental impact (heat, noise). This is even more prevalent in computing systems. A prime example are smartphones, which need to perform very complex energy-demanding tasks (playing multimedia, web browsing, gaming, etc.) while relying on limited battery power. This requires very efficient design and allocation of computing resources to the software tasks. This is handled by the device EM policy. In the context of this work EM is looked at from the software side and complex hardware designs are not explored. The key principle of energy conservation is that devices are usually not required to execute complex workloads all the time. A successful EM tunes device performance to the workload demand so that energy is not wasted when the device is not in active use. A key part of this process involves being able to accurately estimate workload behaviour and demand in order to scale the performance. Dynamic Energy Management(DEM) policies are able to do this during workload runtime and are present in most consumer oriented devices, which are designed to run a wide range of workloads. There are many existing approaches to DEM with various performances and various levels of adoption by the semiconductor industry [23].

The closes example of a widespread DEM solution for desktop systems is Advanced Configuration and Power Interface (ACPI) [24]. It is a specification which provides an open standard for EM by desktop Operating Systems on a large number of supported system architectures. The latest revision "6.2 Errata A" was released in September 2017 [25] and is currently supported by all major PC components designers and manufacturers and many desktop OS.

ACPI defines *power* states for systems, labelled as **G [0-3]**, and devices, labelled as **D [0-3]**. **G0** and **D0** mean the system/device is fully on. **G3** and **D3** indicate system/device off and states in between turn on/off different system or device components. For example **G1** indicates that the system is sleeping and it itself has several different sleep states, **C [1-4]**, ranging from CPU on but not executing instructions (*idle*) to CPU off and volatile memory Random Access Memory(RAM) saved to non-volatile memory Hard Disk Drive(HDD). These states affect how quickly the system state can be restored on next wake up. ACPI also defines specific processor states **C[0-x]** which enable/disable specific functions of the CPU such as maintaining cache coherency and are dependent on the specific processor (some have up to 10 **C** states). ACPI also supports *performance*, or **P**, states which indicate different levels of scaled voltage/frequency levels for the device essentially throttling the speed of the device but reducing energy consumption and can be accessed when CPU is on - **C0**, or for the device when it is on - **D0**. These performance throttling states are in broader terms a type of Dynamic Frequency and Voltage Scaling(DVFS) which scales the power rail voltage and clock frequency of the processor in order to reduce its power consumption. In the context of ACPI, the DVFS in P states is done by the OS, when it can determine that the processor can be slowed down

and still perform the workload in time, for example if applications become memory bound the OS can enter the CPU in a higher (less energy demanding) **P** state, during a more memory intensive execution phase that does not require many computations, and then bring it back to full speed, thus saving energy. Recently more and more GPUs are starting to have extensive **P** state support as well.

ACPI may be a good example of a widely adopted EM specification, but ES are not as complex as desktop computers and consume much less power in general to warrant many different system states. Currently these is no universal EM approach available for ES with may different strategies being used such as Power-Aware Software Compilation, Low Power Bus Encoding, Low Power System Synthesis, Dynamic Power Management and DVFS [26]. It is becoming ever more critical to develop better EM solutions for the current and future generations of ES. Figure 2.1 depicts current trends in Power Dissipation for Mobile Embedded Systems based on data from Samsung in 2011[27][28].

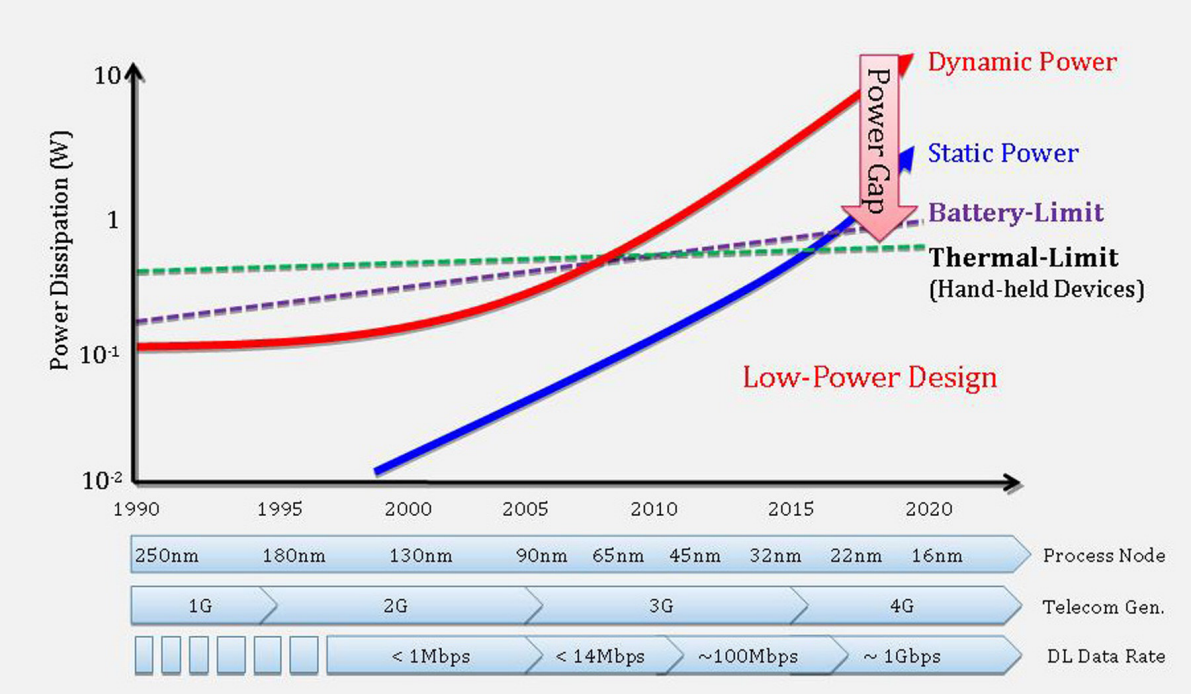


Figure 2.1: **Power Dissipation trends for Mobile Embedded Systems.** Data from Samsung, the world leading smartphone manufacturer, shows that battery technologies cannot keep up with projected system power consumption caused by increasing consumer performance demands [27][28].

The presented trend seems quite alarming, since it shows that battery technologies are not being able to keep up with the demand for more power, so even more physically bigger batteries will be needed to power mobile devices in the future. Not only that, but also existing EM methodologies cannot keep up with industry demand. HES like *big.LITTLE* provide the

necessary hardware platform for advances in low-power design, however Esmaeilzadeh et al. [29] and Shafique et al. [30] argue that further advances in Power Management are still needed to keep up with Moore's Law.

Energy Management is becoming increasingly complex as well, because power consumption in modern chips is becoming harder and harder to define mathematically. Traditionally the total chip power on a MOSFET system has been defined as:

$$P_{TOTALL} = P_{SYSTEM} + P_{LEAKAGE} + P_{DYNAMIC} \tag{2.1}$$

In this equation, $P_{LEAKAGE} + P_{DYNAMIC}$ is the power consumed by the CPU and P_{SYSTEM} is the power consumed by the rest of the system. $P_{DYNAMIC}$ is the power consumed by the CPU during the switching activities of transistors during computation. $P_{LEAKAGE}$ is power originating from leakage effects inherent to silicon-based transistors and is dependent on manufacturing technology and not the CPU workload.

$P_{DYNAMIC}$ can be expressed in terms of CPU chip per clock switching capacitance as C, CPU voltage as V, Scaling activity vector as A and CPU frequency as f as the following equation:

$$P_{DYNAMIC} = C \times V^2 \times A \times f \tag{2.2}$$

Naturally in the past, when $P_{DYNAMIC}$ constituted the much larger part of the CPU power consumption, one could just reduce the CPU frequency and voltage and expect reduction in energy consumption due to the quadratic relationship between voltage and power. $P_{LEAKAGE}$ is also shown to be related to operating voltage, so reducing CPU voltage can greatly reduce the overall power of the entire system. This is why DVFS strategies are quite effective for EM.

As components become smaller, static power increases and is becoming a more dominant part of the power consumption of the device. De Vogeleer et al. [31] highlight the convex relationship between energy and CPU frequency on a modern mobile device, indicating that the lowest voltage/frequency CPU operating point is not the most energy-efficient.

Due to these reasons it is argued that any successful EM solutions for ES need to be able to capture this complex relationship between energy and frequency. One feasible way of doing this is to use accurate CPU power models in order to determine the most optimal configuration. Following are a few examples of EM solutions utilising different forms of power estimators.

A notable example of an Heterogeneous Multi-Processing (HMP) scheduler targeting HES is Thread Motion (TM) by Rangan et al. [32]. They propose a scheme to dynamically analyse and migrate threads between processing cores with different, but fixed voltage/frequency levels. TM uses CPU Cycles-per-Instruction(CPI) prediction to assign program threads to faster cores and calculates migration cost to adjust for overhead and ensure energy savings. It is tested on SPEC 2006 and achieves around 20% improvement compared to a static DVFS scheme for the same power budget, with only 1% power overhead. This scheduler is designed to run on a dedicated microcontroller, however it can also be incorporated into the operating software.

Choi et al. [33] in contrast, propose a software DVFS algorithm that separates activities into CPU or Memory intensive. Their approach uses the fact that the CPU clock can be reduced when RAM intensive activities are performed and vice versa. Their algorithm monitors dynamic events at runtime to and does not need compiler support or modification to the program. It uses a regression model on PMU events as well as instruction information to determine CPU idle time due to memory stalls by estimating the ratio of off-chip to on-chip computation time. The scheduler is tested with various Linux utility programs and they report 70% CPU energy saving with 12% performance degradation for memory intensive programs and 15% to 60% CPU energy savings with 5% to 20% performance degradation for CPU intensive programs.

Shelepov et al. [34] propose Heterogeneous-Aware Signature Supported (HASS) scheduler. They use an Intel framework called PIN to estimate cache miss rates for different CPU configurations (the different domains of the heterogeneous system) and this constitutes the code signature and instrument the code with this information. HASS uses this information to make decisions on migrating tasks. It is evaluated on several platforms featuring modern Intel and AMD CPUs running SPEC 2000 [35] with a reported 15% speedup compared to an on-line IPC-based phase-aware scheduler and it did perform better.

Curtis-Maury et al. [36] present a very interesting approach for allocating resources for HPC systems, which shows how IPC-driven schedulers like [34] can be used in an Energy Management context. The scheduler utilized both DVFS and Dynamic Concurrency Throttling (DCT) to determine the fastest configuration on run-time. The scheduler uses compiler support in the form of an extension to the Intel OpenMP [37] compiler with added mechanistic-empirical prediction models, which use PMU events, for estimating effective IPC (effective IPC is IPC without parallelisation and synchronization instructions). The models are trained on the NAS Parallel Benchmarks [38] and the development platform features an Intel Xeon E5320. The model information is then instrumented into the code at compile time and used by the scheduler. The scheduler achieves a performance increase of 14%, power savings of 6% and energy savings of 19% compared to running all cores on maximum frequency. This clearly shows that optimizing performance can reduce energy usage.

Of particular interest is also the work of Imes et al. [39]. They investigate heuristic power-optimization scheduling techniques for an Intel and a *big.LITTLE* platform. The PARSEC benchmarks [40] are used to evaluate *race-to-idle* and *never-idle* techniques for optimizing power consumption. Results indicate that on the Intel platform *race-to-idle* is usually better and on the *big.LITTLE* platform *never-idle* saves the most amount of energy. This goes to show that existing solutions for Intel or AMD CPUs cannot be blindly migrated for ARM *big.LITTLE* CPUs and further research in the field is required dedicated for *big.LITTLE* HMP.

Craeynest et al. [41] also investigate scheduling solutions for *big.LITTLE*. They have developed PIE, an experimental scheduler that focuses on determining Memory and Instruction

Level Parallelism (MLP and ILP) using mathematical derivations from PMU activities to determine which core to run on. The migration decision is based on performance thresholds, which translate to power savings. The scheduler needs specific hardware support to make it work, which makes it inapplicable in existing systems, however achieves around 5% to 8% improvement in performance compared to existing system schedulers on SPEC 2006 [42].

2.2 Power Modelling

The ability to model system and CPU power has been investigated ever since the very birth of the semiconductor industry. As electronic systems increase in complexity system power can no longer be derived mathematically. Researchers have tackled this problem by using Supervised Machine Learning to derive accurate power models. SML algorithms operate by automatically mapping sample inputs to outputs. In the case for deriving power consumption, the system/component of interest is driven by a controlled workload input and power consumption as well as selected internal characteristics of the components are collected. The ML algorithm then is used to derive a power model by mapping the internal characteristics from all intervals to the power consumption. The derived mapping is then verified against another input set, not used in training, since the aim is to predict power consumption for any future workloads. In this section some of the published methodologies for developing power models are summarized and contrasted to the developed approach.

An example of using Machine Learning to derive an equation for power consumption is the work of Hsieh et al. [43]. They use Neural Networks to compute the power model for small metal-oxide-semiconductor circuits achieving 4.1% accuracy against a test set. In their work they monitor physical changes in the circuit and use those as predictors. However in modern ES it is not feasible to use this approach because of the increased complexity and reduced size of the circuit. Modern power models use high-level events to derive power, since they are much easier to observe.

Takouna et al. [44] present a very minimal linear power model for the Intel Xeon E5540 CPU, which uses frequency and number of cores to predict power with an average of 7% error. A limitation to this approach is that it cannot predict the changes in power consumption at a particular frequency level, as it will only model the average power for that frequency. This is illustrated in the model comparison in Chapter 5 Subsection 5.3 where the generated single-thread models are compared with other published work, including Takouna et al.

The need for fine-granularity models is also shown by Sherwood et al. [45] who use instructions grouped in Basic Block Vectors (BBV) to identify different phases in program execution and characterize their behaviour.

Another successful way to observe finer changes in program execution is using hardware system information available from the Performance Monitoring Units on a CPU. Historically

PMUs have been used to estimate performance, however research has shown that CPU/system power estimation is also possible.

The work by Isci et al. [46] and Bertran et al. [47] use PMU events to compute accurate fine-grained power models. Isci compare BBVs computed from instruction blocks or PMU events on a Pentium 4 CPU and concludes that Overall PMU phases are up to 33% more accurate when tested on SPEC CPU2000.

Bertran et al. [47] present an empirical 2-level Functional Level Power Analysis (FLPA) model for an Intel Core2 Duo CPU. It splits the architecture into functional blocks (Front End; Integer; Floating Point; Branch Prediction; L1 Cache; L2 Cache; Front Side Bus) and then develops power models individually using tailored micro benchmarks and specific PMU events for every block. The system-level power model is then computed as an Ordinary Least Squares (OLS) liner regression [48] on the block power models and is able to successfully predict program phases within 83.91% success when tested on SPEC CPU2006. Tested against more basic models showing that even though they can achieve similarly low error percentage they are pretty bad at following program phases unlike the FLPA model.

In contrast to this is the work of Nunez-Yanez et al. [49], which makes a strong case that system-level modelling is better that component level modelling, They use a large number of PMU events collected with a simulator on an ARM Cortex-A9, to train a linear model using linear regression. Instead of using micro benchmarks they use cBench [19] as a workload stress the entire system as a whole and report an average of 5% estimation error.

Pricopi et al. [50] develop complex models for predicting performance by predicting the CPI stack. As part of their work, however, they have also built a mechanistic model for the ARM Cortex-A15, which utilises CPU design experience and a deeper understanding of the architecture to select the list of PMU events used. They train the model with an average error of 2.6% when trained and tested on SPEC CPU2000 [35] and SPEC CPU2006 [42] benchmark suites. They have not produced a model for the ARM Cortex-A7 on the justification that the processor does not exhibit much variation in its power dissipation and can be approximated by a single number. In this research this assumption is refuted and it is demonstrated that the ARM Cortex-A7 also exhibits significant power variation and dedicated power models are required to capture its behaviour. Their work is done on an experimental platform, the ARM Versatile Express Motherboard [15] with CoreTile TC2 Daughterboard [17], and on a single CPU frequency, hence the simplified power profile. Nevertheless this is one of the earliest PMU based models available for the ARMv7 architecture and provides great insight into the use of PMU events for power modelling. Their research platform is also featured in this research with information given in Chapter 4, Section 3.2. Their power model is also used in the model comparison and validation for the developed single and multi-thread models for the ARM Cortex-A15 in Chapters 5 and 6.

Similarly, Singh et al. [51] have developed a power model based on 4 PMU events on AMD

Phenom 9500 CPU. They use micro benchmarks to train the model and events are collected every second. The model is computed using piece-wise linear regression with least squares estimator and is tested on NAS, SPEC-OMP, and SPEC 2006 with median errors of 5.8%, 3.9%, and 7.2% respectively. They further this work by using the model to guide a single-thread scheduler, which suspends processes to ensure a power budget. This shows how power models can be used effectively in dynamic schedulers ho help improve the power efficiency of systems.

Rodrigues et al. [52] develop a model, designed to offer the most accuracy with a minimal set of events for both a *high-performance* and a *low-power* execution unit, represented by an unnamed Intel Nehalem and Atom processors in a simulation environment. Their research is also very interesting, because the authors have performed a comprehensive analysis of PMU events by comparing several models, utilising different numbers of predictors. The models are trained and validated on an extensive suit of benchmarks, consisting of SPEC 2000 [35], MiBench [53] and mediabench [54]. The final reported model error is less than 5% for both CPU types for a single-core set-up. Their models were attempted for translation to the ARM *big.LITTLE* SoC platforms in order to be validated in this research, but unfortunately they used some particular pipeline stall events unavailable in the ARM PMU.

Some researchers have successfully combined physical information about the platform, such as frequency and mechanistic information like pipeline depth with PMU events to achieve complex high-accuracy models. Blume et al. [55] present a very thorough research on different model generation methodologies. Models were built for a stand-alone ARM 940T CPU as well as a complex OMAP5912 system (ARM926EJ-S and C55x DSP cores) and tested on a wide-ranging workload. A complex Functional and Instructional Level Power Analysis (ILPA) approach is compared to a simpler FPLA one and just a frequency based model highlighting that model performance increases with its complexity. The FLPA/ILPA model achieves up to 9% accuracy on the standalone ARM core and 4% on the OMAP system, highlighting that complex systems hide some of the dynamic power usage.

Rethinagiri et al. [56] present another work incorporating physical platform information and PMU events to predict power consumption. They have developed a power-estimation tool for embedded systems, incorporating physical platform information and PMU events to predict power consumption, tested for ARM9, ARM Cortex-A8 and ARM Cortex-A9 CPUs. They base their approach around accurate run-time system-level power models and use micro benchmarks to obtain cache information and *intuitively* selected PMU events and train a linear model using Ordinary Least Squares linear regression [57] . Their model has a small set of regressands, since they use just CPU frequency and 4 PMU events. Despite this they report around 4% for all three CPUs on a custom microbenchmark test set. The interesting thing about this model is the heavy emphasis on cache events. This model is also used for comparison in this research in Chapters 5 and 6. It is demonstrated that this model performs poorly, precisely due to the high variability of cache events in complex workloads. This research shows that

analysing the events with high variation and removing them from the event selection process results in a much more stable and accurate models.

Eyerman et al. [58] also build mechanistic-empirical models to predict CPU cycles on a wide range of Intel processors. The interesting aspect of their research is their robust methodology which they use to test the accuracy of purely empirical models (in which all estimation events are dynamic) compared to mechanistic-empirical ones (with added information about system components, like cache size, pipeline depth, etc.) . Their results using SPEC CPU2000 and SPEC CPU2006 show that purely empirical models are quite prone to overfitting the data at the benefit of reduced complexity.

Jacobson et al. [59] have investigating how the number of events affects the model performance. They use a cycle-accurate simulator to run various workloads including SPEC 2006 and other commercial applications on a POWER7 CPU. 10% of the workload is used to train the model and the remaining 90% is used to test. They collect 2300 different attributes and after a thorough correlation analysis a final 54 attributes grouped in 35 pairs were chosen for a scalable model. Their research shows that 8 attributes is enough for accurate (5%) power prediction, but the additional events are added for better model scalability for future technology trends (which is validated against a scaled reference platform). Another interesting aspect of this work is the comparison between the mathematical derivations of the top 8 events with an expert-driven approach, which showed that the intuitive approach does not always give good results.

Another approach to predicting power is using information about the CPU state and utilization. For example, Zhang et al. [60] examine the performance of the Chrome web browser on multi-core smartphone platforms. Their work shows that existing models based only on frequency and CPU utilization are not accurate for multi-core systems. They propose a model using also weighted average duration spend in idle states and report 96% accuracy on their own constructed set of benchmarks. However their model uses a predictor for ACPI states to estimate CPU idle time, which is very difficult to do in real-time so it only works offline.

A similar approach is used by Walker et al. [61]. They present two different methodologies for 2 development platforms. They develop a model using 4 PMU events for a system featuring the ARM Cortex-A8. With that approach they report 1.9% average error while predicting power consumption using MiBench [53] as a workload, which is the predecessor to cBench. They also present a CPU frequency and utilization based model for a *big.LITTLE* platform, which did not have support for PMU. Contrary to Zhang they obtain information about CPU time spent in idle using information available from the Linux kernel running on the device. Tested on the same workload as the PMU model, the CPU frequency and idle time model achieves 10.4% and 8.5% error for the ARM Cortex-A7 and ARM Cortex-A15 respectively. This model is used in Chapter 5, Subsection 5.3 for comparison against the generated models. Those results

show this model having a significantly higher error than reported. The same researchers have continued their work [62] and have developed a model for *big.LITLE* on the HARDKERNEL ODROID XU3 [18] development board. That is also the main platform used in this research and its characteristics are presented in Chapter 3, Section 3.3. Their updated methodology uses the SPEC 2006 [42] workload and a simple SML method to traverse the list of available PMU events. They have developed individual models for the ARM Cortex-A15 and ARM Cortex-A7, though only the events list for the former is published. Their model is used in the model comparison and validation experiments in Chapters 5 and 6. They have done a very thorough research into the statistical and mathematical drawbacks of regression-based models had have presented a method for increasing model accuracy and flexibility by addressing the problem of heteroskedasticity in power modelling and report 2.8% and 3.8% average error for the ARM Cortex-A15 and ARM Cortex-A7. This work proposes a more effective strategy to ensure model accuracy and stability combined with an extended analysis of different model event selection search algorithms. These are described in Chapter 4, Section 4.3.

2.3 big.LITTLE in Detail

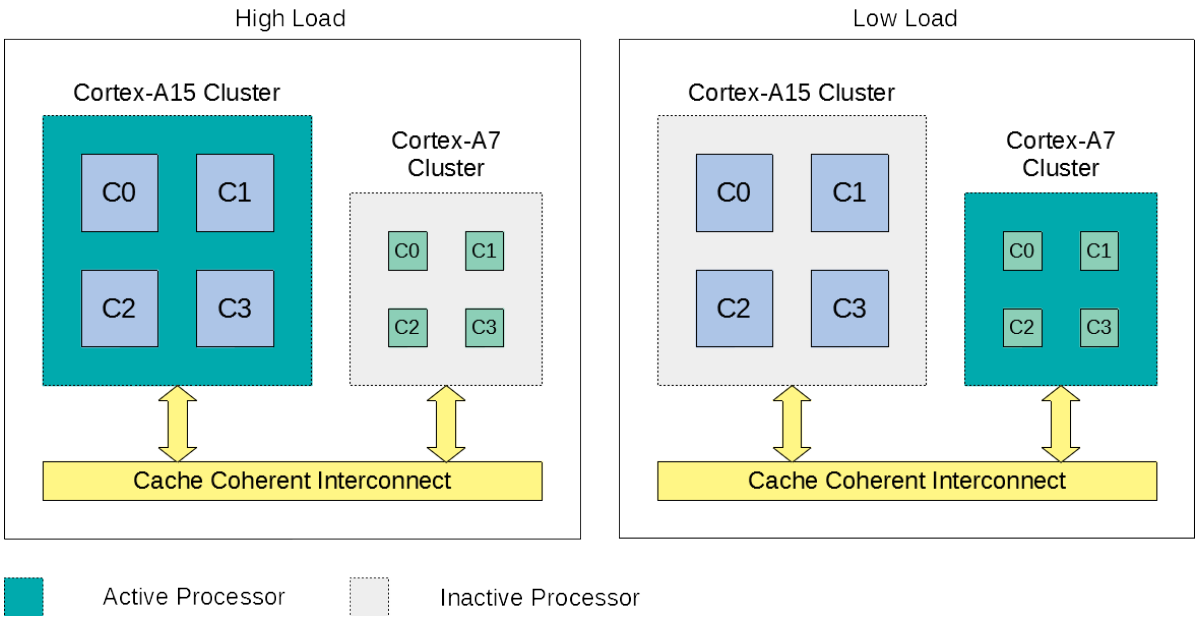


Figure 2.2: **Overview of the Samsung Exynos 5410 *big.LITTLE* SoC.** The Samsung Exynos 5410 SoC [63] is the first commercial *big.LITTLE* implementation, released in 2011 [64]. The CPU consists of 4 ARM Cortex-A15 and 4 ARM Cortex-A7 processing cores, grouped in two clusters. The system could only use one cluster at a time, due to limitations in the scheduler.

In the previous Chapter 1 the ARM *big.LITTLE* SoC and some of the research validating the benefits of such Heterogeneous Single-ISA Systems were briefly introduced. The methodology

is aimed at *big.LITTLE*, because it is believed that utilising this hardware will help overcome some of the foreseeable problems for the IoT and mobile industry. The best thing about *big.LITTLE* is the software support and the availability of development platforms on the market. 3 different platforms were used in this research and managed to find great software support for all of them. This is detailed in Chapter 3. There are plenty of resources available on [6] including a few white papers that detail the capabilities of the system [8] [65]. The key feature of the Soc is the Cache Coherent Interconnect (CCI) [66] [67], which enables quick task migration between the two CPU islands. The CCI interface is built on the AMBA bridge specification [68], making it really configurable. This allows for great power savings by utilising the *LITTLE* core while maintaining good performance levels. More details on how task migration work are available in section 2.3.4.

The fist iteration of *big.LITTLE* came out in 2011 [64] [7] and one of the first industry adopters of big.LITTE is Samsung [69] who use it as part of their Exynos SoCs. The first one of them to become industry wide is the Samsung Exynos 5410 SoC [63]. It is an 8 core system with 4 ARM Cortex-A15 and 4 ARM Cortex-A7 and it is commercially realised as the CPU of the Samsung Galaxy S 5 [70]. An example of the SoC is given in Figure 2.2.

Since then ARM has moved on to migrate *big.LITTLE* to their ARMv8 64-bit ISA and implement it with the more advanced Cortex-A53 and Cortex-A57 CPUs [71] [72] [73]. However this work is done on the first generation of the *big.LITTLE* platform so the models are build for the Cortex-A15 and Cortex-A7.

2.3.1 ARM Cortex-A15 CPU

The ARM Cortex-A15 CPU [74] [75] [76] is a high-performance 32-bit multi-core processor implementing the ARMv7-A architecture. Each processor cluster can have up to 4 Cores, which share an L2 cache. Each core has an individual L1 Instruction and Data cache. The processor is capable of addressing up to 1 TB of memory. The key feature is its triple-issue out-of-order 15 stage pipeline which achieves a significant 50% improvement over its predecessor the Cortex-A9, which is another highly used CPU in embedded systems. All these features mean that the CPU is targeted heavily towards the more high-performance segment.

2.3.2 ARM Cortex-A7 CPU

The ARM Cortex-A7 CPU [77] [78], on the other hand is an extremely energy-efficient 32-bit multi-core processor. It is also implementing the ARMv7-A ISA and both Cortex-A15 and Cortex-A7 are binary compatible. The Cortex-A7 can also be configured to have up to 4 Cores per CPU cluster with an optional integrated L2 cache subsystem. It has the same interconnect capability as the Cortex-A15, but its distinctive feature is its 8-stage in-order partial dual-issue pipeline. This design is and update from the Cortex-A5 and boast a 20% increased single-thread performance, while maintaining energy efficiency levels.

2.3.3 The Performance Monitoring Unit (PMU)

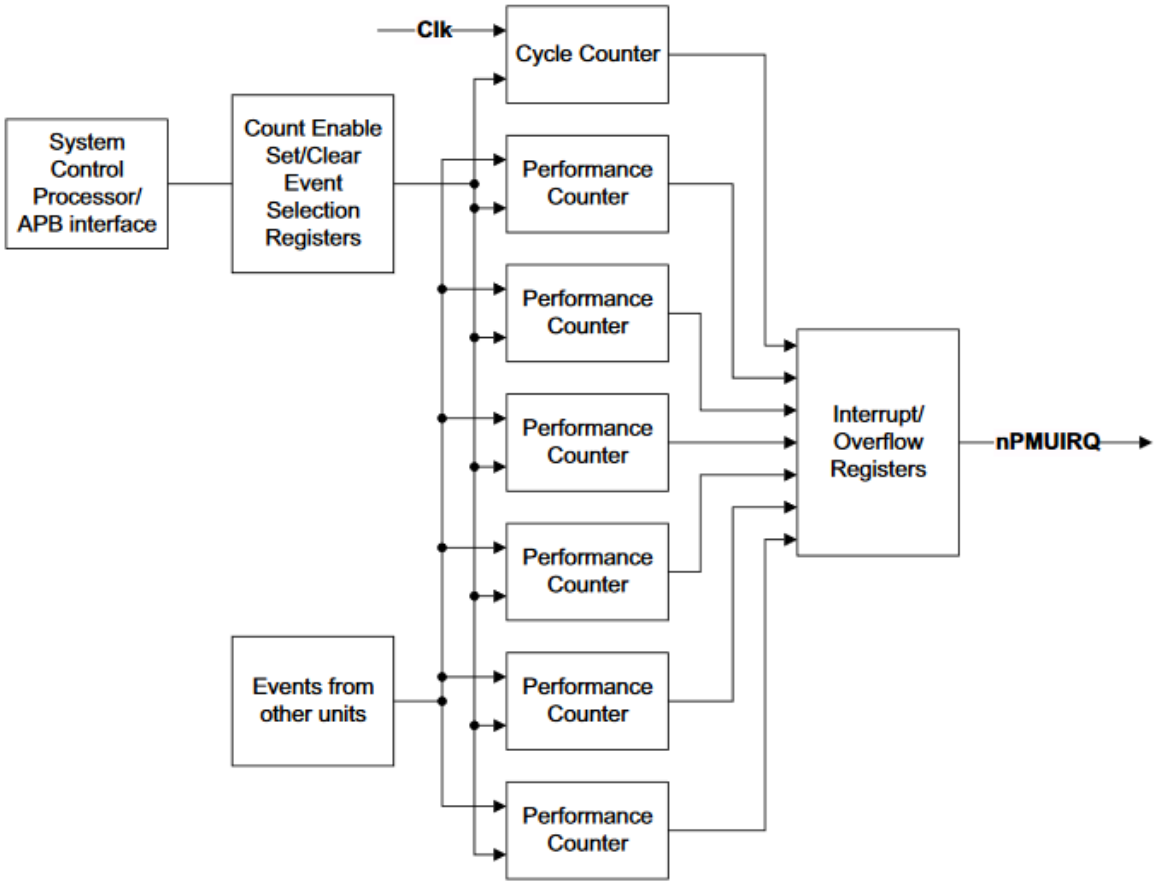


Figure 2.3: **Overview of the ARM Cortex-A15 PMU.** It has 6 programmable 32-bit performance counters, with an additional dedicated counter for CPU cycles [75]. The counter values are saved in special registers in the event of a system interrupt and/or overflow in order to provide accurate, repeatable readings.

Hardware performance counters are special registers available in most modern day CPUs, that allow tracking of system events on the hardware level. Example of such events are Cache hits or refills, CPU cycles, Memory access, etc. These counters are typically configurable and are able to collect from a large number of defined events. Since the number of counters is limited, some implementations use multiplexing to switch between the events they are monitoring. These counters are a great tool for low-level system monitoring and analysis and are used in various applications. The main advantage of using hardware counters rather than software tools is the low overhead of collection. In this research they were used them to try and capture system behaviour for the purposes of predicting CPU power.

In ARMv7 the collection of hardware counters is called the Performance Monitoring Unit (PMU). An overview of the PMU available in the Cortex-A15 is given in Figure 2.3. It provides

six configurable counters with an additional seventh reserved just for CPU cycles [75]. In contrast the PMU available for the Cortex-A7 only has four configurable counters, but still has a dedicated one for CPU cycles [78]. A total of 67 different hardware events are available for collection for the Cortex-A15 and 42 for the Cortex-A7. Interestingly only 23 are common to both PMUs. A full list of the PMU events is available in the Appendix A.

In order to communicate with the PMU without a debugger and configure the performance counters, the linux tool ***perf*** [79] was used. ***perf*** allows communication with the linux kernel from userspace and can profile the entire system. The PMU events for collection can be specified using the RAW identifiers shown in A.

2.3.4 Existing Energy Management Solutions for the *big.LITTLE* Platform

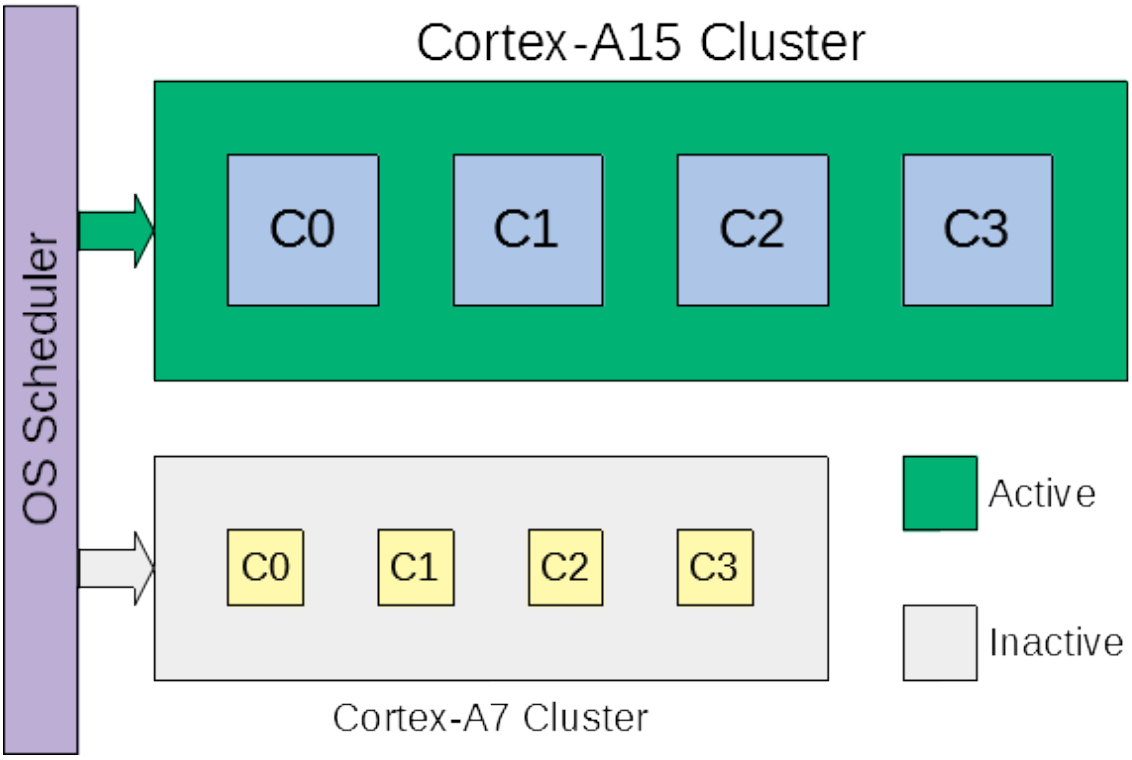


Figure 2.4: **Overview of the Cluster Migration Scheduling Policy for the ARM *big.LITTLE* SoC.** The *big* and *LITTLE* processing cores are grouped in two clusters [80]. The OS can execute tasks on only one of them at a time. When switching operating mode all tasks must be moved to the other cluster.

In the context of Heterogeneous Embedded Systems in particular the *big.LITTLE* SoC, the Energy Management system has an additional level of complexity. For that purpose ARM have developed patches for the Linux and Android Operating Systems, which support a custom scheduler for *big.LITTLE* [80]. The scheduler is a natural extension of DVFS, which allows tasks to be migrated from one CPU cluster to the other. Thanks to the Cache Coherent

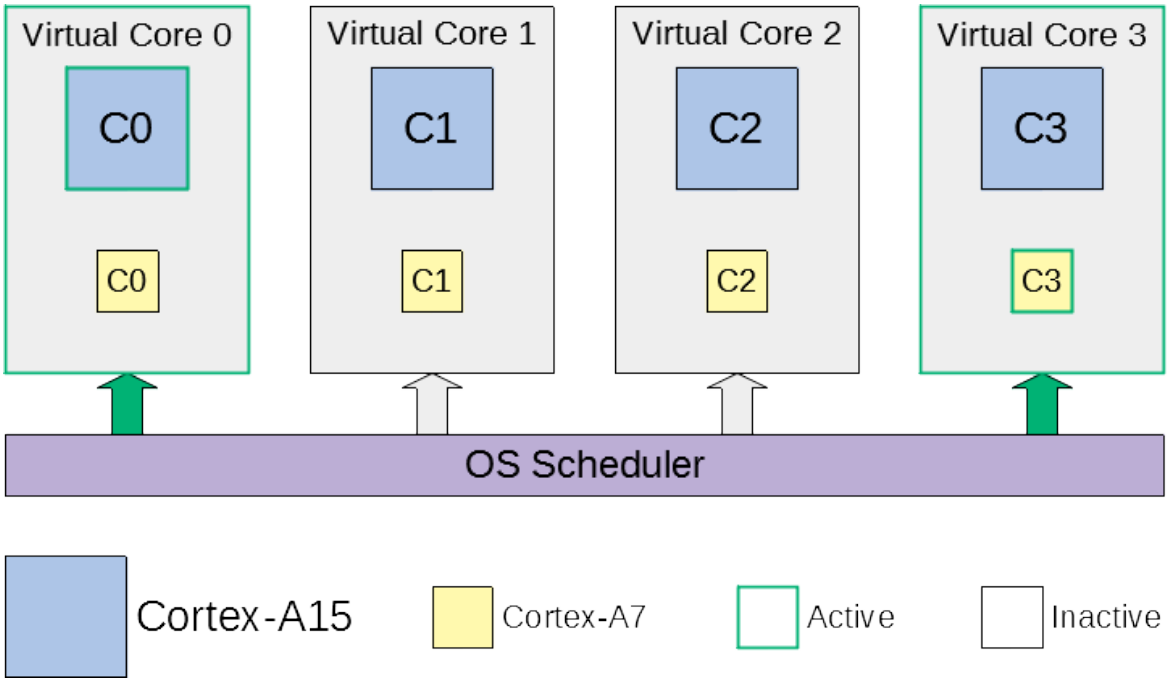


Figure 2.5: **Overview of the In-Kernel Switcher Scheduling Policy for the ARM *big.LITTLE* SoC.** The *big* and *LITTLE* processing cores are grouped in Virtual Cores, each giving access to one physical core of each type [81]. Each Virtual Core can have only one physical core active at a time (either the *big* or the *LITTLE*), controlled by the scheduling policy. Task migration happens only within the Virtual Core, so the system can have at most 4 processing cores running at a time.

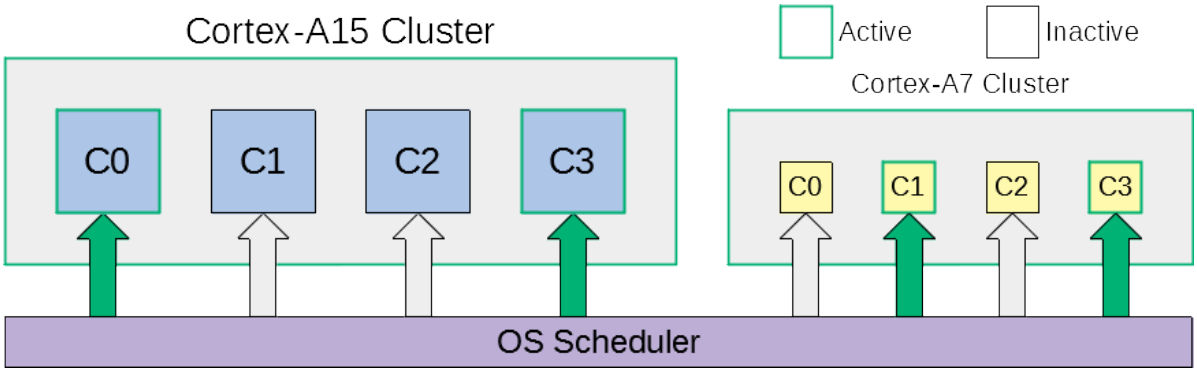


Figure 2.6: **Overview of the Global Task Scheduling Policy for the ARM *big.LITTLE* SoC.** The OS can schedule tasks to each *big* and *LITTLE* processing core independently, unlocking the full capabilities of the SoC [82]. Task migration can be performed between any two compute cores, regardless of their physical cluster association.

Interconnect the overhead of migrating the task is kept low. The scheduler has 3 operating modes, depending on the particular implementation – Cluster Migration (CMT), In-Kernel Switcher (IKS) and Global Task Scheduler(GTS). Most of the documentation for the *big.LITTLE*

scheduler comes from ARM in the form of white papers [8] [81] [65] [82].

2.3.4.1 Cluster Migration

Cluster Migration, presented in Figure 2.4, is the first developed thread migration solution. The OS scheduler can only select one of the clusters to execute all tasks. This means that only 4 cores can be utilised at the same time. This type of scheduler is available on the Samsung Exynos 5410, which is used by the ODROID XU+E.

2.3.4.2 In-Kernel Switcher

The In-Kernel Switcher, presented in Figure 2.5, is the second scheduling solution developed by ARM. Instead of migrating all the tasks between the clusters it pairs a *big* and a *LITTLE* core as one virtual core. Which core is being utilised depends on the performance demand, with the other core being shut off. This still limits the number of cores that can be used by the system, but it allows more flexibility and *big* and *LITTLE* cores that can be on at the same time, as long as they are not part of the same virtual core. Pourier [81] gives more details about the IKS.

2.3.4.3 Global Task Scheduler

The final scheduler developed by ARM is the Global Task Scheduler, presented in Figure 2.6, which allows migration between any two CPU cores, even the same type. This also enables the full capabilities of the system with the ability to use all cores at the same time. Task allocation priority is given to the *big* cores. This implementation is available for the latest *big.LITTLE* SoCs including the Samsung Exynos 5422. The GTS is available for one of the development platforms - the ODROID XU3. More information about the GTS can be found here [65] [82]. [83] from Samsung describes their adoption of the GTS and comparing it to the initial CMT scheduler.

2.3.4.4 Thread Migration Criteria

All the aforementioned schedulers rely on migration thresholds to decide when it is time to migrate the task to a performance or a power-efficient CPU. Figure 2.7 gives a visual example of that. The task when first scheduled starts at the power-efficient cluster and if the CPU utilization gets above a certain threshold the scheduler moves the task to the performance cluster. If the utilization drops then it moves back to the power-efficient CPU. The threshold levels are dependent on implementation, but in all cases are chosen apart enough to prevent overly zealous switching. More information about the different switching scenarios can be found here [8].

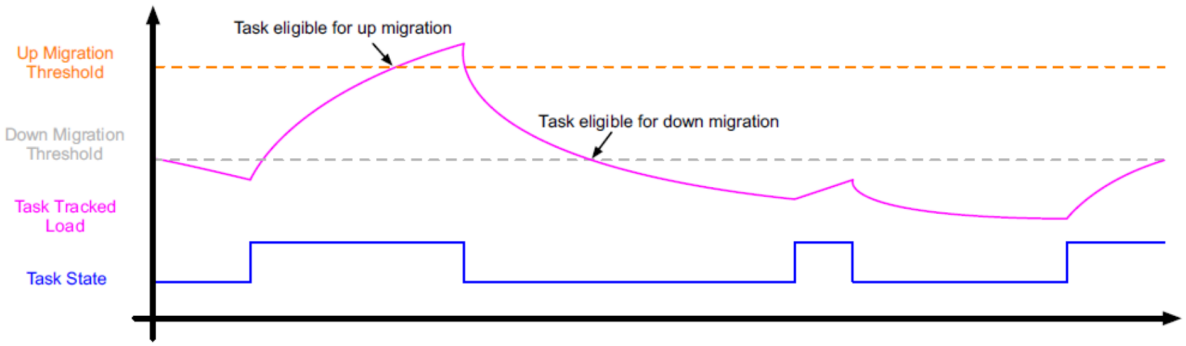


Figure 2.7: **Example of CPU utilisation-based scheduling on *big.LITTLE*.** The CPU state is monitored when executing the task, which is migrated to the *high-performance* or the *energy-efficient* processing core when the task load passes the corresponding *up* or *down* threshold [8]. The utilisation thresholds are defined in the kernel and are specifically tuned to prevent excessive task switching between the two processor types.

2.4 Summary

This chapter outlines the background research regarding the current state of energy management and power modelling for embedded systems. Existing solutions are explored in detail and the importance of HES in the Mobile industry as well as the need for advancements in energy management solutions are determined.

Related work in the field of Power Modelling is also summarised. The majority of power models do not consider the capabilities of heterogeneous systems and predominantly focus on capturing the behaviour of isolated processing units, which is an issue that the proposed heterogeneous models in Chapter 7 directly address. Some key models were listed, which are validated with the developed methodology and are compared against the custom generated models in Chapters 5 and 6.

Additionally a detailed description of the ARM *big.LITTLE* SoC is provided. This includes details about the two processor types in the system - the ARM Cortex-A15 and the ARM Cortex-A7 as well as the PMU and hardware events available on the chip. The operation of the current CPU utilisation-based Energy Management policy for the ARM *big.LITTLE* SoC is also presented in detail.

CHAPTER 3

DEVELOPMENT PLATFORMS

Early research on *big.LITTLE* SoCs involved using simulators due to unavailability of suitable hardware platforms. Since then several companies have come up with development boards for *big.LITTLE*. The ideal development platform for the proposed approach to power modelling is a system which has both PMU events as well as sensors to collect the CPU power to be modelled. During the research 3 different boards implementing the ARM *big.LITTLE* SoC were used to develop the power model generation methodology. Information about these 3 development platforms is given in this Chapter.

Section 3.1 gives an overview of the HARDKERNEL ODROID XU+E platform [14], which is used for the initial development of the methodology and generating the first power models. The purchased board had a critical problem, however thanks to Mr Eric Van Hensbergen from ARM, initial data was obtained.

The second platform used in this research is the ARM Versatile Express Motherboard [15] with CoreTile Express A15x2_A7x3 (TC2) [17]. Overview of this platform is given in Section 3.2 This platform is well supported by ARM and Linaro software-wise, but the hardware manufacturing process for the CoreTile Express TC2 is outdated, so the results that were obtained were not very representative of modern technologies. The results from this platform were later compared to the results obtained using the HARDKERNEL ODROID XU+E platform.

The third and final development platform used is the HARDKERNEL ODROID XU3[18], described in Section 3.3. Extra focus is put on the ODROID XU3, since it is the development board used for the majority of the experiments and most of the methodology was developed on it. It has the most features and is the best supported of the three.

This Chapter concludes with a commentary on the challenges and key insights obtained from using the three developments platforms presented in Section 3.4.

3.1 ODROID XU+E

The ODROID XU+E development platform by HARDKERNEL [14] is the first board used in this research. It has an ARM *big.LITTLE* chip in the form of a Samsung Exynos 5410 SoC [63], which features four ARM Cortex-A15 (big) [74] and four ARM Cortex-A7 CPU (LITTLE) [77] and four Texas Instruments INA231 sensors [84], measuring the ARM Cortex-A15, ARM Cortex-A7, RAM and GPU power. More information about the SoC is detailed in Section 1.2. Figure 3.1 shows the top view of the board.

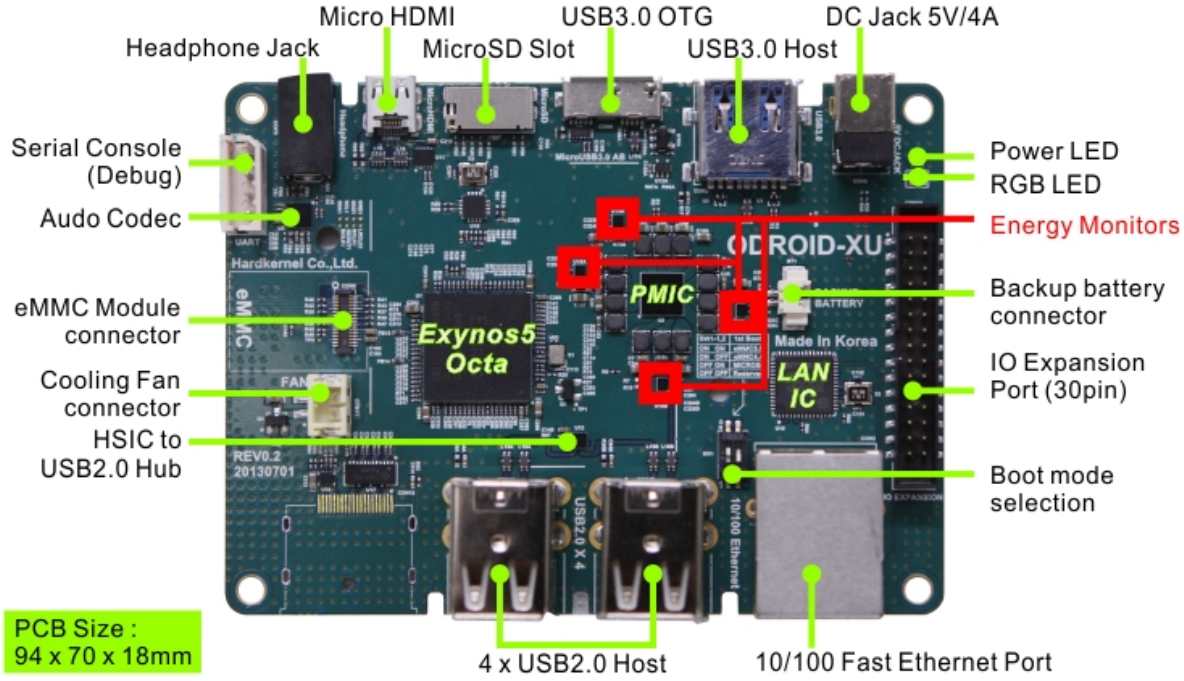


Figure 3.1: **Top view of the HARDKERNEL ODROID XU+E development board.** The first platform used in this research to develop the model generation methodology and obtain the initial results. Released in 2013 by HARDKERNEL, the platform features a Samsung Exynos 5410 SoC with 4 dedicated energy monitors - one for each proceeding cluster, one for the RAM and one for the GPU [14].

On paper this seemed the ideal development platform specifically for this project and it was sufficiently cheap to warrant an immediate purchase. However some critical problems were discovered while setting up the board. Firstly, the kernel scheduler, which enables tasks to migrate from the *big* to the *LITTLE* CPU cluster, was only capable of Cluster or Task Migration [85]. This was not really significant in the beginning since the initial focus was on developing power models, but it did provide overhead to the experiments, since both workload execution and data collection were done on the target processing cluster.

The platform was set up with a minimal Ubuntu Server 12.04 and the latest kernel available from HARDKERNEL. Afterwards the workflow, described later in Chapter 4,was set up and

then a second problem was discovered. Due to a hardware fault with the ODROID XU+E board, the PMU was inaccessible and perf [86] was not able to collect any events, which was critical to progress. Luckily the project industrial sponsor Dr Matt Horsnell, was able to provide a point of contact with Mr Eric Van Hensbergen, who generously offered remote access to a stable ODROID XU+E platform, so work was able to proceed.

Another issue was that the TI sensors were set up with a really low sampling rate (about 3 samples per second) so long workloads were needed to ensure collection of enough regression points, which is one of the things that guided the choice of using cBench as workload. More details on the workloads used in the power model generation methodology are given later in Chapter 4.

In this initial stage of the research the workloads were run on the maximum frequency of the ARM Cortex-A15 (1600 Mhz) and minimum on the ARM Cortex-A7 (500 Mhz), because the goal was to test the methodology and use the extremes as case study. There was also a limit on the number of events that could be collected, since the ARM Cortex-A7 PMU has 5 event counters, of which one is dedicated to cycles. In contrast, the ARM Cortex-A15 has 6+1 counters, however for the sake of consistency 4 PMU events + CPU cycles were collected for both CPUs. The 4 events chosen were the ones read by default with perf: instructions, cache references, cache misses and bus cycles. Details of these initial models are presented in Section 5.1.

3.2 ARM Versatile Express Motherboard with CoreTile Express TC2 Daughterboard

As work progressed, the industrial sponsor provided access to another platform to test the *big.LITTLE* power models on. The ARM Versatile Express Motherboard [15] [16] is a configurable system capable of hosting a number of configurations. For the purposes of this research a CoreTile Express A15x2_A7x3 (TC2) Daughterboard [17] which is a *big.LITTLE* system [6] was connected to the ARM Versatile Express Motherboard. Figure 3.2 shows a top view of the ARM Versatile Express Motherboard and Figure 3.3 shows a block diagram of the CoreTile Express TC2 Daughterboard.

The system implements 2 ARM Cortex-A15 and 3 ARM CortexA7 processing cores in a *big.LITTLE* configuration. The board was set up with a terminal-based version of Ubuntu Developer with the latest kernel, available from Linaro [88]. There were no issues with the PMU and perf was working correctly, which made the model generating platform relatively straightforward to migrate from the ODROID XU+E. First step was to reproduce the experiments ran on the ODROID XU+E and generate respective models for the new platform.

The only drawback of the development platform is that it uses an older manufacturing process and thus its power usage is not really representative of a real-world device, in the

Figure 3.2: **Top view of the ARM Versatile Express Motherboard.** The second development platform used in this research. It is intended to work with a selection of *daughterboards*, providing a number of experimental ARM based platforms [15]. For this research it was used with an early implementation of a *big.LITTLE* SoC. The platform contains power sensors for both CPU islands, which made it suitable for power modelling research.

way that the HARDKERNEL ODROID XU+E is since it has a Samsung Exynos SoC, which is actually used in commercial devices. However, this board, has very good hardware sensors, including an accumulating energy monitor. This made collecting power usage information very convenient.

cBench was run on a similar configuration as on the ODROID XU+E, which is the fastest possible frequency on the ARM Cortex-A15(1200Mhz on the CoreTile Express TC2 Daughterboard) and slowest possible on the ARM Cortex-A7(175Mhz on CoreTile Express TC2 Daughterboard). In addition to generating power models a few additional experiments were done. A "temperature spike" experiment was also performed to see how prolonged usage without CPU cooling can affect the SoC power usage. This was done by running the consumer_jpeg_c benchmark from the cBench suite 50 times with and 50 times without a cooling fan. That specific benchmark was chosen, since it is quick and performance intensive, so there was no long waiting period for the CPU to start dissipating large amount of heat. This experiment was done in order to further investigate how the physical environment can affect

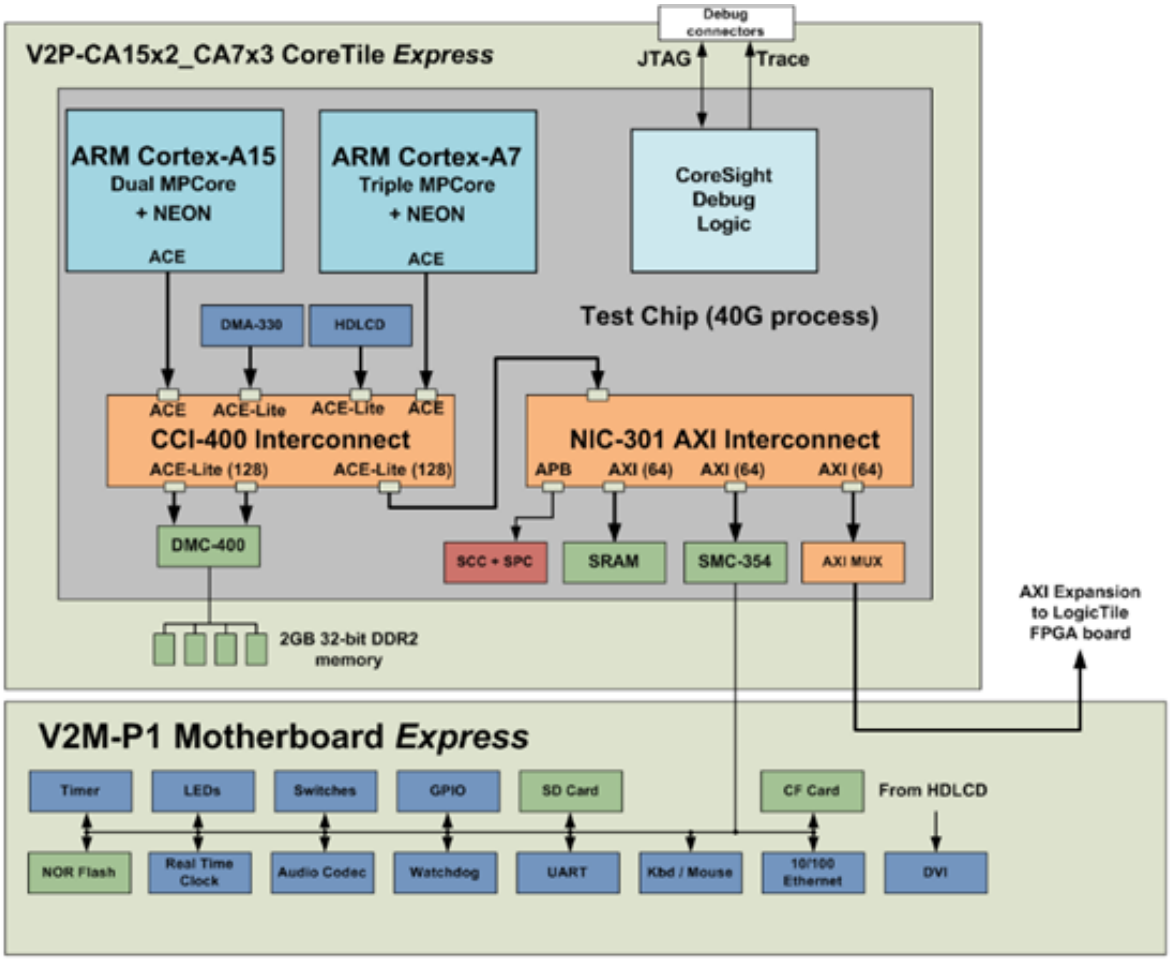


Figure 3.3: **Block diagram of the Arm CoreTile TC2 System Daughterboard.** This attachment board was used with the ARM Versatile Express Motherboard to form a usable *big.LITTLE* system in order to develop a power modelling methodology. The chip has 2 ARM Cortex-A15 and 3 ARM Cortex-A7 processing cores, split in two clusters, along with power sensors for each cluster [87].

the SoC and how to take that into consideration when trying to build a power model. As a final experiment the performance and energy usage between the ODROID XU+E and the ARM Versatile Express Motherboard with CoreTile TC2 Daughterboard was compared and it was identified that the more modern manufacturing process significantly reduces energy consumption. These results are presented in detail later in Chapter 5.

3.3 ODROID XU3

At the time of finishing the power model generation experiments on the ARM Versatile Express Motherboard with ARM CoreTile TC2 Daughterboard, HARDKERNEL released another development board featuring the *big.LITTLE* SoC. The HARDKERNEL ODROID XU3 [18]

development platform has all of the features required to allow developing PMU-based power models. It has an ARM *big.LITTLE* chip in the form of a Samsung Exynos 5422 SoC, which features four ARM Cortex-A15 (the *big*) [74] and four ARM Cortex-A7 CPU (the LITTLE) [77]. The board also has four Texas Instruments INA231 sensors [84], measuring the ARM Cortex-A15, ARM Cortex-A7, RAM and GPU power, current and voltage. These are highlighted in red on Figure 3.4. Shin et al. [89] have explained the design and topology of the Exynos SoC in more detail.

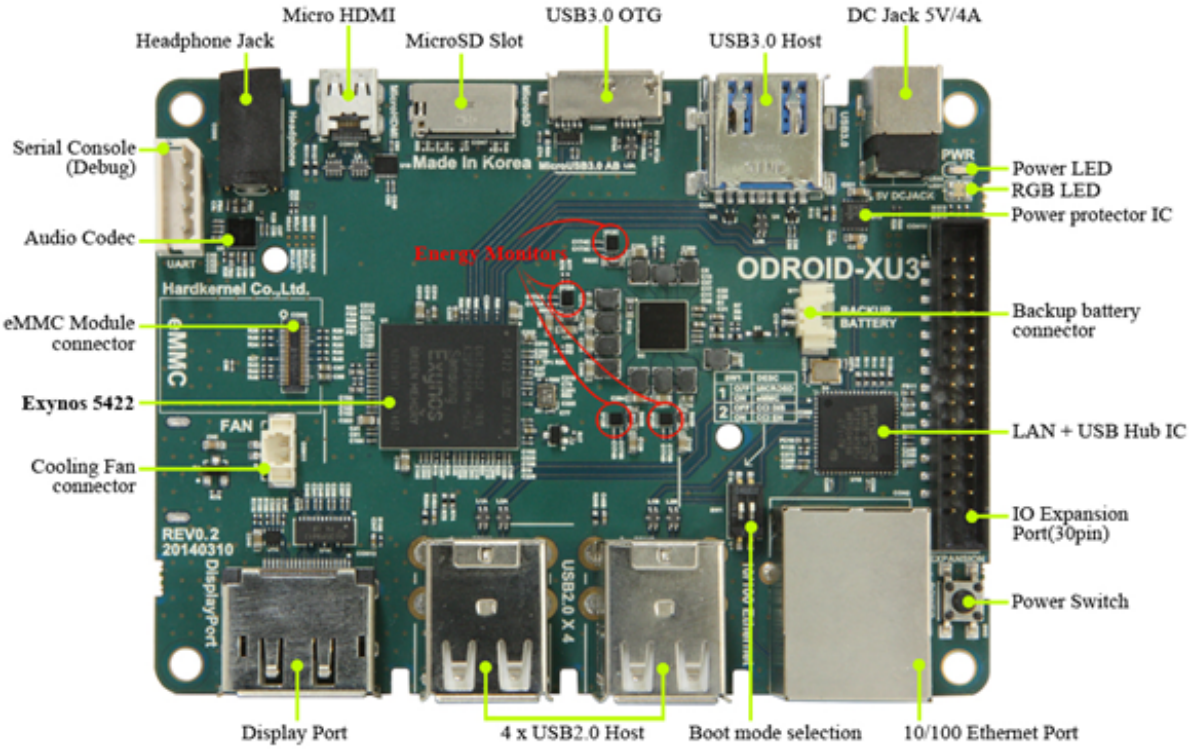


Figure 3.4: **Top view of the HARDKERNEL ODROID XU3 development board.** The final development platform used in this research. It features a Samsung Exynos 5422 Octacore SoC with fully working GTS and also has 4 on-board energy monitors - one for each processing cluster, one for the RAM and one for the GPU [18].

This platform is one of the few ones with working GTS [85] which provides the ability to use it in future stages of research. Another very important feature is that this platform has a working PMU unit, which often does not work on development platforms using *big.LITTLE* as documented by Walker et al. [61] and the initial research efforts on the HARDKERNEL ODROID XU+E board.

Another interesting feature is that the TI energy monitors have a low sampling rate at about 2 samples per second. The ideal workloads for this system would be exhaustive benchmarks with diverse behaviour in order to capture different scenarios and long runtime, which is why cBench [19] and PARSEC 3.0 [20] were used. Specific mechanisms for experiment data

collections were developed, which reduce experiment power and performance overhead. Details of the stages and techniques used in the methodology are available in Chapter 4.

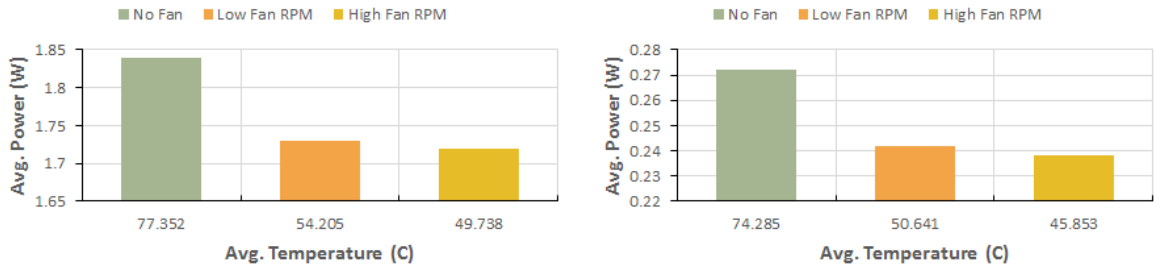
The platform was set up with a minimal Lubuntu 12.04 running the latest kernel available from the board support team. The platform OS is chosen to be small enough to avoid large background software overhead, buts still have enough features to provide easy software development. The software support allowed easy migration of the methodology to the new platform. Because the HARDKERNEL ODROID XU3 is a successor to the HARDKERNEL ODROID XU+E and uses a similar file system structure, most of the data collection tools were reused.

Another key feature of the HARDKERNEL ODROID XU3 is that it has a broad DVFS range with the ARM Cortex-A15 having 18 available frequency levels ranging from 0.2-2 GHz and 5 corresponding voltage levels and the ARM Cortex-A7 with 14 available frequency levels from 0.2-1.4 GHz again with 5 available voltage levels. The Voltage/Frequency relationship is presented on Figure 4.3a in Chapter 4. This increased complexity initially proved a significant challenge, resulting in the first power models developed for the system exhibiting very low accuracy. However, this was ultimately overcome by using a specific technique outlined in Section 5.2.2. A significant range in power consumption for both processor types was observed while running the workloads. An average power variation for the single-thread case was recorded of up to 126% at 1.6 GHz and 143% at 0.8 GHz for the Cortex-A15 and Cortex-A7 respectively. This seems to indicate that the conclusion made by Pricopi et al. [50], which states that due to its simplicity, the ARM Cortex-A7 power can be modelled by using single average number for each frequency level, does not hold true for more modern systems. It is argued that due to the high variation of the ARM Cortex-A7 a more complex model should be used to accurately predict its power consumption.

The large amount of energy levels for the system is shown to be unnecessary, when multi-thread power modelling is explored. In the multi-core case the last 2 frequencies from the ARM Cortex-A15 experiments have been omitted, namely 1.9 and 2.0 GHz. This is due to the fact that the PARSEC 3.0 workload would cause the CPU to throttle to 1.8 GHz when running on all 4 available cores, despite the installed cooling unit.

A quick investigation as to how the fan affects the produced power models was also carried out. Figure 3.5 shows the result of the *thermalspike* experiment, which involved running a single workload multiple times consecutively with the on-board fan set up in 3 different cases. It can be observed from the experiment that, at least for the single benchmark, the fan is sufficient to cool the CPU even at low rpm. For the purposes of future experiments, the fan is kept at maximum rpm. Temperature is also omitted from the model. This might seem like a bad idea at first glance, since temperature contributes greatly to power dissipation. This is clearly indicated by the *thermalspike* experiment as well. However, due to the on-board heat-sink and fan, the cooling unit has a very unpredictable behaviour, which is why despite the fact that

the fan seems powerful enough the system still throttles at the high frequencies during the multi-thread experiments. Also, since the CPU temperature is managed via the cooling unit and thermal management controller, which are controlled by the OS, its behaviour cannot be captured and used by the developed hardware-event-based power models.



(a) ARM Cortex-A15 Data (b) ARM Cortex-A7 Data

Figure 3.5: **Power/Temperature relationship on the ODROID XU3 development platform under three Thermal Management profiles.** Results of repeatedly running a single-thread benchmark on the ODROID XU3 board under 3 different configurations for the on-board cooling system. The benchmark is executed on a single ARM Cortex-A15 core (on the left) and an ARM Cortex-A7 core (on the right).

Last but not least, another interesting feature of this platform is the eMMC card slot in addition to the standard mSD slot. A significant variation was identified when comparing results obtained using and eMMC card and a mSD card as the board driver. This prompted a detailed investigation of the effects of memory system on power. The outcome of those experiments clearly showed that the eMMC card is much more stable and more suited for power model development. The result of this investigation is presented in Section 5.6.

3.4 Summary

Brief details about the development platforms used in this research are presented. Some of the problems with the HES are listed, such as lack of support and features and inconsistent performance. A lot of unexpected variables like badly realised thermal management and memory card volatility influence the system greatly and a lot of effort was spent trying to understand and minimise these factors. It is safe to say that finding a suitable hardware development platform for this research was a very big challenge. In the end, however, the HARDKERNEL ODROID XU3 platform was successfully used to develop the methodology and produce accurate real-time PMU-based models. Unfortunately all three of these development platforms have been discontinued with HARDKERNEL no longer providing on-board power sensors in their latest products. However, interest in the latest generation *big.LITTLE* is growing with more and more companies including ARM wanting to produce research-viable boards.

Chapter 4

Methodology

This chapter describes the novel methodology for generating real-time power models, developed during the course of this research. It goes in detail about all the steps and processes involved and the characteristics of the final completed workflow. The completed methodology involves many different control processes and analysis scripts, which can be split into three distinct stages:

1. Data Collection - The first stage involves setting up the software scripts and utilities on the development platform along with the workloads. A custom control script is used to direct program execution and collect sensor and PMU event samples at regular intervals. More details about the platform set-up and used software are given in Section 4.1.

2. Data Processing - The next stage processes the collected experiment data. This involves synchronising the different measurements and concatenating them in one big dataset. This step is necessary since the model generation software analyses the data and computes the model coefficients offline. This stage is described in Section 4.2.

3. Model Generation - The final stage of the methodology involves using a linear regression algorithm to analyse the processed experiment data and produce power models for the system. This is another multi-step process in which involves a number of custom algorithms. Search algorithms have been developed that go through the experiment data to identify the most optimal set of events to use in the power models, according to various criteria. This stage is also used to interpret the experiment data to validate other published models and compare those results against the proposed power models. A complete breakdown of this stage is available in Chapter 4.3.

A comprehensive overview of the three different stages of the methodology including the internal procedures and data involved are shown in Figure 4.1.

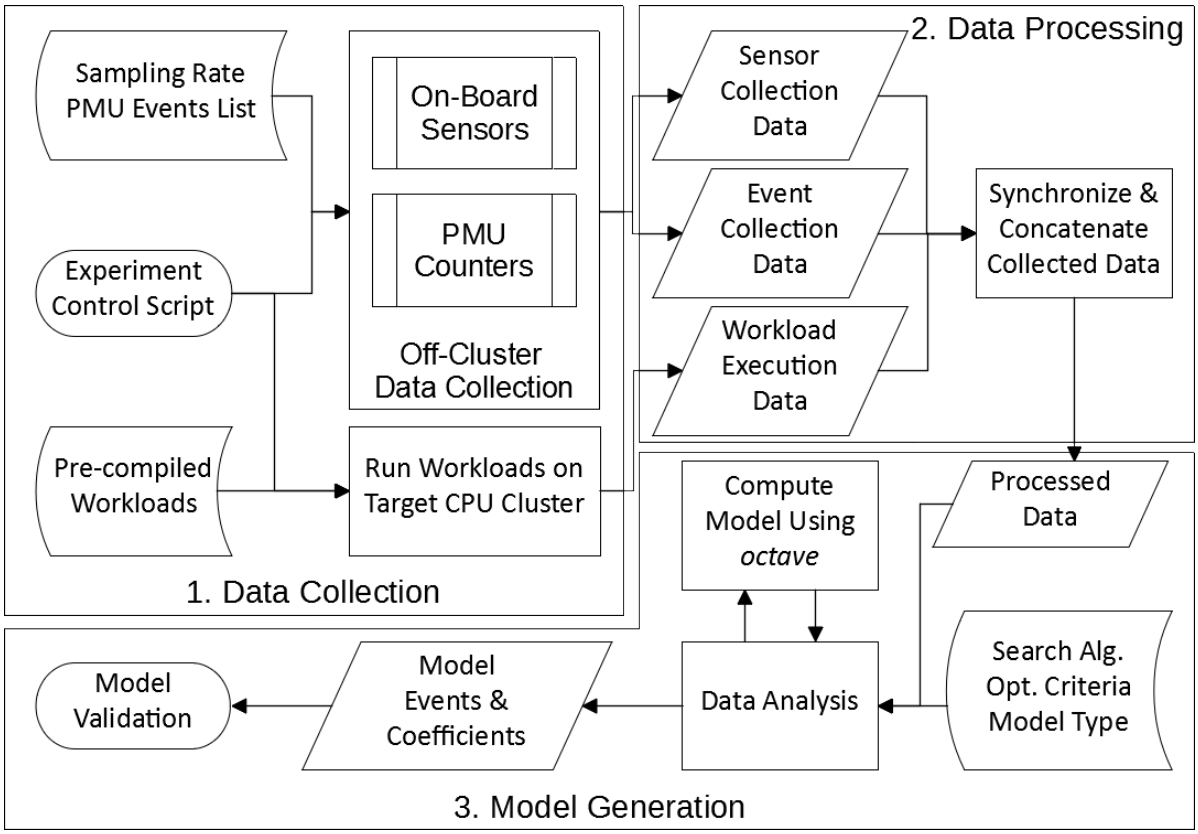


Figure 4.1: **Power modelling methodology stages.** The systematic methodology developed in this research involves many steps, grouped into three distinct stages - Data Collection, Data Processing and Model Generation.

Each stage in the methodology involves many custom control scripts, thoroughly commented in order to enable easy adoption and continuation of the methodology by other researchers. An overview of the methodology software, obtained using the code analysis tool *cloc* [90], is given in Table 4.1.

Table 4.1: **Characteristics of the methodology code for each stage.**

Stage	Language	files	blank	comment	code
Data Collection	Various	33	1041	1023	3476
Data Processing	Bourne Shell	17	551	1099	4134
Model Generation	MATLAB	3	70	108	241

4.1 Data Collection

Data collection consists of 3 key components - system configuration, workload selection and finally program control. Details about each one are given in the following Subsections.

4.1.1 Experimental Setup

The first operation is to set up the platform for the experiment. It involves updating and patching the kernel to provide access to the ARM Cortex-A15 and ARM Cortex-A7 PMUs in order to collect the hardware events as well as downloading the support tools, used in the methodology for program execution control. This step differs from platform to platform so it needs to be done manually. ***cset*** [91] and ***cpufrequtils*** [92] [93]. Downloading and compiling the workloads - cBench [19] for the single-thread case and PARSEC 3.0 [94] for the multi-thread case. And last but not least downloading the custom control script from the project github repository, which will guide the data collection process - ***MC_XU3.sh*** [95]. All the developed project software, excluding the workloads which are already available online, is uploaded online in order to make setting up new platforms less cumbersome.

A key component in this process is the choice for which type of memory card to host the OS and experiment software. During the course of the research the eMMC card is compared against a microSD class 10 memory card. The final methodology uses an eMMC memory card on the ODROID XU3, which provides a very stable environment. Figure 4.2 shows the platform variation in runtime, average power, temperature as well as the **CPU_CYCLES** hardware event while running the cBench (single-thread) and PARSEC 3.0 (multi-thread) workloads on both processor types using the same eMMC card as driver. The results clearly indicate that the eMMC is a much more stable memory card with consistent performance and variation between experiment runs below 5% for all four markers. The microSD card on the other hand had sudden spikes of up to 55% and 28% workload runtime variation for the Cortex-A15 and Cortex-A7 respectively. Details about the direct comparison between the microSD and eMMC cards are available in Chapter 5, Section 5.6.

4.1.2 Workload Characteristics

A lot of time was spent on choosing a suitable workload to use in the methodology. The ideal workloads would be exhaustive benchmarks with diverse behaviour in order to capture different scenarios and long runtime, because of the hard set 0.5s sampling rate on the ODROID XU3 on-board power sensors.

A few open-source options were tried ranging from simple performance benchmarks like Dhrystone [96], Whetstone [97], LINPACK [98] to complex test suites like MEVBench [99], Parboil [100], Rodinia [101] and the benchmarks available through the phoronix-test-suite [102].

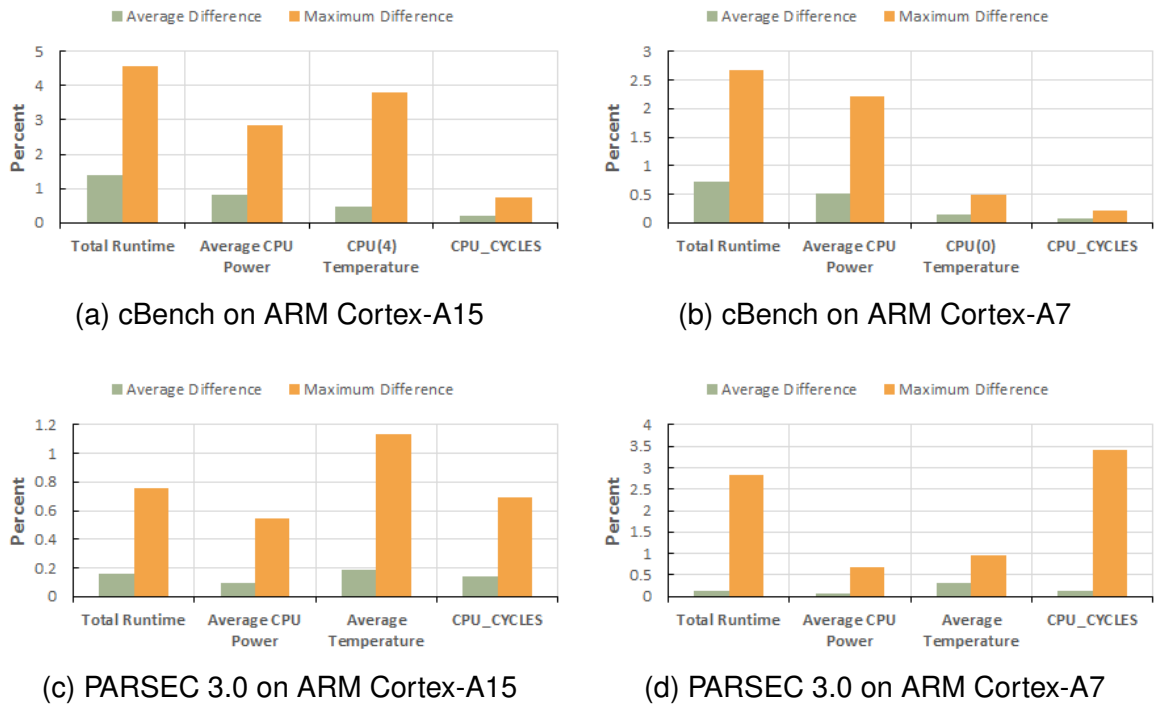


Figure 4.2: **Difference between executions of the two workload types on the ODROID-XU3 board.** The data presented is the *Average* and *Maximum Percent Difference* between different executions of the cBench and PARSEC 3.0 workloads for both processor types on the ODROID XU3 development platform, using the eMMC card as OS driver. The showcased metrics are the workload *Total Runtime*, CPU *Average Power*, CPU *Average Temperature* and the CPU_CYCLES hardware event.

Eventually cBench [19] was selected, because it consists of a large set of smaller benchmarks designed to represent real-life workloads and its long runtime (to ensure enough samples are collected from the energy monitors). cBench is also single-threaded so it is ideal for developing a single-thread model, which was believed to be a first and necessary step in this research.

For the multi-thread case PARSEC 3.0 [94] was used, since it is well established in the research community and also consists of several smaller benchmarks. It is highly configurable and the number of threads for most of the workloads in the suite can be adjusted. This makes it ideal for the 8 core development system, since it allows exploration of all the possible multi-core configurations of the system.

4.1.2.1 cBench

cBench was selected as the single-thread workload for the methodology. Choosing a workload representative of real-world applications is preferable to using custom microbenchbarks. The main goal of this work is to generate empirical models, which are prone to overfitting the

workload they are trained on [58], therefore the models should be trained on workloads that are as close as possible to real-world applications. cBench has been successfully used by other researchers as a test and train workload to develop accurate power models [49] [61].

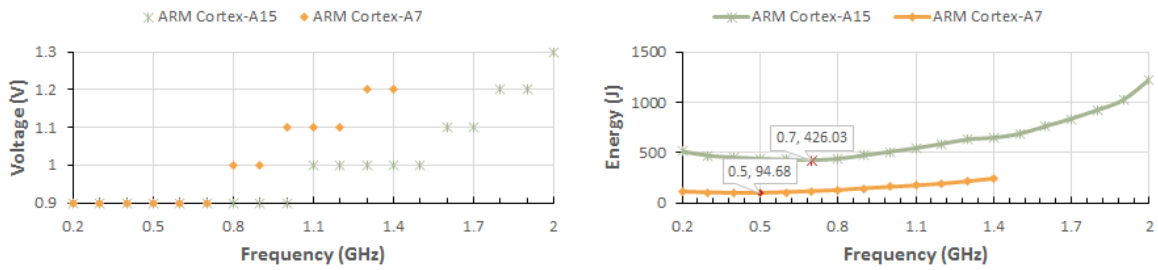
cBench is a collection of 32 benchmarks, split into 7 categories. For the initial results obtained on the ODROID XU and ARM Versatile Express Motherboard with CoreTile TC2 Daughterboard a partial split detailed in Appendix B.1 is used. The model is trained using the larger set and then test on a representative of each category.

For the recent results a more balanced split is used, which is listed in Appendix B.2 When tested on the ODROID XU3, out of the 32 benchmarks, two could not successfully run. ***consumer_mad*** would not run compile properly and ***consumer_lame*** would cause a buffer overflow. All the remaining 30 benchmarks are split randomly in half and 1 set is used to train and 1 to test the model. This split is computed once and then used for the rest of the model generation and validation steps in order to maintain consistency when comparing the accuracy of the power models generated and validated using the methodology. Keeping the workload split the same for all generated models ensures and differences would only occur from the characteristics of the model, a.k.a. the markers used for prediction (PMU events, physical information or system state), and not from any fitting bias due to a different selection of the training benchmarks.

cBench was used [19] as the single-thread workload compiled for the ODROID-XU3 using ***gcc*** with the ***-O3*** flag. The workload power characteristics are presented in Figure 4.3. There are a total of 18 frequency levels available for the Cortex-A15 and 13 frequency levels for the Cortex-A7 with 5 available voltage levels, which are shown in 4.3a. Exploring this entire space for the purposes of power modelling is very challenging. This is overcome by producing an individual set of coefficients for the PMU event based model for every frequency level. The result is called a ***per-frequency*** model. More details about this type of model are available in Chapter 5, Subsection 5.2.2. Figure 4.3b highlights that the lowest energy point on the convex E/F curve is not linearly determined. This complex relationship is also used in Nikov et al [21] to justify the need for power modelling in modern embedded systems.

4.1.2.2 PARSEC 3.0

PARSEC 3.0 was used [20] as the workload for the multi-thread model generation and validation. Since the system has 4 CPU cores in each CPU cluster, the execution is explicitly split into 4 categories - 1 core, 2 cores, 3 cores and 4 cores, with each core executing 1 workload thread. The analysis is done on the combined data from all 4 configurations. The initial assumption (besides having more detailed control over the system variables) was that having dedicated models for each configuration (1,2,3 or 4 running cores) would yield greater accuracy, the way splitting the model for every frequency level did. However that was proven by the experiments to not be the case and actually using the data from all four categories produces a better model



(a) Voltage/Frequency Levels

(b) Energy/Frequencty Relationship

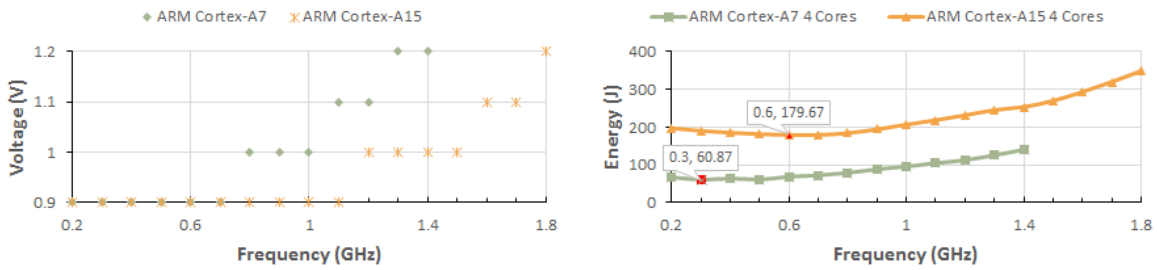
Figure 4.3: **Characteristics of the cBench workload on the ODROID-XU3 board.** The CPU *Frequency* and corresponding *Voltage Levels* for both processor types are presented on the left Figure 4.3a. The *Relationship* between total used CPU *Energy* for each CPU *Frequency* is presented on the right Figure 4.3b with a highlight on the lowest energy points.

overall, even when only tested on a single configuration. The details of this investigation with supporting results are presented in Chapter 6.

Figure 4.4 details the overall characteristics of PARSEC 3.0 on the HARDKERNEL ODROID XU3. As mentioned previously in Chapter 3, Section 3.3 the ARM Cortex-A15 maximum frequency is limited from 2.0 GHz to 1.8 GHz in order to avoid thermal throttling from the Linux kernel. This is illustrated in Figure 4.4a where the frequency and corresponding voltage levels for the system running the multi-thread workload are given. This still provides sufficient data points to use in the analysis. The energy usage data if Figure 4.3b is given only for the 4 Core configuration since it has the most dynamic energy profile and is representative of the full multi-threaded capabilities of the system. It clearly illustrates, that the lowest energy points are not linearly defined by frequency like in the data from cBench in Figure 4.3b.

Appendix B.3 details the randomised benchmark split used in training and testing the multi-thread model. The PARSEC 3.0 set also contains some benchmarks from splash2 with added multi-thread scaling. Only the benchmarks that compiled successfully are used, were able to scale for all 4 core configurations and were long enough to obtain a sufficient amount of data points. The final workload consists of 9 suitable benchmarks, with 4 being used for training and the rest for testing. This is a much more detailed level of exploration than other multi-thread models which just use workloads that leave the compiler/OS to dynamically assign the number of threads.

In addition to the custom experiment a modified version of the PARSEC blackscholes application was used for the collaboration presented in Chapter 7 in which the mathematical model the research group from the collaboration is validated on the HRDKERNEL ODROID XU3 development platform using the methodology.



(a) Voltage/Frequency Levels (b) Energy/Frequencty Relationship

Figure 4.4: **Characteristics of the PARSEC 3.0 workload using 4 processing cores on the ODROID-XU3 board.** The CPU *Frequency* and corresponding *Voltage Levels* for both processor types are presented on the left Figure 4.4a. The *Relationship* between total used CPU *Energy* for each CPU *Frequency* is presented on the right Figure 4.4b with a highlight on the lowest energy points. For easier illustration, only the data from the extreme case of running a symmetric 4-thread workload on all available cores of each processor type has been presented.

4.1.3 Workload Execution

Several tools in the Linux ecosystem were used to help control the execution of the workflow and to read the PMU events and on-board power sensors. The two four-core CPU clusters can be configured separately, which allows many options for the system set-up. This is done in three steps. First, ***/sys/devices/*** is used to identify and *hotplug* the processing cores not required in the experiment. Second, ***cpufreq-set*** from the ***cpufrequntils*** package is also used to set the CPU cluster frequency. Finally ***cset*** is used to assign the workload application to the appropriate cores and isolate all the data collection scripts and OS threads on the opposite CPU cluster. This prevents the collection interfering with the experiment and minimises overhead. All of this is controlled by the ***MC_XU3.sh*** [95] script. A diagram is presented in Figure 4.1

The data collection scripts read the on-board power sensors data and use the Linux ***perf*** tool to communicate with the PMU and collect the events. This is all done in real-time, in parallel to the workload execution at a 0.5s interval for the HARDKERNEL ODROID XU3 development platform. The interval is imposed by the minimal refresh rate of the power sensors and since all measurements need to be synchronised. The PMU event collection is also done at the same interval, despite the fact that the PMU can be sampled much faster.

The final methodology explores the entire set of PMU events available. The number of events that can be collected in parallel from the PMU is also limited to just 7 for the Cortex-A15 and 5 for the Cortex-A7 CPU cluster, with 1 of those registers being reserved for the **CPU_CYCLES** event. With 67 total events available for the Cortex-A15 and 42 for the Cortex-A7, this means that multiple runs are required and collections in order to capture all events for analysis. In order to facilitate data analysis precise timestamps are generated for each measurement so that they can be concatenated later on.

This complicated set-up is necessary in order to minimise experiment interference and variability. Table 4.2 shows the low overhead of the off-cluster data collection. These measurements are the difference between CPU power when not collecting the data samples and when collecting the data samples on the unused processing cluster. It can be seen that the methodology has minimal impact on the measurements, precisely because data collection is done in a way that does not affect normal workload execution. Measurement overhead and noise are a large contributor to increasing model error, which is why reducing them is imperative to generating accurate models.

Table 4.2: **Overhead of the data collection stage of the methodology on the ODROID-XU3 board.** These results present the *Average* as well as *Maximum Difference* of the CPU *Power* and workload *Total Runtime* between execution with PMU event collection enabled and without for both processor and workload types.

	Average Difference		Maximum Difference	
	Single-Thread/Multi-Thread			
Total Runtime	0.16%/1.18%	0.12%/0.95%	0.41%/3.10%	0.42%/7.03%
Average Power	0.27%/0.38%	0.40%/0.62%	0.90%/1.30%	1.64%/5.00%
ARM Core Type	Cortex-A15	Cortex-A7	Cortex-A15	Cortex-A7

4.2 Data Processing

The second stage of the methodology is probably the most important one. In order to make the experiment data usable the different sensor and PMU event measurements need to be synchronized.

4.2.1 Data Synchronization

This is done by utilizing precise timestamps for each measurement, generated during result collection. A complicated data processing script was developed, which synchronizes the benchmark execution times with the sensor and PMU event information in one file. It is available as ***XU3_results.sh*** [103]. If only a single set of PMU events has been collected the synchronized data can already be used for model generation.

4.2.2 Data Concatenation and Analysis

The methodology involves the collection of the full list of available PMU events in order to do a more thorough analysis. The events themselves cannot be gathered concurrently so require multiple experiment runs to capture. After collection, they are first synchronized with their respective sensor and benchmark sample data point using the method in the previous section. This results in many data files with different timestamps.

A special concatenation script is used which averages sensor information and presents all PMU events into one big dataset. This is to enable the event selection machine learning algorithms to traverse the data quicker. The script is called ***XU3_merge_events.sh*** [103]. A pseudocode version of the data concatenation algorithm is available in Appendix C.1.

One additional step is also performed - namely removing any events which have high variability between platform runs and any events that are very specific and do not get triggered by the workload. This ensures that the events list is consistent and stable, which improves model stability as well.

4.3 Model Generation

After collection and processing have been completed by the previous steps in the methodology, the next step can occur, which is generating the power models. This step requires the synchronized data from the CPU power and PMU event samples as well as the full concatenated list of collected events. The analysed data consists of one big datafile, which holds all the samples (one for each line in the file) at the specified sampling interval. The model generation step uses this data and performs two distinct operations - mathematical derivation of the model using the ***octave*** environment and the OLS procedure and automatic model event selection, which searches the full list of collected PMU events and identifies the most optimal set to be used in the model. Both of these steps are implemented via a custom script.

4.3.1 Offline Analysis Using *octave*

After data collection and processing the mathematical analysis is performed on the results off-line on a supporting machine. The workload is split into a train and test set and compute the model using ***octave***.

Ordinary Least Squares, a well-know linear regression algorithm, is used to identify the events that best approximate average power from the train set. The mathematical expression is shown in equation 4.1.

$$\alpha = (X^T X)^{-1} X^T y = (\sum_{i=1}^{n} x_i x_i^T)^{-1} (\sum_{i=1}^{n} x_i y_i) \tag{4.1}$$

Power is used as the dependent variable y in the above equation, also known as regressand. The events are expressed as the X vector of independent variables, a.k.a. regressors. The OLS method outputs a vector α, which holds coefficients extracted from the activity vectors. Then the equation 4.2 is used to estimate power usage using a new set of events.

$$P_{CPU} = \alpha_0 + \alpha_1 \times event_1 + \alpha_2 \times event_2 + \ldots + \alpha_n \times event_n \tag{4.2}$$

The accuracy of the modelled equation is evaluated by using data from the benchmark test set with a new set of power values and events and measure the percent difference (error)

between the measured power and the estimated power by plugging the new events in the equation. Other metrics like Root Mean Squared were explored, but they can be very sensitive to outliers.

In general, data modelling approaches like linear regression are quite dependent on the inputs and equations used. If the model is too simple it might not give accurate predictions, because it does not use a sufficient number of characteristics (events/regressors) to fit the data properly. On the other hand a too complicated model, using many events, can make it hard to compute in real-time and can be prone to overfitting the training data and if the training data is not broad enough it might perform poorly on future types of work that have not been included in the training data. There is a fine balance between simplicity/real-time usability and good performance, but there is a lot of evidence that linear regression models can be used in power optimization techniques in embedded systems and produce accurate models [49], [59], [51].

In the methodology ***octave*** is used just as a computational call. On top of it lies the ***octave_makemodel.sh*** script which controls how the data is split and which events are used in the file. That script is used to generate the per-frequency, as well as intra and inter-core models by modifying the test and train set before the call to octave. Further information about the different types of models that are generated can be found in Chapters 5, 6 and 7. Reference pseudocode implementations of the per-frequency and intra/inter-core model generation scripts can be found in Appendix C.2 and C.6.

The main point to make here is how flexible the methodology is. Any point in time new model generation algorithms can be implemented and even the ***octave*** call can be replaced for something else.

4.3.2 Event Selection

In addition to performing the per-frequency model generation, ***octave_makemodel.sh*** is modified with custom intelligent search algorithm scripts to identify the best power models from the collected events. The impact of this machine learning approach to develop the custom power models is explored in Section 5.5. Three different search algorithms were developed - ***bottom-up***, ***top-down*** and ***exhaustive***. As their names suggest they traverse the PMU events search tree in different ways. Pseudocode examples of the three algorithms are found in Appendix C.3, C.4 and C.5. [1]

In addition to implementing several search algorithms, multiple options for model optimisation criteria are also integrated. Instead of average relative error, event cross-correlation or error standard deviation can be minimised. The ***octave_makemodel.sh*** is flexible and can easily be configured to add even more functionality.

[1] The reason behind including pseudocode versions of the algorithms is because the actual code takes too much space. It can always be seen on-line on the github code repository [103] [95].

4.3.3 Model Accuracy Metrics

Table 4.3 shows the target model error for the single-thread workload on the HARDKERNEL ODROID XU3 development platform. The calculated values are the average power difference between two adjacent frequency levels for both processor types while running cBench. The *Up* value is the difference between a level and the next higher one, for example going from 1.4 GHz to 1.5 GHz, and the *Down* value is the difference between the level and the next level lower, for example 0.8GHz to 0.7GHz. The table gives the average of these differences between all two adjacent levels. These metrics represent the required model error for the ***per-frequency*** level models to predict the power on the level they are generated for. If the model error is higher than the *Up* or *Down* percent difference then the model can predict power values attributed to its adjacent frequency levels and if used as a reference by an EM policy can needlessly trigger a CPU frequency scaling to a less optimal level. The average power difference values provide the tightest constraints that allow the model to work accurately on all of them. Therefore the model error for the ARM Cortex-A15 needs to be lower than 8.69% and for the ARM Cortex-A7 - 13.55%. Identifying the target error is a necessary step in identifying any problems with the methodology and verifying that it can produce accurate models. Due to the limited number of PMU events that can be use and the nature of the automatic search algorithms, it is expected to never have the absolute ideal model. However, a satisfactory model performance can be guaranteed if the error is below the minimum difference between the energy levels, so the model may be usable as a guide to average power at every level.

Table 4.3: **Target error for the single-thread per-frequency power models fitted on the ODROID-XU3 board.** The data presents the CPU *Average Power Difference* between adjacent energy levels while running cBench for both processor types on the ODROID XU3 development platform using the eMMC card as OS driver. Two indicators are used - from a level to the next higher (*up*), or from a level to the previous lower (*down*). The average of these differences between all neighbouring levels represents the model target accuracy.

ARM Core Type	Average Power Difference Up	Average Power Difference Down
Cortex-A15	9.51%	8.69%
Cortex-A7	15.68%	13.55%

The target model error for the multi-threaded workload on the ODROID XU3 platform are presented in Table 4.4. They are computed using the same technique that calculates the target error for the single-threaded execution by obtaining the average power difference between adjacent energy levels. However for this case the data from running PARSEC 3.0 on all four CPU core configurations (1,2,3 and 4 cores under load) is used. The target model error for the multi-threaded scenario is less than 7.57% for the ARM Cortex-A15 and less than 7.04% for the ARM Cortex-A7. The accuracy needs to be even greater than what the single-thread case requires due to the lower energy difference between levels while executing the multi-thread workload. The lower error requirement is a difficult target, that the generated ***per-frequency***

models for the multi-threaded workload are unable to meet. The cause of this and possible solutions to improving the accuracy of regression-based models and meeting this target are presented in Chapter 6.

Table 4.4: **Target error for the multi-thread per-frequency power models fitted on the ODROID-XU3 board.** The data presents the CPU *Average Power Difference* between adjacent energy levels while running PARSEC 3.0 for both processor types on the ODROID XU3 development platform using the eMMC card as OS driver. Two indicators are used - from a level to the next higher (*up*), or from a level to the previous lower (*down*). The average of these differences between all neighbouring levels across all 4 system configurations for the workload (1,2,3 or 4 cores) represents the model target accuracy.

ARM Core Type	Average Power Difference Up	Average Power Difference Down
Cortex-A15	7.57%	7.04%
Cortex-A7	14.81%	12.9%

4.3.4 Model Validation and Comparison

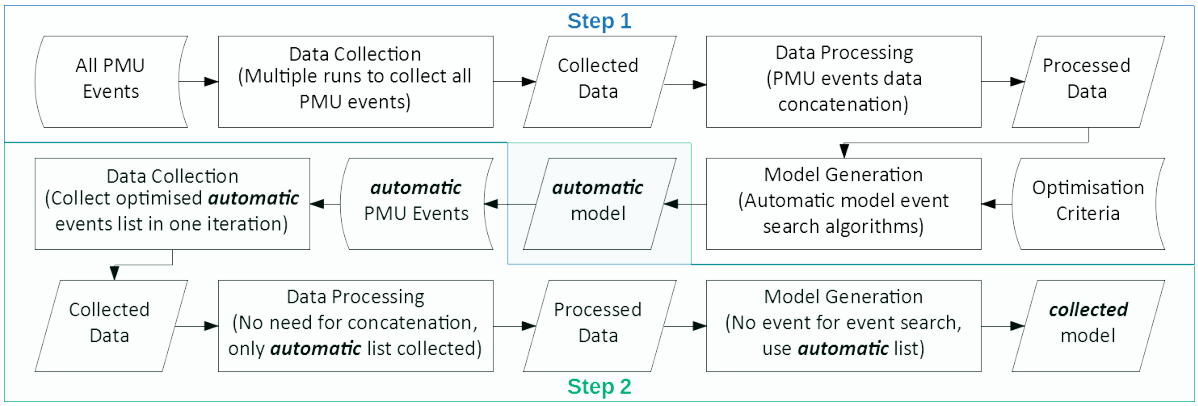


Figure 4.5: **Model generation and validation steps.** The model generation and validation stage of the systematic methodology involves two overlapping steps. *Step 1* generates the ***automatic*** model from the experiment data. *Step 2* revalidates this model on the hardware and generates the final ***collected*** model.

The final stage of model generation is validation, which is a two step process. Firstly an intermediate model is computed using the full PMU events list. This model is called ***automatic*** and it uses the optimal subset of events that was identified. Afterwards this optimal of events is collected on the hardware on its own by repeating the experiment. This is done to eliminate any approximations that can affect the model accuracy, caused by the data processing and event concatenation on the initial data collection. The second validated model is called ***collected***. In simple terms the model is first calculated from a bigger list of events and then the optimal list of events is collected on its own and validated again. This often results in higher error for the ***collected*** model due to the lack of data averaging introduced by the PMU event concatenation

part of the data analysis stage in the methodology. In addition to these steps the finalized ***collected*** model can be compared against other published models to further validate the methodology. An illustration of the steps to generate the ***automatic*** and ***collected*** models is given in Figure 4.5. Validating other published models is done by performing **Step 2** with the ***automatic*** events list at the start of the stage, replaced by the events of the model under test. If the published model was developed on another architecture its events need to be translated for the ARM *big.LITTLE* ISA. This requires a good understanding of both architectures and is done manually by the researcher.

4.4 Summary

The developed systematic model generation methodology is thoroughly described. Great details are given about each of the three stages.

The data collection process is explained with an emphasis on the experiment set-up procedure and the characteristics for the workloads. Details and examples are also given about the eMMC memory used to host the OS as well as the methodology overhead and the development platform power and performance variation between experiment iterations.

The data processing stage is briefly described, with a quick outline of the scripts used to analyse and prepare the experiment data for model generation.

Finally the techniques used to compute the regression-based models and calculate the model error are described. A method for identifying the model performance targets is presented. At the end the two steps involved in generating optimal models from the list of PMU events are detailed. A description of how to use the methodology to validate other published work is also presented.

CHAPTER 5

SINGLE-THREAD MODELS

This chapter describes the single-thread scenario of the power model generation and validation using the developed methodology. The methodology goes though several different stages of development, producing better and better power models along the way. All the generated models during the methodology evolution and all the key insights gained that enabled progression are included.

Section 5.1 introduces the initial stages of the methodology and the first power models, developed for the HARDKERNEL ODROID XU+E and the ARM Versatile Express Motherboard with CoreTile TC2 Daughterboard development platforms. It includes some insights into environmental factors, such as SoC temperature, with regards to power consumption as well as results, comparing the performance and energy consumption of the ARM Cortex-A15 and ARM Cortex-A7 processors.

Afterwards, the first models developed for the ODROID XU3 board are described in Section 5.2. These results include description of the ***per-frequency*** level model generation techniques and comparison against related work.

The next Section 5.4 describes the efforts in analysing the model reproducibility on multiple ODROID XU3 boards. Section 5.5 details the efforts to improve the model accuracy using an automatic search algorithm to identify the optimal list of PMU events for power modelling. This section also details the results of repeating the board variability experiment with the new models.

The ultimate outcome of that analysis was that the memory system used to build the models, namely the eMMC card used with the first ODROID XU3 board, was vastly different than the mSD cards used in the variability experiment. Chapter 5.6 presents the results of the detailed comparison analysis between the eMMC card and mSD card.

Sections 5.7 and 5.8 detail the final stages of the methodology and compare the final single-thread power models for both processor types to other published work.

5.1 Initial Results

The initial methodology involved collecting a limited set of PMU events for the frequencies at the SoC extreme on the target board. This means that only the events for the highest frequency available for the Cortex-A15 and the lowest available for the Cortex-A7 were collected. The initial justification was that if the model could correctly capture system behaviour in the extreme scenarios it would work on all other. This assumption has been proven wrong and has been corrected in later iterations of the methodology. Still these models serve as a good starting point to the analysis.

This section describes the models obtained for both ODROID XU+E 3.1 and ARM Versatile Express Motherboard with CoreTile TC2 Daughterboard 3.2. cBench was used for the methodology workload and each experiment was ran 5 times to obtain a good average. The PMU events that were collected are the standard ones visible through the perf aliases, namely cycles, instructions, cache references(CR) and cache misses(CM). At this stage the script calls to perf to select events via their RAW identifiers have not been developed. For the power model analysis combinations of the events are manually combined and tested and the corresponding model error is reported. Initially the model was trained and tested on the full cBench set and the best performing events are then trained and tested on the *partial* set in Appendix B.1.

5.1.1 ODROID XU+E Power Models

5.1.1.1 ARM Cortex-A15

Figure 5.1, presents the average measured CPU power consumption and total energy usage by each benchmark category in the cBench set averaged over the 5 runs of the experiment. The average system power and energy usage is also measured. The values are obtained from the on-board sensors. The workloads are quite varied, which ensures the model does not overfit on a specific type of workload and lose its flexibility. For every run the 4 events mentioned previously are collected and the different models are built as explained earlier.

Appendix D.1 shows the regressand coefficients for the different models and table 5.1 shows the resulting Predicted Power Standard Deviation, Average Root Mean Squared Difference between prediction and actual measurements and the Average Dynamic Contribution, which shows what percentage of the predicted power consists of the dynamic events by subtracting the constant part from the prediction.

Another indicator for a good model is the Standard Deviation of Predicted Power. The measured cBench Power standard deviation is 14% so ideally the generated model should be able to capture the full range and get as close to that as possible. The Average Dynamic

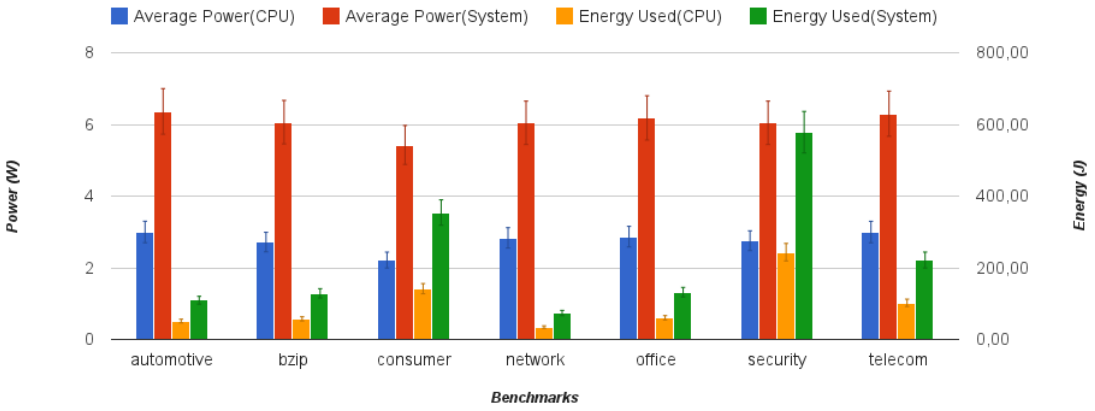


Figure 5.1: **Characteristics of cBench running on ARM Cortex-A15 on the ODROID XU+E board.** The figure presents the CPU as well as the System *Average Power* and *Total Energy Used* for each group of benchmarks in the cBench suite while running on the ARM Cortex-A15 processor at 1.6GHz on the ODROID XU+E development platform.

Table 5.1: **Performance of the single-thread power models for ARM Cortex-A15 on the ODROID XU+E board.** The presented data uses three different metrics to illustrate the accuracy of the various models generated from combinations of the four accessible hardware events - CPU *Cycles*, *Instructions*, *Cache References(CR)* and *Cache Misses(CM)* as well as *Cycles-per-Instruction(CPI)* and *Instructions-per-Cycle(IPC)*.

Model Type	Predicted Power Standart Deviation	Average RMS Difference	Average Dynamic Contribution
Full Cycles	5,20%	11,60%	-6,43%
Full Instructions	3,58%	12,31%	-8,03%
Full CPI	3,48%	12,31%	-5,74%
Full IPC	6,06%	11,42%	15,73%
Full CR	5,58%	11,21%	-8,71%
Full CM	8,89%	9,31%	-8,15%
Full IPC,CR,CM	9,11%	8,92%	-2,71%
Partial IPC,CR,CM	5,49%	4,36%	2,70%

Contribution should be positive, which means the dynamic events in the system are correctly modelled. If it is negative it means a greater emphasis is placed on the static power, so some of the dynamic power costs are masked and the model can be prone to overfitting the data. However the main task is to minimise the Average RMS Difference, since that is the main indicator that the model fits the data correctly.

For the Cortex-A15 the top regressands based on models built and tested on the full benchmark set are ICP,Cache References and Cache Misses, with Cache Misses achieving the best single-event performance with 9,31% difference. This is not surprising since cache events are shown to contribute to power greatly so the number of such events should play a large role

in determining the power consumption of the task. However the best performing model using *IPC,CR* and *CM* as events achieves a rather poor 9% difference, when it should ideally be close 0%. Also the Power Deviation is 9% instead of the 14%. This means the model cannot accurately capture all of the dynamic range of the test suite and more events/increased complexity should be considered. However when built on the partial set and tested on the smaller number of benchmarks it actually achieves better results getting as close as 4.5% and actually having a positive dynamic contribution component. This indicated on one hand that cBench includes a wide range of workloads and their specific choice can affect the model greatly and including all benchmarks to train actually introduces greater variability of the model. The simple model is quite flexible, but it seems to be under fitted if such small variations in the training set. Looking at specific benchmarks the model shows poor performance on bzip2d with 20% difference and almost all of the consumer benchmarks getting on average a 25% error for that set which is huge. Further investigation is needed in order to find suitable events which can help capture the power consumption contributions of those specific workloads. That is why the testing is done on the first benchmark of every set. If the whole consumer set and bzip2d are included in testing the reported model performance will be greatly reduced.

5.1.1.2 ARM Cortex-A7

The experiments were repeated for the Cortex-A7. The LITTLE core has a much smaller dynamic power range, because it is low power and the workload is ran on the smallest possible CPU frequency, so it is expected to use much less energy than the big core.

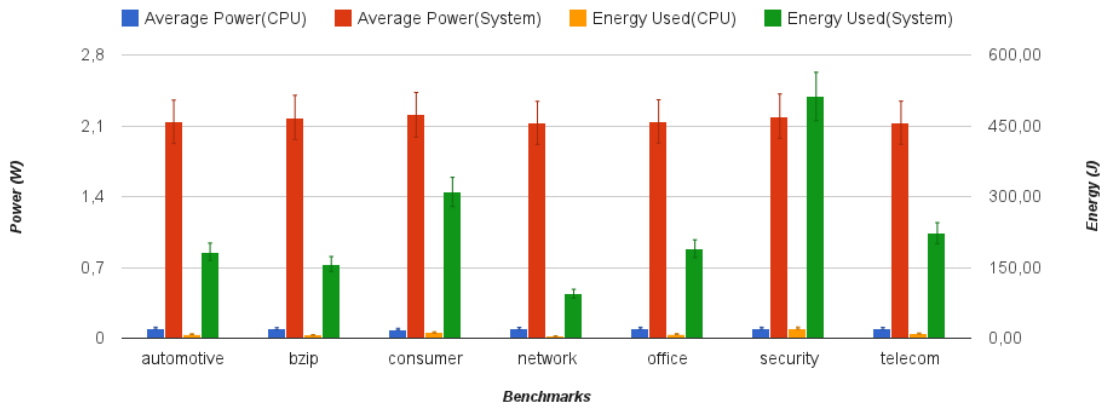


Figure 5.2: **Characteristics of cBench running on ARM Cortex-A7 on the ODROID XU+E board.** The figure presents the CPU as well as the System *Average Power* and *Total Energy Used* for each group of benchmarks in the cBench suite while running on the ARM Cortex-A7 processor at 0.5GHz on the ODROID XU+E development platform.

The Cortex-A7 completes cBench on average about 2.5 times slower that the Cortex-A15, however the total energy consumption is on average 5-6 times lower than the Cortex-A15,

which is quite significant. The Cortex-A7 is approximately about double the efficiency of the Cortex-A15, which makes it much suitable for parallel mobile systems than the Cortex-A15, since more performance per core is achieved. However the mobile software stack is still predominantly single-threaded, so a good sequential performance that the Cortex-A15 provides, is needed to meet heavy consumer demands. As software matures it is predicted that more systems will emerge, where the majority of cores are cost-effective ones like Cortex-A7, and they will rely on parallelisation to achieve high performance.

Table 5.2: **Performance of the single-thread power models for ARM Cortex-A7 on the ODROID XU+E board.** The presented data uses three different metrics to illustrate the accuracy of the various models generated from combinations of the four accessible hardware events - CPU *Cycles*, *Instructions*, *Cache References(CR)* and *Cache Misses(CM)* as well as *Cycles-per-Instruction(CPI)* and *Instructions-per-Cycle(IPC)*.

Model Type	Predicted Power Standart Deviation	Average RMS Difference	Average Dynamic Contribution
Full Cycles	2,06%	4,66%	-3,52%
Full Instructions	1,08%	4,99%	-2,15%
Full CPI	1,70%	4,85%	-5,84%
Full IPC	3,03%	5,15%	7,49%
Full CR	1,41%	4,80%	-2,08%
Full CM	3,39%	4,25%	-2,93%
Full C,CR,CM	3,57%	4,04%	-3,31%
Full C,I,CR,CM	3,57%	4,04%	-3,18%
Full CPI,CR,CM	3,41%	4,23%	-2,86%
Partial C,CR,CM	3,51%	3,76%	-0,64%

Compared to the Cortex-A15 Power Models, the Cortex-A7 ones exhibit some different characteristics. For example the Power Standard Deviation of cBench is only 6,32% which is a lot smaller than the Cortex-A15, mainly because the core power is much lower and also it has low-power components (short pipeline, smaller cache size) with which the static power consumption is a lot more prevalent. However the best performing model only achieves 3.57% standard deviation of power, which is even worse that the Cortex-A15 (A7 is 50% off the measured Power Standard Deviation). Another very interesting observation is that this time just Cycles on its own produces a better model than Instructions, CPI or ICP, however Cache Misses is still produces the best single event power model. Overall the best model seems to be Cycles, Cache References and Cache Misses and the same relationship between the full set model and the partial set model is present as on the ARM Cortex-A15 (partial is better). Adding the Instructions event to the model seems to make no improvement. The model again struggles with predicting the consumer set of benchmarks, but this Cortex-A7 model achieves about 10% average Power difference. Some further investigation with varying the frequency should be done, since from comparing the Cortex-A15 and Cortex-A7 models initial conclusion

is that with increasing the frequency CPI and ICP show increasing contribution to the power consumption compared to just Cycles and Instructions.

5.1.2 ARM Versatile Express Motherboard with CoreTile TC2 Daughterboard

The same methodology was replicated on the other board provided by the industrial sponsor.

5.1.2.1 Cortex-A15

The ARM Cortex-A15 processor on the ARM Versatile Express Motherboard with CoreTile TC2 Daughterboard is expected to be considerably slower and use more energy than the ODROID XU+E one since this platform was produced using an older manufacturing process. The two systems also have different memory systems, so memory bound applications could also potentially differ greatly. The following figure shows average CPU power for each set of microbenchmarks in cBench and the average PCU energy. System-level information was not included, since the ARM Versatile Express Motherboard with CoreTile TC2 Daughterboard has a lot of peripherals on the motherboard, which are not present in a standard SoC and introduce quite a large system energy. Also the chip average power consumption can be lower due to the fact that this board runs slower than the ODROID XU+E, but overall due to the inferior manufacturing process total energy usage should still be relatively higher. Also the ARM Versatile Express Motherboard with CoreTile TC2 Daughterboard runs a much simpler OS than the other boards, which means less overhead and this can affect the average system power and energy consumption.

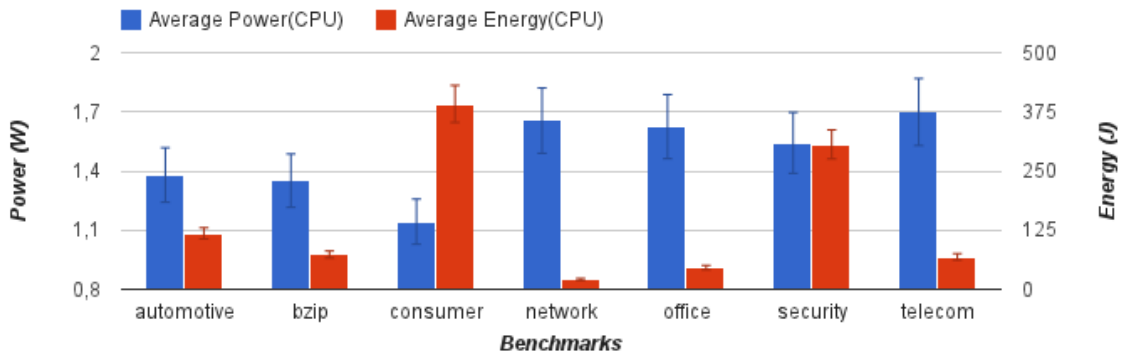


Figure 5.3: **Characteristics of cBench running on ARM Cortex-A15 on the Versatile Express Motherboard with CoreTile TC2 Daughterboard platform.** The figure presents the CPU *Average Power* and *Energy* for each group of benchmarks in the cBench suite while running on the ARM Cortex-A15 processor at 1.2GHz on the Versatile Express Motherboard with CoreTile TC2 Daughterboard development platform.

The graph shows that most of the expectations were generally met. Despite the smaller OS the new platform has lower power usage but increased energy consumption. However it is

surprising to see that the consumer subset of benchmarks use quite a lot more energy compared to the ODROID XU+E. This can be explained by the fact that they do image processing and use large images as inputs, so because of the different (slower) memory structure on the ARM Versatile Express Motherboard with CoreTile TC2 Daughterboard, the CPU ends up waiting for a much longer time for data coming back and forth from memory.

The next tables represent the models built from the collected events, in the same way this was done on the ODROID XU+E. The same events were intentionally collected, so as to compare the systems as objectively as possible.

Table 5.3: **Performance of the single-thread power models for ARM Cortex-A15 on the Versatile Express Motherboard with CoreTile TC2 Daughterboard platform.** The presented data uses three different metrics to illustrate the accuracy of the various models generated from combinations of the four accessible hardware events - CPU *Cycles*, *Instructions*, *Cache References(CR)* and *Cache Misses(CM)* as well as *Cycles-per-Instruction(CPI)* and *Instructions-per-Cycle(IPC)*.

Model Type	Predicted Power Standart Deviation	Average RMS Difference	Average Dynamic Contribution
Full Cycles	4,85%	20,00%	-6,96%
Full Instructions	3,00%	19,81%	-6,81%
Full CPI	3,11%	20,72%	-5,81%
Full IPC	5,99%	19,89%	17,02%
Full CR	9,85%	17,64%	-20,47%
Full CM	11,44%	16,88%	-11,17%
Full I,CR,CM	12,34%	15,57%	-12,00%
Full IPC,CR,CM	12,34%	16,90%	-18,42%
Partial I,CR,CM	16,79%	10,87%	-3,49%

The Cache Misses is again the best fitting event on its own. However this time Instructions is a better regressand than Cycles, CPI and ICP. The best model, consisting of Instructions, Cache References and Cache Misses fits the full model rather poorly at 16%. However it is to note that this platform shows a larger variance in measured power at 21.67% and the full model only predicts a 12.24% standard deviation. It also has quite a large negative Dynamic Contribution at -12%. These differences occur because, even though the two boards have the same CPU architecture, they have different chip topologies and manufacturing processes which influence the power consumption drastically. In the case of the second board the same events do not produce the same results, which means that in order to model a platform correctly an individual model needs to be computed every time. The partial model again has better performance for partly the same reasons as it did for the ODROID XU+E board, but this time the full model has difficulty not just with the consumer set of benchmarks (average 30% difference), but also with automotive, with automotive_susan_c and automotie_susan_e having 56% and 49% prediction error respectively.

The next experiment captures the Ubuntu Developer OS overhead on the SoC by disabling

the Cortex-A7 cluster and focusing on the Cortex-A15. Then the average power was measured by collecting the energy accumulated during a 12 hour sleep and dividing that by the total time. During this time the Cortex-A15 cluster average idle power is measured to be 0.05 W which is negligible. Interestingly the powered-down Cortex-A7 cluster can be seen to having 0.002 W average power, which can be attributed to leakage. Overall, the background energy consumption is very minimal so the choice for a minimal OS for the experiments was justified.

The next experiment shows the average overhead of event collection, measured by running the full cBench set without events and registering the difference in power and energy measurements from the runs with event collection. The CPU shows a 5,62% difference in power consumption, which means event collection still has a small overhead on the ARM Versatile Express Motherboard with CoreTile TC2 Daughterboard and this needs to be factored in when trying to develop a dynamic power optimisation algorithm that uses event collection.

However the overhead is still relatively small and this type of model, base on hardware events is suitable for on-the-fly computation, at least in terms of collecting the variables to plug into the equation (there might be additional costs with calculating the models from the events) as long as the achieved power reduction is greater than 5%.

5.1.2.2 Cortex-A7

Reproducing the experiments on the Cortex-A7, should show the same specificities for the LITTLE while having similar level of the difference between the ARM Versatile Express Motherboard with CoreTile TC2 Daughterboard and ODROID XU+E implementation as observed in the Cortex-A15 results above.

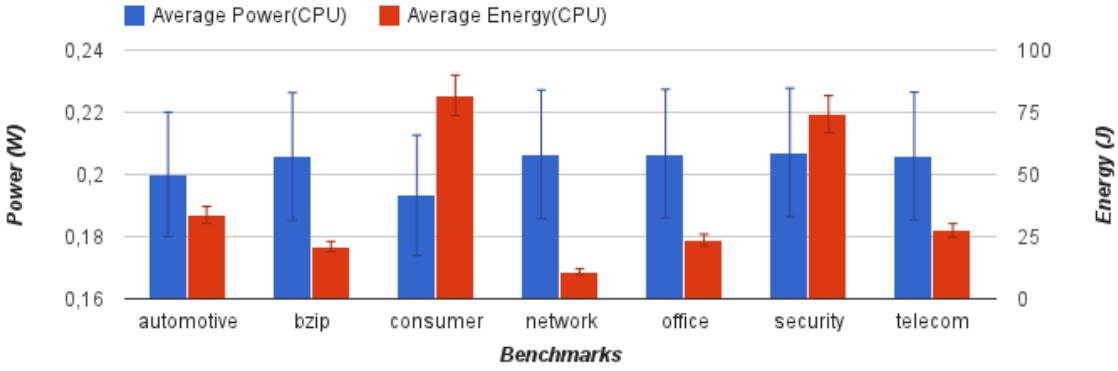


Figure 5.4: **Characteristics of cBench running on ARM Cortex-A7 on the Versatile Express Motherboard with CoreTile TC2 Daughterboard platform.** The figure presents the CPU *Average Power* and *Energy* for each group of benchmarks in the cBench suite while running on the ARM Cortex-A7 processor at 0.175GHz on the Versatile Express Motherboard with CoreTile TC2 Daughterboard development platform.

The ARM Cortex-A7 on the CoreTile TC2 Daughterboard is again significantly more

power-hungry, despite running at an even lower frequency. The following table shows the performance of the computed power models. All of them compared should have a better fit compared to the ones for the ARM Cortex-A15, the same way it was on the ODROID XU+E.

Table 5.4: **Performance of the single-thread power models for ARM Cortex-A7 on the Versatile Express Motherboard with CoreTile TC2 Daughterboard platform.** The presented data uses three different metrics to illustrate the accuracy of the various models generated from combinations of the four accessible hardware events - CPU *Cycles*, *Instructions*, *Cache References(CR)* and *Cache Misses(CM)* as well as *Cycles-per-Instruction(CPI)* and *Instructions-per-Cycle(IPC)*.

Model Type	Predicted Power Standart Deviation	Average RMS Difference	Average Dynamic Contribution
Full Cycles	0,85%	3,80%	-1,60%
Full Instructions	0,47%	3,80%	-1,06%
Full CPI	0,55%	3,90%	-1,91%
Full IPC	1,05%	3,78%	3,88%
Full CR	1,37%	3,65%	-2,60%
Full CM	2,17%	3,52%	-1,83%
Full IPC,CR,CM	2,19%	3,54%	-2,55%
Full CR,CM	2,19%	3,53%	-2,33%
Partial IPC,CR,CM	1,07%	2,43%	-4,42%
Partial CM	0,96%	1,97%	-0,58%

It is very interesting to see that just Cache Misses alone achieves the best fit with only 3.52% difference for the full set and 1.97% for the partial model. However the actual system measured Power Standard Deviation is 5.44% so the model still overestimates the static component and cannot capture all the dynamic contributions. It still has negative coefficient for the event as well as -0.6% dynamic contribution. However it is still reassuring that Cache Misses proves to be an important contributor to the power consumption for both development platforms. Similarly to the Cortex-A15 model the full set best fit model still struggles with consumer benchmarks as well as automotive and bzip2e.

The next set of experiments again deals with the OS and event collection overhead. which should also be quite low, like on the Cortex-A15. The Cortex-A7 cluster average power is estimated to be 0,008(W) and the Cortex-A15 leakage power is measured as 0,003 (W). The OS overhead is minuscule when running on the Cortex-A7 and is quite close to the static Cortex-A15 leakage consumption. The conclusion is that the model greatly overestimates the static power consumption of both CPUs and the dynamic component should be significantly more prevalent. This is due to the poor choice of regressors, so other combinations of events should be explored. Also this might be due to the varied nature of the workload. Maybe a unified power model is not sufficient and separate workload-specific models need to be built. A scheduling algorithm using multiple models needs to have the ability to choose the best one depending on the task at runtime, which will increase the complexity and compute overhead

so this trade-off needs to be evaluated. Next the overhead of event collection is measured as an average of 6.47%, which is close to the Cortex-A15 event collection overheads on the ARM Versatile Express Motherboard with CoreTile TC2 Daughterboard, which means that Cortex-A7 models based on events can also be used in real time as long as the power save is more than 7%.

5.1.2.3 Environmental Effects on Models

The next set of experiments again tried to evaluate the environmental effects on the power consumption of the ARM Versatile Express Motherboard with CoreTile TC2 Daughterboard platform. The board does not have a huge heatsink on top of the SoC, so it was expected to show a greater difference that the results obtained on the ODROID XU+E. The experiment was done without event collection, because they are not needed for just thermal analysis.

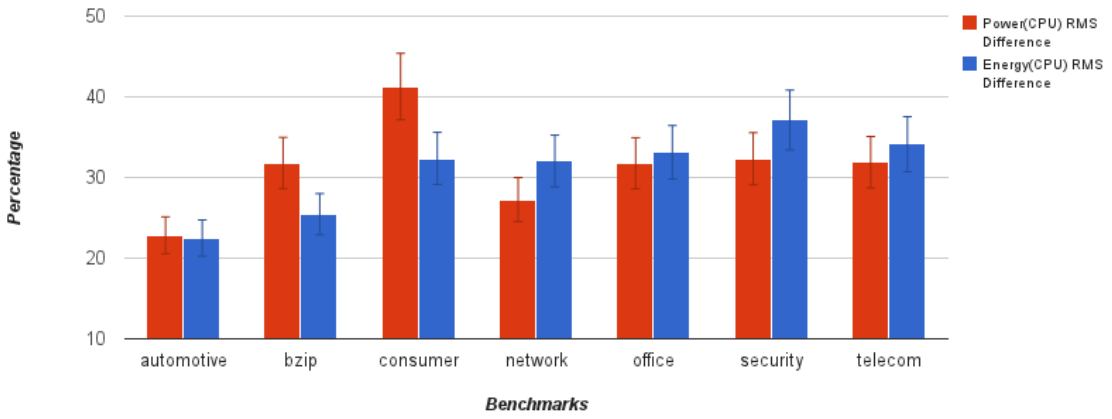


Figure 5.5: **Effects of the on-board cooling unit on the ARM Versatile Express Motherboard with CoreTile TC2 Daughterboard platform for ARM Cortex-A15.** The data presents the *Root Mean Square(RMS) Difference* in CPU *Power* and *Energy* between switching on and off the on-board cooling unit for each group of benchmarks in the cBench suite while running on the ARM Cortex-A15 processor on the Versatile Express Motherboard with CoreTile TC2 Daughterboard development platform.

It is evident that there is quite a big difference. The CPU temperature under normal conditions is around 20C, while with the fan off it reached a steady 40C, almost double and this affected the power usage of the system greatly. Even though this second platform has an older manufacturing technology, this trend should also be observable in commercially available SoCs (which have no fans like dev boards do), which means that temperature should definitely be considered when building model which can potentially be used in consumer systems.

The next experiment has the similar idea, but this time a single benchmark (consumer_jpeg_c)

was ran continuously for 50 runs then turning the fan off and running for another 50 runs as described earlier. The idea is to capture a trend.

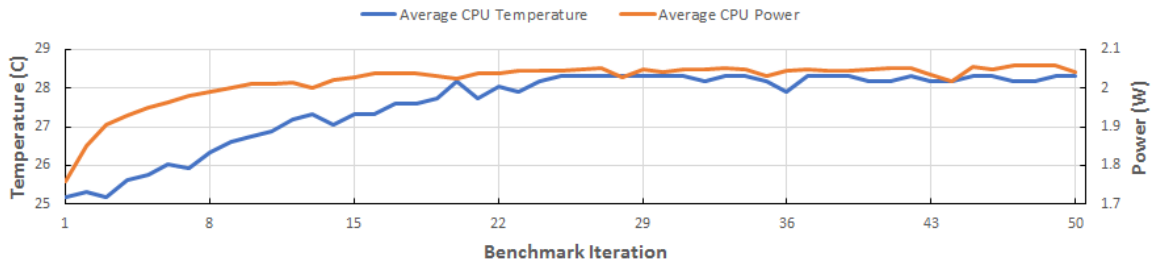


Figure 5.6: **Effects of increasing temperature over time on the ARM Versatile Express Motherboard with CoreTile TC2 Daughterboard platform for ARM Cortex-A15.** The data presents the CPU *Average Temperature* and *Power* while repeatedly running the *consumer_jpeg_c* benchmark from the cBench suite on the ARM Cortex-A15 processor on the Versatile Express Motherboard with CoreTile TC2 Daughterboard development platform with on-board cooling switched on.

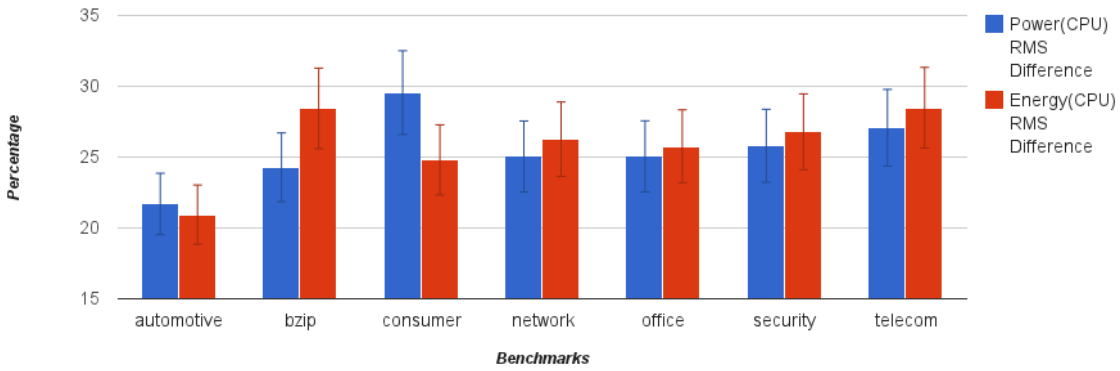


Figure 5.7: **Effects of the on-board cooling unit on the ARM Versatile Express Motherboard with CoreTile TC2 Daughterboard platform for ARM Cortex-A7.** The data presents the *Root Mean Square(RMS) Difference* in CPU *Power* and *Energy* between switching on and off the on-board cooling unit for each group of benchmarks in the cBench suite while running on the ARM Cortex-A7 processor on the Versatile Express Motherboard with CoreTile TC2 Daughterboard development platform.

It is evident that the SoC got hot too quickly by the immediate jump in temperature. It stops rising around 28C and does not get as hot as the full cBench run. This experiment should be repeated with a more CPU bound benchmark, which does not allow the CPU to cool down while waiting for data from memory. The current measurements still showcase that temperature plays a huge role in power consumption, 32% increase in temperature results in 30% increase in power consumption which indicates a high correlation. The first experiment

was also repeated for the Cortex-A7 and the following plot shows the relative difference in power consumption.

Again a significant increase in energy consumption is seen, however the SoC did not go over 38C, so it was a bit cooler that the ARM Cortex-A15 run. This is reflected in the percentage differences, with them being lower than ones measured on the Cortex-A15. The same reasoning that the Cortex-A7 can use the Cortex-A15 area on the SoC to dissipate more heat can be applied to justify the lower relative increase in power consumption.

5.1.3 Platform Comparison between ARM Versatile Express Motherboard with CoreTile TC2 Daughterboard and ODROID XU+E

The final results that are presented are from comparing the events between the ODROID XU+E and ARM Versatile Express Motherboard with CoreTile TC2 Daughterboard platforms. This is done in order to see how, in spite of the fact that the CPUs have the same architecture and run the same workload, the different SoC configurations and manufacturing technologies affect power consumption. A similar number of architecturally retired instructions on both platform is expected, with a difference in the other events due to differences in the cache sizes and memory speed for the different SoC topologies.

5.1.3.1 Cortex-A15

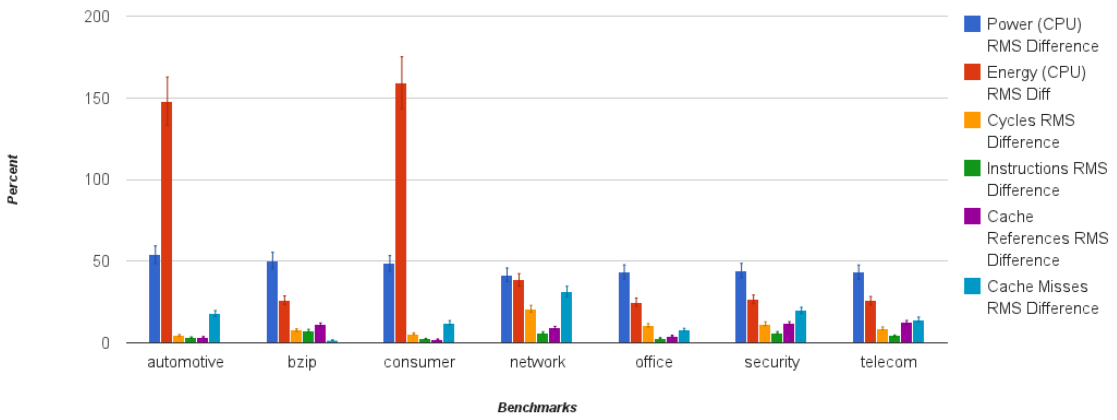


Figure 5.8: **Difference between single-thread workload executions on the ARM Versatile Express Motherboard with CoreTile TC2 Daughterboard platform and the ODROID XU+E board for ARM Cortex-A15.** The data presented is the Average *Root Mean Square(RMS) Difference* of the *CPU Power*, *Energy*, *Cycles*, *Instructions*, *Cache References* and *Cache Misses* between the cBench workload executions for ARM Cortex-A15 processor at maximum available frequency on both development platforms.

Observing the Cortex-A15 data shows that even the instruction count is different, which is probably due to a slight difference in the compilers for both systems, since the pre-installed GCC compiler was used for the respective OSes installed. Other than that the second platform, the ARM Versatile Express Motherboard with CoreTile TC2 Daughterboard, has much higher power consumption and overall energy usage, which is expected and seen from previous results. There are relatively minor differences in the events, but a big difference in energy consumption in automotive and consumer, which are benchmark sets that have big data file inputs, so this is mainly due to difference in memory speed.

5.1.3.2 Cortex-A7

Figure 5.9 shows a comparison between the Cortex-A7s on the two boards as well as the Cortex-A15 and Cortex-A7 on TC2, the same way it was done for the ODROID XU+E.

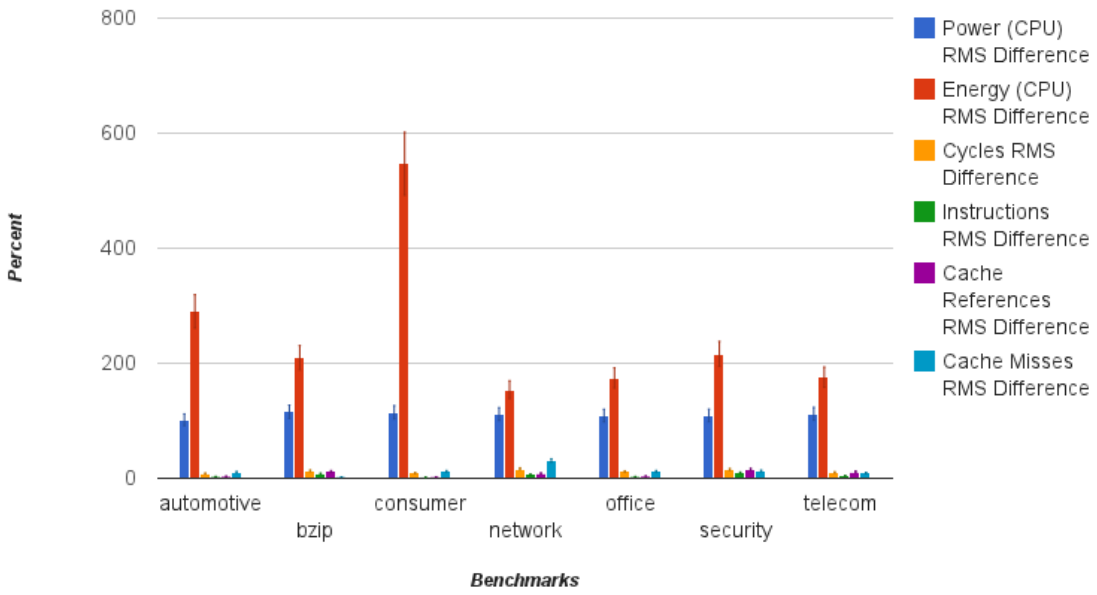


Figure 5.9: **Difference between single-thread workload executions on the ARM Versatile Express Motherboard with CoreTile TC2 Daughterboard platform and the ODROID XU+E board for ARM Cortex-A7.** The data presented is the Average *Root Mean Square(RMS) Difference* of the *CPU Power*, *Energy*, *Cycles*, *Instructions*, *Cache References* and *Cache Misses* between the cBench workload executions for ARM Cortex-A7 processor at minimum available frequency on both development platforms.

The data shows similar differences as the ones between the ARM Cortex-A15s. However total system energy usage is much greater, because the ARM Versatile Express Motherboard

with CoreTile TC2 Daughterboard Cortex-A7 ran at 1/3 of the frequency of the ODROID XU+E Cortex-A7 and since the Cortex-A7 has a minimal voltage level to operate this could be due to a particularly unfavourable DVFS configuration. There is still a much larger difference in power consumption than the difference in frequency so this still indicates how manufacturing process and SoC topology affect the power consumption of CPU even with the same architecture.

5.1.3.3 Cortex-A15 against Cortex-A7

Figures 5.10 and 5.11 show the difference between the Cortex-A7 and Cortex-A15 events on the ODROID XU+E and the ARM Versatile Express Motherboard with CoreTile TC2 Daughterboard, just to highlight the heterogeneous side of the systems. The Cortex-A7 has a much smaller cache size so resulting in more Cache Misses and more memory accesses. In terms of instructions there should be no difference since the compiled workload is unchanged and the same number of them should be architecturally retired by both CPUs, since they both run the same binaries. Due to that the same number of Cache References are also expected. The CPU cycles are also expected to greatly vary between the two CPUs since the ARM Cortex-A15 is superscalar out-of-order so it should complete CPU bound tasks with significantly less cycles, compared to the in-order ARM Cortex-A7.

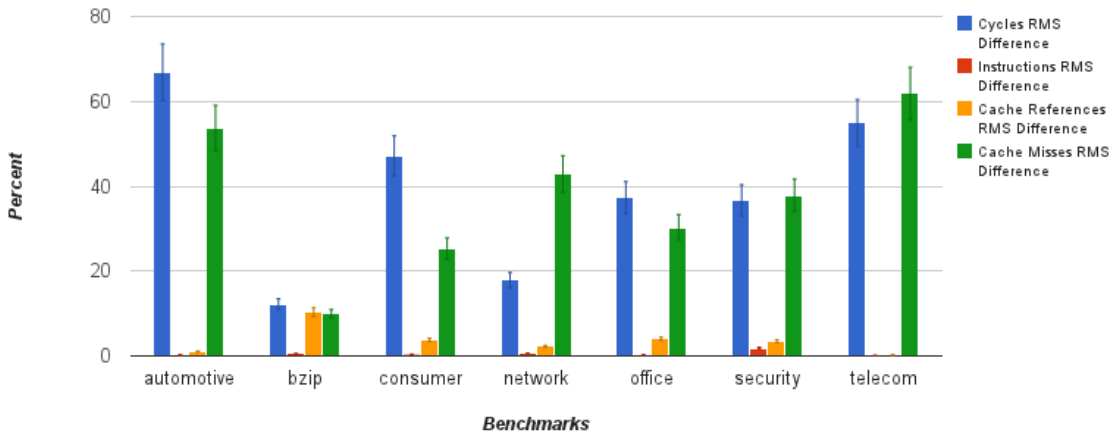


Figure 5.10: **Difference between single-thread workload executions on ARM Cortex-A15 and ARM Cortex-A7 on the ODROID XU+E board.** The data presented is the Average *Root Mean Square(RMS) Difference* of the CPU *Cycles*, *Instructions*, *Cache References* and *Cache Misses* between the cBench workload executions on ARM Cortex-A15 processor at 1.6GHz and ARM Cortex-A7 processor at 0.5GHz on the ODROID XU+E development platform.

The presented data highlights the different focus of each CPU and should not surprise anyone. These differences show the opportunities for optimising Core task allocation, since

the Cortex-A7 is not just a throttled down Cortex-A15, but also has specific strengths on its own and can be used as a valuable resource for power management.

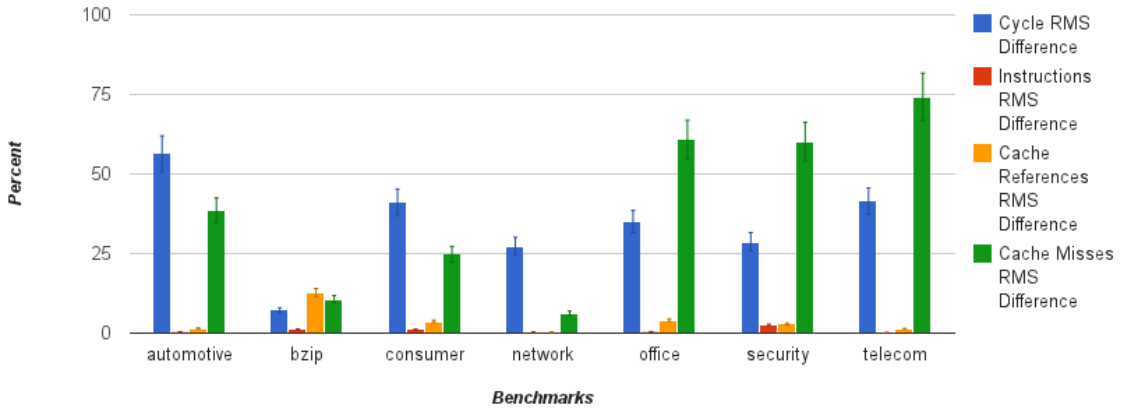


Figure 5.11: **Difference between single-thread workload executions on ARM Cortex-A15 and ARM Cortex-A7 on the ARM Versatile Express Motherboard with CoreTile TC2 Daughterboard platform.** The data presented is the Average *Root Mean Square(RMS) Difference* of the CPU *Cycles*, *Instructions*, *Cache References* and *Cache Misses* between the cBench workload executions on ARM Cortex-A15 processor at 1.2GHz and ARM Cortex-A7 processor at 0.175GHz on the ARM Versatile Express Motherboard with CoreTile TC2 Daughterboard development platform.

For the ARM Versatile Express Motherboard with CoreTile TC2 Daughterboard, a similar overview to the ODROID XU+E one is presented, where instructions and Cache References remain similar with greater differences between cache misses and cycles. It is evident that that even though both development boards have very different energy consumptions at least in terms of relative trends the two behave quite similarly, meaning that it possible to deduce generalized power consumption characteristics just by analysing the CPU architecture and at least the general trends for the *big.LITTLE* architecture remain the same regardless of implementation. This means it is possible to predict how much relative power can be saved on a given system if the architecture is know, but in order to estimate the absolute power consumption SoC topology, manufacturing process and environmental factors needs to be taken into consideration.

5.2 Extending the model for the ODROID-XU3 platform

After the initial results on the ODROID XU+E and the ARM Versatile Express Motherboard with CoreTile TC2 Daughterboard development platforms, an ODROID XU3 board was purchased for development during this research. The key features are outlined in chapter 3.3.

The new platform had a working PMU and on-board power sensors. This provided a much quicker and more stable experiment set-up so the model was quickly extended and refined. The methodology was quickly adapted to be able to collect system data and analyse the entire frequency range of the system. This enables generation of complete models that can predict average power at any energy level, instead of at the highest and lowest points like the initial results. Since the exploration space was much broader, more work was spent on refining the methodology and tuning the model in order to achieve low prediction error. The models are built from obtained measurements from the board using the OLS algorithm. Correlation analysis is done for events on the train set and only the ones that best model power (minimal error) are used in the final model. The model that have been developed consists of 3 distinct components

1. Physical, expressed in equation 5.1, which has physically controlled regressands such as CPU voltage, CPU frequency and CPU temperature. CPU frequency is proven to be highly correlated to power consumption [44] however it is claimed, that voltage and temperature also play a crucial role in determining dynamic power and are not dependent on frequency. It is demonstrated that adding them increases model accuracy, compared to just using frequency.

2. PMU events, expressed in equation 5.2, which has events available for both Cortex-A15 and Cortex-A7. The PMU events common to both CPUs are used, instead of ones available for only the ARM Cortex-A15. This is necessary if the power model is to be extended in the future to help make scheduling decision, since it needs to have the same environments as inputs to both models in order to decide where the tasks should be migrated. Ten common PMU events available to both CPU core types are considered and after mathematical analysis the top 4 most correlated to power are chosen in addition to CPU cycle count, since that is the maximum number of events the Cortex-A7 PMU can read concurrently without multiplexing. CPU_CYCLES, L1D_CACHE_ACCESS, L1I_CACHE_ACCESS, INST_RETIRED, DATA_MEM_ACCESS, L2D_CACHE_ACCESS, EXCEPTION_TAKEN, BRANCH_PRED, BRANCH_MISPRED, BUS_ACCESS were examined. The list of PMU events is available in the Technical Reference Manuals available for both CPUs [74], [77].

3. CPU state, expressed in equation 5.3, uses information about time spent in different relevant CPU states. It is inspired by the insight of Dr Nunez-Yanez in [49], the academic supervisor of this project. The model expanding on the work done by Walker et al. [61] by not just using CPU Idle but also other states. This work concludes that just using CPU Idle does not capture the dynamic CPU power accurately since it was observed that during workload executions the CPU spent very little time in that state.

$$P(W) = \alpha_0 + \alpha_1 \times CPU\_voltage + \alpha_2 \times CPU\_frequency + \alpha_3 \times CPU\_temperature \tag{5.1}$$

$$\begin{aligned} P(W) = {} & \alpha_0 + \alpha_1 \times CPU\_CYCLES + \alpha_2 \times L1D\_CACHE\_ACCESS \\ & + \alpha_3 \times L1I\_CACHE\_ACCESS + \alpha_4 \times INST\_RETIRED \\ & + \alpha_5 \times DATA\_MEM\_ACCESS \end{aligned} \tag{5.2}$$

$$\begin{aligned} P(W) = {} & \alpha_0 + \alpha_1 \times CPU\_user\_state + \alpha_2 \times CPU\_system\_state + \alpha_3 \times CPU\_idle\_state \\ & + \alpha_4 \times CPU\_IO\_wait\_state + \alpha_5 \times CPU\_IRQ\_state \\ & + \alpha_6 \times CPU\_software\_IRQ\_state \end{aligned} \tag{5.3}$$

Then the events from the 3 component models listed above are combined to develop a Physical + PMU events (called P2) model, shown in equation 5.4 and a model using all 3 groups of events Physical + PMU events + CPU state (called P2S) model, shown in equation 5.5. The P2 and P2S model achieve the high baseline accuracy resulting from the use of the physical events with the ability to follow the power usage during program execution, thanks to the PMU events and CPU state information.

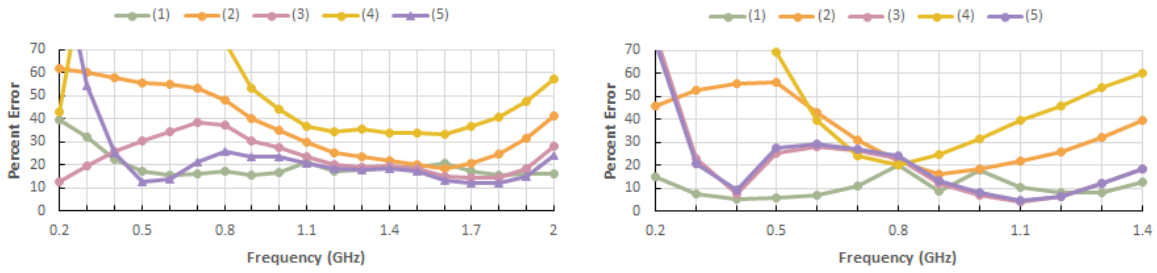
$$\begin{aligned} P(W) = {} & \alpha_0 + \alpha_1 \times CPU\_voltage + \alpha_2 \times CPU\_frequency + \alpha_3 \times CPU\_temperature \\ & + \alpha_4 \times CPU\_CYCLES + \alpha_5 \times L1D\_CACHE\_ACCESS \\ & + \alpha_6 \times L1I\_CACHE\_ACCESS + \alpha_7 \times INST\_RETIRED \\ & + \alpha_8 \times DATA\_MEM\_ACCESS \end{aligned} \tag{5.4}$$

$$\begin{aligned} P(W) = {} & \alpha_0 + \alpha_1 \times CPU\_voltage + \alpha_2 \times CPU\_frequency + \alpha_3 \times CPU\_temperature \\ & + \alpha_4 \times CPU\_CYCLES + \alpha_5 \times L1D\_CACHE\_ACCESS \\ & + \alpha_6 \times L1I\_CACHE\_ACCESS + \alpha_7 \times INST\_RETIRED \\ & + \alpha_8 \times DATA\_MEM\_ACCESS + \alpha_9 \times CPU\_user\_state \\ & + \alpha_{10} \times CPU\_system\_state + \alpha_{11} \times CPU\_idle\_state \\ & + \alpha_{12} \times CPU\_IO\_wait\_state + \alpha_{13} \times CPU\_IRQ\_state \\ & + \alpha_{14} \times CPU\_software\_IRQ\_state \end{aligned} \tag{5.5}$$

In this section the model results are presented in graphical form. 2 types of models are generated – unified single-equation model trained over points on the entire frequency range of the CPU and models trained on a per-frequency basis. It is demonstrated that the per-frequency models perform significantly better than the full frequency range models, which to the researcher's knowledge is an observation not previously documented on big.LITTLE, since other work has focused largely on unified models.

5.2.1 Full frequency range models

First the results using the models from the custom methodology are highlighted and show the performance of the individual component models as well as the grouped P2 and P2S models. All these models are trained on the train set using all points available from all frequencies and then tested on all the points of the test set. The percent error between the predicted and measured power, using the on-board sensors, is calculated and the averages for each frequency level are presented in Figure 5.12 and Table 5.5. Observations show that the resulting complex P2S model is not always the best performer, instead simpler models just using physical characteristics like CPU frequency, voltage and temperature tend to have the lowest average error, so in general the CPU power consumption is mostly influenced by the physical information and this is not surprising. Therefore introducing PMU event information and CPU state information does not improve performance as much as expected. This is due to the fact that both PMU events and CPU state information in general have quite high variance for different frequencies (especially low and high frequencies, where greater increase in error compared to the mid-range frequencies) is observed, so instead of successfully predicting the finer power changes during workload execution they introduce larger margin of error. This is due to the fact that the data sampling rate is time based and the number of samples and the event values at each sample point would vary from frequency to frequency. For example each time slice would capture a different number of CPU cycles at different frequency levels. Another thing to note is that the models for the ARM Cortex-A7 are expected to perform a lot better, since it is a much simple CPU in terms of architecture, however this does not seem to be the case.



(a) ARM Cortex-A15 Models

(b) ARM Cortex-A7 Models

Figure 5.12: **Comparison between the generated full frequency range single-thread models for both processor types on the ODROID XU3 board.** The models are generated and validated by the custom methodology using the cBench workload for the two processor types on the ODROID XU3 development platform with the eMMC card as OS driver. The *Average Percent Error* at each available CPU frequency for the models is presented on the graph. The model references corresponding to the legend are given in Table 5.5.

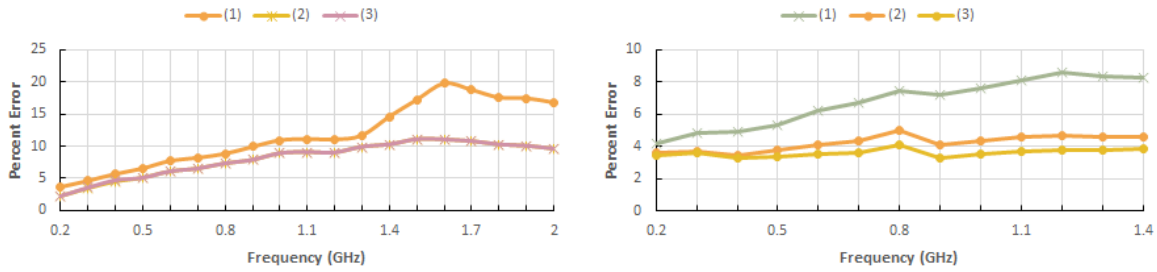
The results shown on Figure 5.12 and Table 5.5 indicate that the best full frequency model

for the Cortex-A15 and the Cortex-A7 is Physical with an average error of 19.57% and 10.46% respectively. It is evident that the accuracy of the models is affected by the CPU frequency and most models achieve the highest accuracy on the mid-range of frequency levels, 1-1.6 GHz for the Cortex-A15 and 0.6-1.2 GHz for the Cortex-A7. Ideally, the model error should be a flat line close to zero so accuracy is high and invariable with frequency. The fact that new frequencies involve different voltages means that it is challenging for the model to maintain accuracy across all the whole frequency range.

Table 5.5: **Model reference and total average error for the data in Figure 5.12.** The model *Reference* is presented for each *Model Code* from the legend. *Total Average Error* values are obtained for each model on the two processor types (when available) by using their prediction error values across all frequency levels.

Model Code	Reference	Total Average Error	
		ARM Cortex-A15	ARM Cortex-A7
(1)	Physical	19.57%	10.46%
(2)	PMU Events	38.02%	35.22%
(3)	P2	23.5%	20.8%
(4)	CPU State	59.27%	59.4%
(5)	P2S	25.3%	20.99%

5.2.2 Per-frequency models



(a) ARM Cortex-A15 Models (b) ARM Cortex-A7 Models

Figure 5.13: **Comparison between the generated per-frequency level single-thread Physical, P2 and P2S models for both processor types on the ODROID XU3 board.** The models are generated and validated by the custom methodology using the cBench workload for the two processor types on the ODROID XU3 development platform with the eMMC card as OS driver. The *Average Percent Error* at each available CPU frequency for the models is presented on the graph. The model references corresponding to the legend are given in Table 5.6.

Based on previous observations on the variability of PMU events and CPU state information between frequency levels it was decided to try and calculate the models on a per-frequency basis. It was decided to not change any of the equations, despite the frequency and voltage

level remaining the same for the training, since all the data point at one voltage/frequency level are used. This is because physical information gives a very good prediction for average power. Figure 5.13 and Table 5.6 show that the analysis is justified by the results. The data indicates a large reduction for the average model error and that adding CPU state and PMU event predictors improve accuracy significantly compared to just using the Physical model. Overall an average of 8% error for the ARM Cortex-A15 and 5% error for the per frequency level P2S power model is reported, which is on the target goal for the Cortex-A7. The fact that the model error is not a straight line means that there is some relationship the model does not accurately cover. It could be possible to add mechanistic information (e.g. pipeline length, microarchitecture details) to try to improve accuracy further however this could translate in an excessively complex model, not suitable for real-time scheduling.

Table 5.6: **Model reference and total average error for the data in Figure 5.13.** The model *Reference* is presented for each *Model Code* from the legend. *Total Average Error* values are obtained for each model on the two processor types (when available) by using their prediction error values across all frequency levels.

Model Code	Reference	Total Average Error	
		ARM Cortex-A15	ARM Cortex-A7
(1)	Physical	11.68%	6.74%
(2)	P2	8.06%	4.2%
(3)	P2S	8.07%	3.58%

5.3 Comparison to related work

The developed composite models P2 and P2S are compared to relevant published work. Equation 5.6 represent the single-thread interpretation of the model presented by Takouna et al. [44] from the University of Potsdam. It is referred to as UoP model for short.

$$P(W) = \alpha_0 + \alpha_1 \times CPU\_frequency + \alpha_2 \times (CPU\_frequency)^2 \tag{5.6}$$

Walker et al. [61] from University of Southampton present 2 models built on *big.LITTLE* using CPU idle time, presented in equations 5.7 and 5.8. Equation 5.8 has added CPU frequency related regressands to help capture the entire frequency range they report around 10% accuracy on the Cortex-A15 and Cortex-A7. They are referred to as UoS CPU Idle model and UoS full model for short.

$$P(W) = \alpha_0 + \alpha_1 \times CPU\_idle\_state + \alpha_2 \times (CPU\_idle\_state)^2 \tag{5.7}$$

$$\begin{aligned} P(W) = {} & \alpha_0 + \alpha_1 \times CPU\_idle\_state + \alpha_2 \times CPU\_frequency \\ & + \alpha_3 \times (CPU\_idle\_state \times CPU\_frequency) \\ & + \alpha_4 \times (CPU\_idle\_state)^2 + \alpha_5 \times ((CPU\_idle\_state)^2 \times CPU\_frequency) \end{aligned} \tag{5.8}$$

Pricopi et al. [50] from Cambridge Silicon Radio present a PMU based model built for the ARM Cortex-A15 running at 1Ghz. This model is referred to as CSR for short and the events they use are shown in equation 5.9. They report around 2.6% average error.

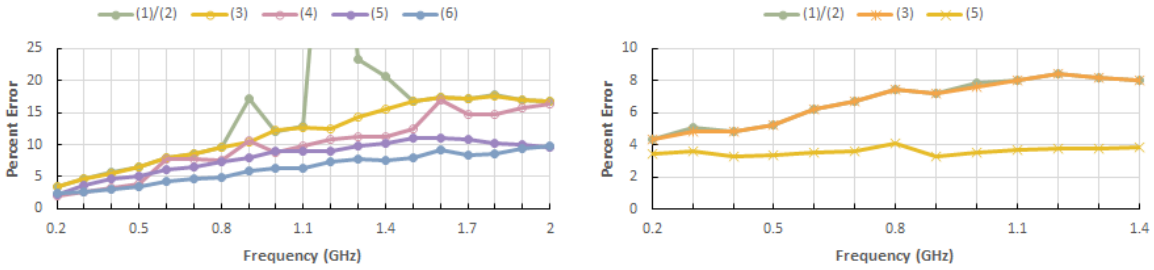
$$\begin{aligned} P(W) = {} & \alpha_0 + \alpha_1 \times IPC + \alpha_2 \times \frac{INST\_SPEC\_EXEC\_INT}{INST\_SPEC\_EXEC} \\ & + \alpha_3 \times \frac{INST\_SPEC\_EXEC\_VFP}{INST\_SPEC\_EXEC} + \alpha_4 \times \frac{L1D\_CACHE\_ACCESS}{INST\_SPEC\_EXEC} \\ & + \alpha_5 \times \frac{L2D\_CACHE\_ACCESS}{INST\_SPEC\_EXEC} + \alpha_6 \times \frac{L2D\_CACHE\_REFILL}{INST\_SPEC\_EXEC} \end{aligned} \tag{5.9}$$

Finally equation 5.10 shows the CSR model extended with physical and CPU state information as done in the custom approach. The CSR model accesses more PMU events since it is designed exclusively for the Cortex-A15 while P2 and P2S work for both Cortex-A7 and Cortex-A15. This model will serve to illustrate how adding physical and CPU state information can greatly improve performance.

$$\begin{aligned} P(W) = {} & \alpha_0 + \alpha_1 \times CPU\_voltage + \alpha_2 \times CPU\_frequency + \alpha_3 \times CPU\_temperature \\ & + \alpha_4 \times IPC + \alpha_5 \times \frac{INST\_SPEC\_EXEC\_INT}{INST\_SPEC\_EXEC} + \alpha_6 \times \frac{INST\_SPEC\_EXEC\_VFP}{INST\_SPEC\_EXEC} \\ & + \alpha_7 \times \frac{L1D\_CACHE\_ACCESS}{INST\_SPEC\_EXEC} + \alpha_8 \times \frac{L2D\_CACHE\_ACCESS}{INST\_SPEC\_EXEC} \\ & + \alpha_9 \times \frac{L2D\_CACHE\_REFILL}{INST\_SPEC\_EXEC} + \alpha_{10} \times CPU\_user\_state \\ & + \alpha_{11} \times CPU\_system\_state + \alpha_{12} \times CPU\_idle\_state \\ & + \alpha_{13} \times CPU\_IO\_wait\_state + \alpha_{14} \times CPU\_IRQ\_state \\ & + \alpha_{15} \times CPU\_software\_IRQ\_state \end{aligned} \tag{5.10}$$

Figure 5.14 and Table 5.7 show them compared to the best performing per-frequency model P2S for the Cortex-A15 and Cortex-A7 respectively. In both cases the P2S model performs a lot better with the exception of the Updated CSR model enhanced by us. This is to be expected since the Updated CSR is a more complex model with larger, more specialized list of PMU events that are used in the model. The similarity between the "UoS full" and "UoP" models highlights that using CPU idle as a regressand does not significantly improve performance

over the physical regressands that the UoP model uses. However including the other CPU states as regressands improves accuracy albeit only marginally and just on the Cortex-A7. Notice that Figure 5.14 does not include results for the CSR models because a model has not been developed for the Cortex-A7 in their approach. This shows that this work is effective in producing models with competitive accuracy to the state-of-the art in current research.



(a) ARM Cortex-A15 Models (b) ARM Cortex-A7 Models

Figure 5.14: **Comparison between the generated per-frequency level single-thread P2S model and other published work for both processor types on the ODROID XU3 board.** The models are generated and validated by the custom methodology using the cBench workload for the two processor types on the ODROID XU3 development platform with the eMMC card as OS driver. The *Average Percent Error* at each available CPU frequency for the models is presented on the graph. The model references corresponding to the legend are given in Table 5.7.

Table 5.7: **Model reference and total average error for the data in Figure 5.14.** The model *Reference* is presented for each *Model Code* from the legend. *Total Average Error* values are obtained for each model on the two processor types (when available) by using their prediction error values across all frequency levels.

Model Code	Reference	Total Average Error	
		ARM Cortex-A15	ARM Cortex-A7
(1)	UoS CPU Idle	15.65%	6.72%
(2)	UoS Full	15.65%	6.72%
(3)	UoP	11.89%	6.69%
(4)	CSR	9.89%	-
(5)	P2S	8.07%	3.58%
(6)	CSR Updated	6.32%	-

5.4 Evaluating model reproducibility

The next stage in this research was to evaluate the P2S per-frequency model, shown in section 5.2.2 on other systems. Observations on the ODROID XU+E board highlight that the hardware exhibits a noticeable energy usage variation with an average of 7.5% power difference and up

to 25% energy difference for some parts of cBench. The aim was to evaluate if the ODROID XU3 platform had such a big variance and if the model could be successfully tuned to overcome this. For this purpose a second ODROID XU3 development board was purchased. The initial results showed a significant variation between the two when running cBench, despite using the same operating system and memory card. After some investigation it was decided to use a command line tool for linux called cset [91]. It provides better program execution control by setting up "shields", which isolate CPU cores and allow for fine-grain task assignment. An example of how to use cset to isolate CPU cores 1-3 is given in 5.15

```
[zuul:cpuset-trunk]# cset shield -c 1-3
cset: --> activating shielding:
cset: moving 176 tasks from root into system cpuset...
[==================================================]%
cset: "system" cpuset of CPUSPEC(0) with 176 tasks running
cset: "user" cpuset of CPUSPEC(1-3) with 0 tasks running
```

Figure 5.15: **Example of system resource allocation using cset.** The tool is used to set up a protected environment on CPU(1-3) with all moveable background and OS tasks transferred to CPU(0).

With the help of the tool, the CPU core running the workload could be completely isolated from any external threads and thus reduce OS and data collection experiment overhead. This greatly reduces platform variability compared to using taskset to assign the workload to the desired CPU core. Table 5.8 shows that the power difference is halved and more notably the CPU_CYCLES PMU event variation has been significantly reduced, indicating that the OS interference has been indeed minimised during the experiment.

Table 5.8: **Difference between resource allocation using taskset and cset for both processor types on the ODROID XU3 board.** The data presented is the *Average CPU Power* and *CPU_CYCLES* hardware event *Difference* between cBench workload executions for both processor types across all frequency levels on the ODROID XU3 development platform, with the eMMC card as OS driver.

ARM Core Type	Comparison Metric	taskset	cset
Cortex-A15	Average Power Difference	20.51%	9.39%
	CPU_CYCLES Difference	38.49%	1.37%
Cortex-A7	Average Power Difference	18.87%	13.41%
	CPU_CYCLES Difference	6.74%	1.04%

After this crucial step, the next goal was to improve the P2 model by reducing event multicollinearity, which consists of two or more of the events having a relationship between them. This results in the model being very sensitive in small variations in the events, thus reducing stability and accuracy [104], [105], [106]. Reducing the event cross-correlation is

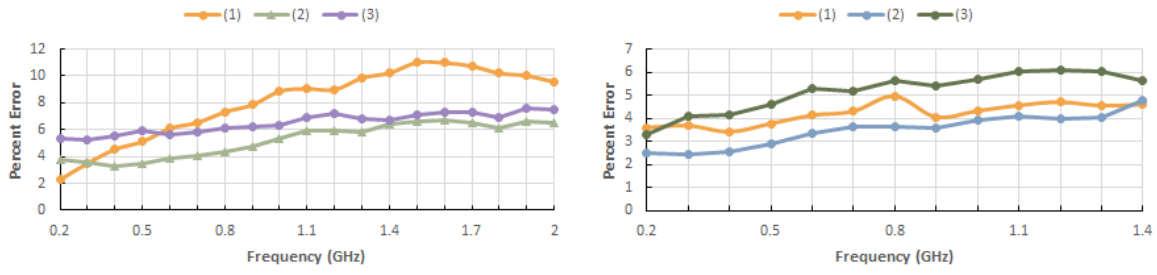
crucial in order to ensure the developed models have a consistent performance when tested on multiple platforms. For this purpose two groups of events from the available common events for the Cortex-A15 and Cortex-A7 have been collected.

Event List 1 (EvL1 for short) is used in conjunction with the physical model from section 5.2.2 to form P2Ev1 as seen in equation 5.11. The events are mainly cache related. On the other hand Event List 2 (EvL2 for short) predominantly consists of events that handle irregular execution - branches and exceptions. The full P2EvL2 model is shown in equation 5.12.

$$\begin{aligned} P(W) = {} & \alpha_0 + \alpha_1 \times CPU\_voltage + \alpha_2 \times CPU\_frequency + \alpha_3 \times CPU\_temperature \\ & + \alpha_4 \times CPU\_CYCLES + \alpha_5 \times L1D\_CACHE\_ACCESS \\ & + \alpha_6 \times L1I\_CACHE\_ACCESS + \alpha_7 \times L2D\_CACHE\_ACCESS \\ & + \alpha_8 \times BUS\_ACCESS \end{aligned} \tag{5.11}$$

$$\begin{aligned} P(W) = {} & \alpha_0 + \alpha_1 \times CPU\_voltage + \alpha_2 \times CPU\_frequency + \alpha_3 \times CPU\_temperature \\ & + \alpha_4 \times CPU\_CYCLES + \alpha_5 \times INST\_RETIRED \\ & + \alpha_6 \times EXCEPTION\_TAKEN + \alpha_7 \times BRANCH\_PRED \\ & + \alpha_8 \times BRANCH\_MISPRED \end{aligned} \tag{5.12}$$

The new models are evaluated on the ODROID XU3 using the methodology, now updated to use cset instead of taskset. Figure 5.16 and Table 5.9 show that significantly improvements on the original P2 model have been made with the P2EvL1 model achieving the lowest percent relative error on both the ARM Cortex-A15 and ARM Cortex-A7.



(a) ARM Cortex-A15 Models (b) ARM Cortex-A7 Models

Figure 5.16: **Comparison between the generated per-frequency level single-thread P2, P2EvL1 and P2EvL2 models for both processor types on the ODROID XU3 board.** The models are generated and validated by the custom methodology using the cBench workload for the two processor types on the ODROID XU3 development platform with the eMMC card as OS driver. The *Average Percent Error* at each available CPU frequency for the models is presented on the graph. The model references corresponding to the legend are given in Table 5.9.

Table 5.9: **Model reference and total average error for the data in Figure 5.16.** The model *Reference* is presented for each *Model Code* from the legend. *Total Average Error* values are obtained for each model on the two processor types (when available) by using their prediction error values across all frequency levels.

Model Code	Reference	Total Average Error	
		ARM Cortex-A15	ARM Cortex-A7
(1)	P2	8.06%	4.20%
(2)	P2EvL1	5.25%	3.49%
(3)	P2EvL2	6.50%	5.17%

The final events list and the model are shown in equation 5.13 and the correlation factors are shown in Table 5.10. It is evident that the model uses events from both sets, with only one cache event. It was also decided to only use CPU temperature as the component of the physical part of the model, since both frequency and voltage contribute static values to the per-frequency model and can be captured by the constant, since it represents the static component of the modelled power.

$$\begin{aligned} P(W) = {} & \alpha_0 + \alpha_1 \times CPU\_temperature \\ & + \alpha_2 \times CPU\_CYCLES + \alpha_3 \times L2D\_CACHE\_ACCESS \\ & + \alpha_4 \times BUS\_ACCESS + \alpha_5 \times EXCEPTION\_TAKEN \\ & + \alpha_6 \times BRANCH\_MISPRED \end{aligned} \tag{5.13}$$

Table 5.10: **Cross-correlation values between the events selected from *P2EvL1* and *P2EvL2*.** The data presented is the cross-correlation (*Corr.*) between the selected events from the *P2EvL1* and *P2EvL2* events lists. The events are represented by their consecutive number (*1* to *5*) in their lists. The cross-correlation coefficient range is between *-1* and *1*, where both end points represent high correlation.

ARM Core Type	Corr.[1;2]	Corr.[1;3]	Corr.[2;3]	Corr.[1;4]	Corr.[1;5]	Corr.[4;5]
Cortex A-15	-0.05	0.29	0.09	0.13	0.19	0.07
Cortex A-7	-0.05	-0.04	0.25	0.08	0.150	0.1

Table 5.11 shows error of the individual events of the model, when used on their own with CPU temperature to predict power . Ev1, which corresponds to CPU_CYCLES is the best performing solo event, which is to be expected. Overall the model is expected to perform much better than the solo CPU_CYCLES model.

After the preparation several of the ODROID XU3 development were obtained from the project's industrial sponsor ARM, in order to validate the model. Each board ran the workload 10 times and the data from the PMU and the on-board sensors was successfully collected and analysed. Figure 5.17 represents the experiment set-up. Each board used the same model mSD

Table 5.11: **Average total individual model error for each of the five selected events.** Each event from the low cross-correlation list was used to generate and individual per-frequency level power model by the custom methodology, using the cBench workload for the two processor types on the ODROID XU3 development platform with the eMMC card as OS driver. The data presented is the *Total Average Error* for each model/event across the entire available CPU frequency range.

ARM Core Type	Event 1	Event 2	Event 3	Event 4	Event 5
Cortex A-15	7.67%	16.43%	16.62%	17.1%	14.89%
Cortex A-7	5.59%	10.79%	10.69%	11.83%	10.3%

card, burned with the same OS image. The conditions were kept the same as much as possible so any environmental factors would not affect the results.

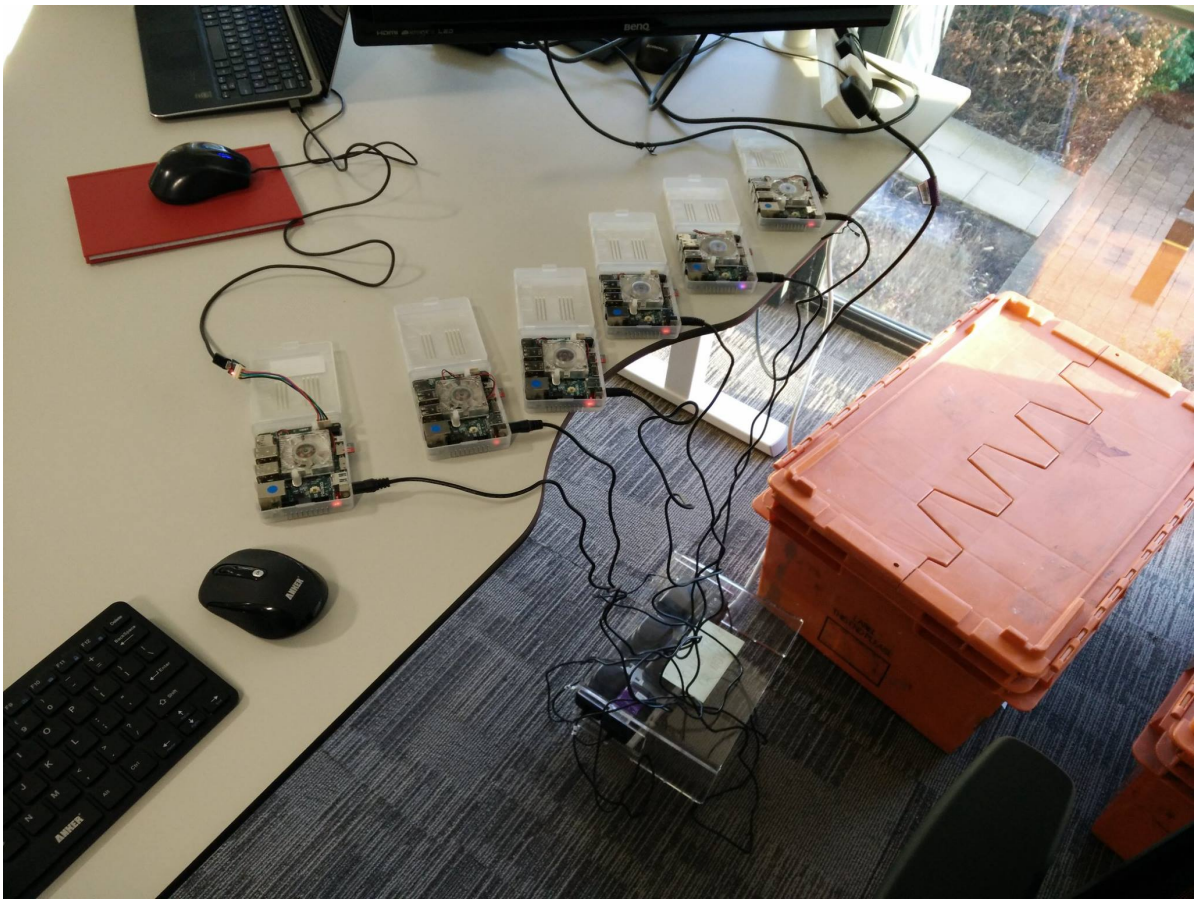
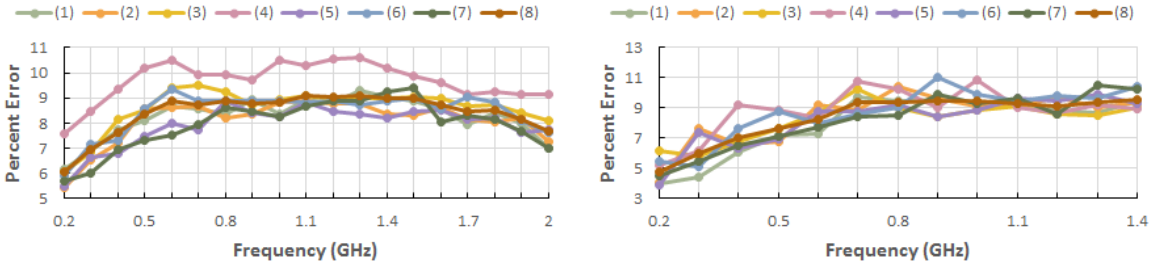


Figure 5.17: **Experiment set-up for the platform difference analysis.** This is the layout of the multi-board experiment performed during the research sabbatical at ARM in Cambridge. The ODROID XU3 boards were set up using the same type of mSD card and OS image. All environmental conditions are kept the same for all boards.

The resulting model error on each board is shown in Figure 5.18 and Table 5.12. A surprising result is that the model overall has a higher measured error than expected, as well as the great

error variation between the different ODROID XU3 boards.



(a) ARM Cortex-A15 Models

(b) ARM Cortex-A7 Models

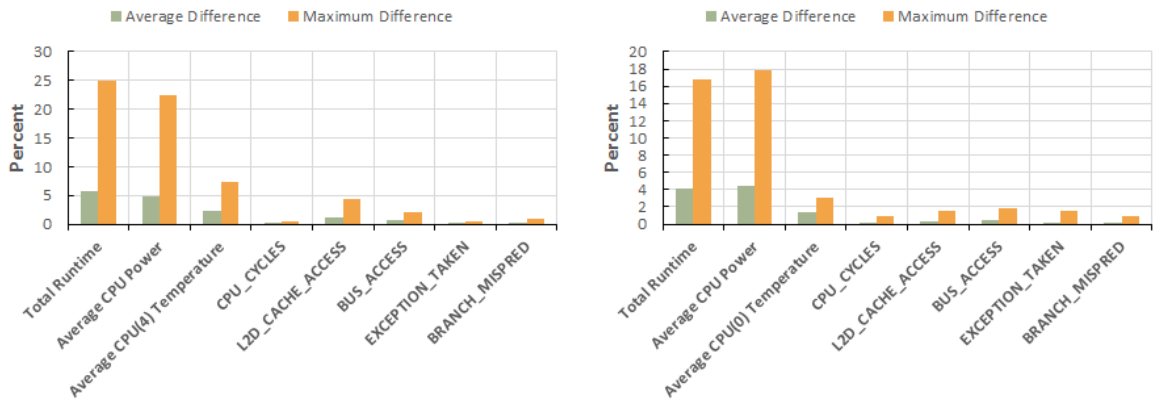
Figure 5.18: **Comparison between the generated per-frequency level single-thread low event cross-correlation model for both processor types on each of the ODROID XU3 boards.** The models are generated and validated by the custom methodology using the cBench workload for the two processor types on each of the ODROID XU3 development platforms with the mSD card as OS driver. The *Average Percent Error* at each available CPU frequency for the models is presented on the graph. The board references corresponding to the legend are given in Table 5.12.

Table 5.12: **Model reference and total average error for the data in Figure 5.18.** The board *Reference* is presented for each *Model Code* from the legend. *Total Average Error* values are obtained for each model on the two processor types (when available) by using their prediction error values across all frequency levels. The boards are represented by a number from *1* to *7*.

Model Code	Reference	Total Average Error	
		ARM Cortex-A15	ARM Cortex-A7
(1)	xu3_1	8.24%	7.93%
(2)	xu3_2	8.10%	8.44%
(3)	xu3_3	8.59%	8.22%
(4)	xu3_4	9.68%	8.82%
(5)	xu3_5	7.91%	8.23%
(6)	xu3_6	8.44%	8.68%
(7)	xu3_7	7.95%	8.20%
(8)	Mean	8.42%	8.36%

However, of particular interest was the dramatic variability between the different boards with respect to total workload runtime and average power despite using cset to control workload execution. The results of the development board variability are shown in Figure 5.19. Yet despite this the models themselves perform restively similar with relative error varying between 8 and 10% on both the Cortex-A15 and Cortex-A7.

The next aim is to further improve the model with the hopes of getting below 5% error on both CPU core types. In order to do that using machine learning methods were used to try and identify the most optimal set of PMU events from a larger pool of events, instead of relying on



(a) ARM Cortex-A15 Data (b) ARM Cortex-A7 Data

Figure 5.19: **Difference between single-thread workload executions on both processor types on the ODROID XU3 boards.** The data presented is the *Average* and *Maximum Percent Difference* between all of the cBench workload executions for both processor types on the ODROID XU3 development platforms with the same type of mSD card as OS driver. The presented metrics are the workload *Total Runtime*, CPU *Average Power*, CPU *Average Temperature* and the collected hardware events for the *low event cross-correlation per-frequency level* model.

intuition. The work of Jacobson et al. [59] and Hsieh et al. [43] greatly influenced the decision to pursue this approach by demonstrating that computer-aided methods produce much better results compared to conventional means of event selection.

5.5 Methodology refinement using automatic search

First a set of 22 common events for the Cortex-A15 and Cortex-A7 was identified from information in their respective Technical Reference Manuals [75][78]. The events are listed below: **CPU_CYCLES**; **L1I_CACHE_REFILL**; **L1I_TLB_REFILL**; **L1D_CACHE_REFILL**; **L1D_CACHE_ACCESS**; **L1D_TLB_REFILL**; **INST_RETIRED**; **EXCEPTION_TAKEN**; **EXCEPTION_RETURN**; **CID_WRITE_RETIRED**; **BRANCH_MISPRED**; **BRANCH_PRED**; **DATA_MEM_ACCESS**; **L1I_CACHE_ACCESS**; **L1D_CACHE_EVICTION**; **L2D_CACHE_ACCESS**; **L2D_CACHE_REFILL**; **L2D_CACHE_WB**; **BUS_ACCESS**; **BUS_CYCLES**; **BUS_READ_ACCESS**; **BUS_WRITE_ACCESS**

In order to be able to do an automated search between these events it was necessary to be able to collect them all in one big dataset. Unfortunately, as detailed in section 2.3.3, the key limitation of the hardware PMU is that it has a limited number of registers that can collect the events concurrently. In order to obtain accurate data several executions of the workload needed to be performed so as to obtain all the PMU events in chunks. In order to obtain a complete

dataset and synchronize all the PMU events together to the data from the on-board power sensors a complicated data concatenation script, detailed in Appendix C.1, was developed.

After all the PMU events have been collected and grouped all the PMU events in one big dataset, the automatic search algorithm can be initiated. A straightforward approach was developed, which uses **CPU_CYCLES** as a base model event and then proceeds to test out all the other PMU events with it. The algorithm tries the combinations one by one and continues only if there is an improvement. Since minimizing event cross-correlation did not yield the expected, the next goal is to minimise the relative model error. After checking the events one by one and identifying the event combination that achieves the best performance, the algorithm tries to add another one to the list. It does this until the predefined number of events is reached or until the model cannot be improved any more. This algorithm is named ***bottom-up automatic search***. Its details are found in C.3. After an optimal events list has been identified, the methodology proceeds to validate the model on real hardware by collecting the events concurrently, which gives the final accuracy metric.

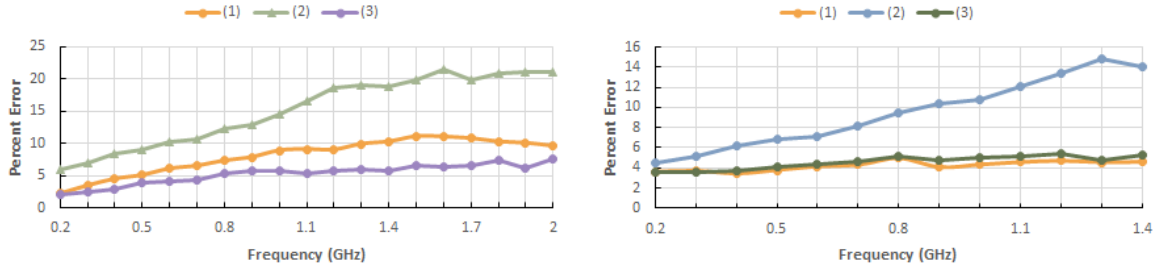
It was decided to train a dedicated model for the Cortex-A15 and the Cortex-A7 separately to see if there would be a variation in the selected events. As expected the algorithm has identified two differing models. Equation 5.14 describes the model for the ***big*** core and Equation 5.14 describes the one for the ***LITTLE***.

$$\begin{aligned} P(W) = {} & \alpha_0 + \alpha_1 \times CPU\_temperature \\ & + \alpha_2 \times CPU\_CYCLES + \alpha_3 \times BUS\_CYCLES \\ & + \alpha_4 \times DATA\_MEM\_ACCESS + \alpha_5 \times L1I\_CACHE\_ACCESS \\ & + \alpha_6 \times L1D\_CACHE\_EVICTION + \alpha_7 \times INST\_RETIRED \end{aligned} \tag{5.14}$$

$$\begin{aligned} P(W) = {} & \alpha_0 + \alpha_1 \times CPU\_temperature \\ & + \alpha_2 \times CPU\_CYCLES + \alpha_3 \times BUS\_CYCLES \\ & + \alpha_4 \times DATA\_MEM\_ACCESS + \alpha_5 \times L1D\_CACHE\_REFILL \\ & + \alpha_6 \times BUS\_READ\_ACCESS \end{aligned} \tag{5.15}$$

Figure 5.20 and Table 5.13 show the model performance. The low event correlation model was labelled as ***T&MLCC***, which stands for ***Temperature and Manual Low Cross-Correlation*** and the model obtained using the automated search ***T&ASLE***, short for ***Temperature and Automated Search Low Error***. It is evident that the newly trained model has very good performance, but the ***Temperature and Manual Low Cross-Correlation*** model has surprisingly high error, higher than the one generated during the methodology reproducibility experiment in Figure 5.18 and Table 5.12. This is investigated this later in Subsection . Nevertheless the results prove that adding automatic search to the methodology can definitely produce accurate models and, as evidenced by this work, is better than just relying on intuition, since the models can be tuned more easily for the Cortex-A15 or the Cortex-A7. Identifying optimal dedicated

models based on intuition, requires a lot of mechanistic knowledge of the architecture and CPU internal. Using machine learning requires more preparation time in order to collect all the events in the data set, but ultimately it is architecture agnostic and can be used on any platform as long as the system is set-up properly. Since the resulting model accuracy is satisfactory, the next step is to revisit the previous experiment investigating the developed methodology's reproducibility.



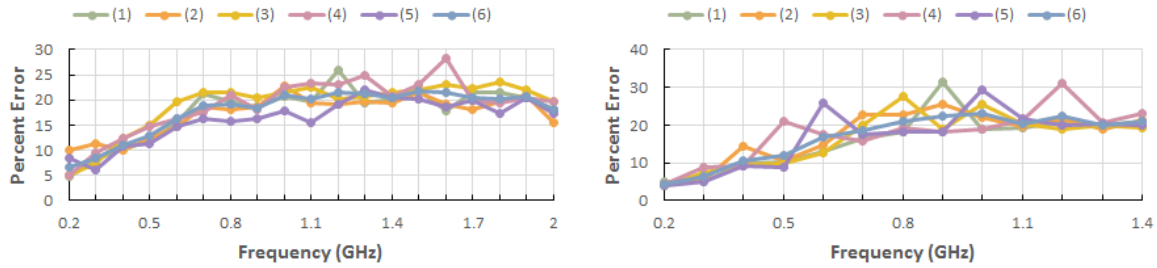
(a) ARM Cortex-A15 Models (b) ARM Cortex-A7 Models

Figure 5.20: **Comparison between the generated per-frequency level single-thread P2, T&MLCC and T&ASLE models for both processor types on the ODROID XU3 board.** The models are generated and validated by the custom methodology using the cBench workload for the two processor types on the ODROID XU3 development platform with the eMMC card as OS driver. The *Average Percent Error* at each available CPU frequency for the models is presented on the graph. The model references corresponding to the legend are given in Table 5.13.

Table 5.13: **Model reference and total average error for the data in Figure 5.20.** The model *Reference* is presented for each *Model Code* from the legend. *Total Average Error* values are obtained for each model on the two processor types (when available) by using their prediction error values across all frequency levels.

Model Code	Reference	Total Average Error	
		ARM Cortex-A15	ARM Cortex-A7
(1)	P2	8.06%	4.20%
(2)	T&MLCC	15.13%	9.45%
(3)	T&ASLE	5.24%	4.57%

This time results from only 5 boards were acquired, 4 from the sponsoring company and the ordinal ODROID XU3 platform used at the in this research. The **XU3_1** had a software big during the course of the experiment and was not able to generate results. Figure 5.21 and Table 5.14 show the results of the experiment. A surprising outcome is that the model performance data that was obtained was much worse than expected, even worse than the models in the initial experiment. The boards exhibited the same variability and characteristics as before, indicated by 5.22.



(a) ARM Cortex-A15 Models

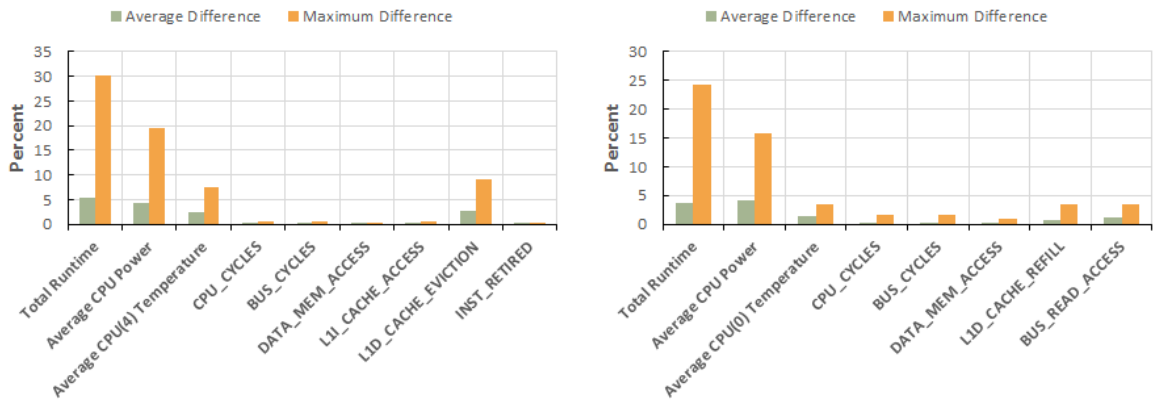
(b) ARM Cortex-A7 Models

Figure 5.21: **Comparison between the generated per-frequency level single-thread T&ASLE model for both processor types on each of the ODROID XU3 boards.** The models are generated and validated by the custom methodology using the cBench workload for the two processor types on each of the 6 ODROID XU3 development platforms with the mSD card as OS driver. The *Average Percent Error* at each available CPU frequency for the models is presented on the graph. The board references corresponding to the legend are given in Table 5.12.

Table 5.14: **Model reference and total average error for the data in Figure 5.21.** The board *Reference* is presented for each *Model Code* from the legend. *Total Average Error* values are obtained for each model on the two processor types (when available) by using their prediction error values across all frequency levels. The boards are represented by a number from *2* to *6*.

Model Code	Reference	Total Average Error	
		ARM Cortex-A15	ARM Cortex-A7
(1)	xu3_2	17.73%	16.06%
(2)	xu3_3	17.33%	17.22%
(3)	xu3_4	19.08%	16.42%
(4)	xu3_5	18.93%	17.60%
(5)	xu3_6	16.25%	16.82%
(6)	Mean	17.87%	16.82%

After investigating this issue, it was identified that there is a fundamental oversight that was made. When the model was trained and tested on the original ODROID XU3 board, the eMMC card was used as OS driver and when doing the reproducibility experiment the mSD cards were used, due to their low cost and easy set-up. The problem is that the mSD card is not very stable in terms of power consumption. This is the reason for the big difference in the results between the first and second experiment, even with the ***T&MLCC*** model. In the first experiment an average of 10 runs was used to obtain results, but the second one needed to be completed at a much shorter time therefore only 5 workload executions were performed. More runs means obtaining more data to train the model on, which for a system with high variability helps a lot. This also explains why the ***T&ASLE*** models performs significantly poorly on the mSD compared to the eMMC - the eMMC is more stable so the power variation is smaller and



(a) ARM Cortex-A15 Data (b) ARM Cortex-A7 Data

Figure 5.22: **Difference between single-thread workload executions on both processor types on the ODROID XU3 boards for the second board difference experiment.** The data presented is the *Average* and *Maximum Percent Difference* between all of the cBench workload executions for both processor types on the ODROID XU3 development platforms with the same type of mSD card as OS driver. The presented metrics are the workload *Total Runtime*, CPU *Average Power*, CPU *Average Temperature* and the collected hardware events for the *T&ASLE* model.

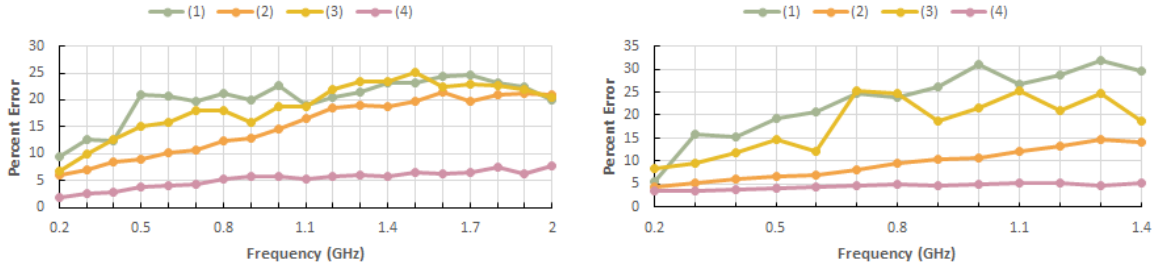
can be modelled witch much less data to fit.

The work of Kim et al. [107] highlights the impact of flash storage to performance in embedded and mobile systems. In addition Linaro's flash memory surveys highlight the differences between the different specifications and implementations [108] [109] [110]. They even have a comparison between the eMMC and mSD cards [111], but not in the context of power efficiency and stability. This has prompted a short analysis between the two memory systems, in order to determine which one is most suitable for the purposes of power and energy analysis and modelling.

5.6 Analysing model performance between the mSD and eMMC memory cards

The analysis is done by retraining and testing the ***T&MLCC*** and ***T&ASLE*** models on the first ODROID XU3 board - XU3_1, since the comparison needs to be done on the same system. This is done in order to avoid platform variability affecting the experiment results. Some of the data from the initial training and testing of the ***T&ASLE*** model is reused, which is why some of the model error numbers in Figure 5.23 and Table 5.15 can be seen in previous result listings. In order to keep things consistent 5 data runs are used for each separate data collection during the experiment. It is observed that as expected the ***T&ASLE*** model on the eMMC has the lowest

error for both Cortex-A15 and Cortex-A7 and none of the models for the mSD card achieve good performance. However it is interesting to note that the ***T&MLCC*** model also has a high error on the eMMC card. This is due to the fact that when the initial analysis to identify the least correlated events from the intuitive set was done, the mSD card was purposefully used in order to prepare the methodology for the reproducibility experiment. This means that the events of the ***T&MLCC*** model are tuned for the mSD card and the reason it performs poorly is because more than 5 runs are needed to generate enough data points for a good model.



(a) ARM Cortex-A15 Models (b) ARM Cortex-A7 Models

Figure 5.23: **Comparison between the generated per-frequency level single-thread T&MLCC and T&ASLE models for both processor and memory card types on the ODROID XU3 board.** The models are generated and validated by the custom methodology using the cBench workload for the two processor types on the ODROID XU3 development platform with the eMMC followed by the mSD card as OS driver. The *Average Percent Error* at each available CPU frequency for the models is presented on the graph. The model references corresponding to the legend are given in Table 5.15.

Table 5.15: **Model reference and total average error for the data in Figure 5.23.** The model *Reference* is presented for each *Model Code* from the legend. *Total Average Error* values are obtained for each model on the two processor types (when available) by using their prediction error values across all frequency levels.

Model Code	Reference	Total Average Error	
		ARM Cortex-A15	ARM Cortex-A7
(1)	T&MLCC on mSD	20.07%	23.05%
(2)	T&MLCC on eMMC	15.13%	9.45%
(3)	T&ASLE on mSD	18.64%	18.24%
(4)	T&ASLE on eMMC	5.24%	4.57%

A very big indicator of the fundamental difference between the mSD and the eMMC card is the incredibly high variation between the two. Figure 5.24 shows that runtime difference can be as high as 90% and the average power as high as 35%. This means that the models cannot be reused between the memory cards at all and in order to achieve a good performance a specific model need to be fitted for the target system. A consistently very high variation between the

PMU events on the two memory cards can be observed for both the ***T&MLCC*** and ***T&ASLE*** models.

(a) ARM Cortex-A15 T&MLCC Events Data

(b) ARM Cortex-A7 T&MLCC Events Data

(c) ARM Cortex-A15 T&ASLE Events Data

(d) ARM Cortex-A7 T&ASLE Events Data

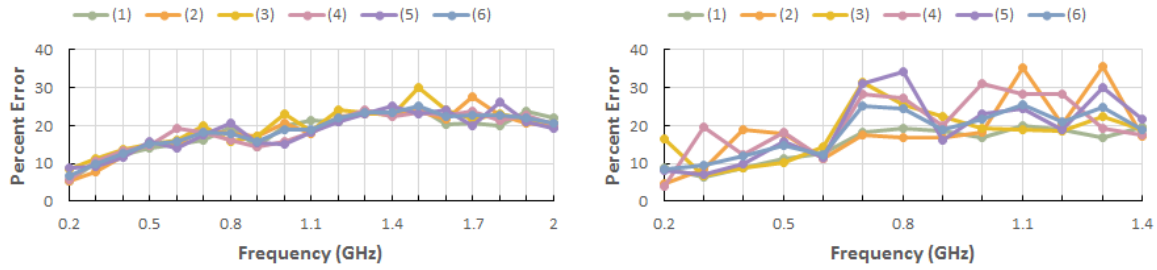
Figure 5.24: **Difference between single-thread workload executions on both processor types on the ODROID XU3 boards using both memory card types.** The data presented is the *Average* and *Maximum Percent Difference* between the cBench workload executions for both processor types on the ODROID XU3 development platform with the eMMC followed by the mSD card as OS driver. The presented metrics are the workload *Total Runtime*, CPU *Average Power*, CPU *Average Temperature* and the collected hardware events for the *T&MLCC* and *T&ASLE* models during independent executions.

Once the big differences between the two flash memory cards and their impact on total system performance and energy efficiency the experiment have been identified and highlighted, key priority becomes identifying which one is better suited for the power modelling methodology. In order to do this the card stability is investigated by isolating the different experiment runs and comparing the workload runtime, average power, temperature and PMU events. For this purpose only data for one set of events needed to be collected so the ***T&ASLE***

model was selected.

5.6.1 Investigating mSD card stability

First the analysis begins with investigating the mSD card. Figure 5.25 shows that despite the high error, the model is fairly consistent across the 5 runs of the experiment. However the performance variability for the Cortex-A7 is very high.



(a) ARM Cortex-A15 Models

(b) ARM Cortex-A7 Models

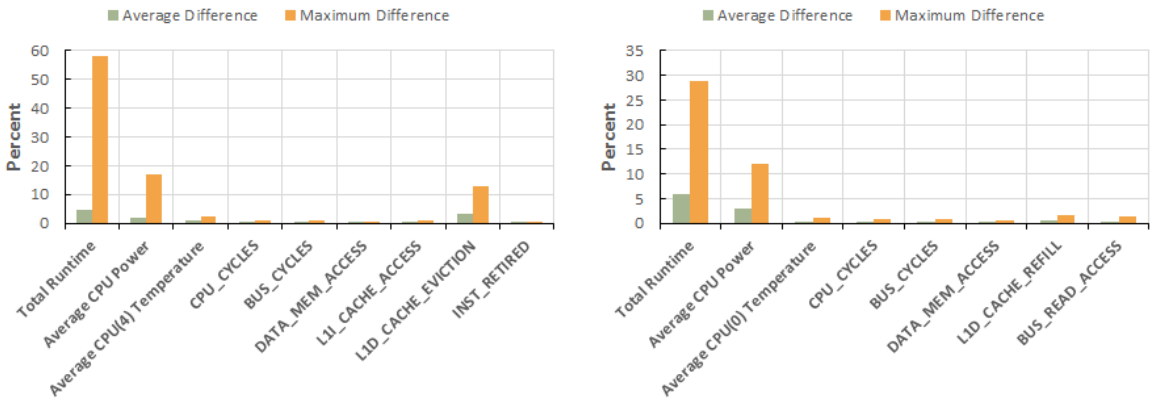
Figure 5.25: **Comparison between the repeatedly generated per-frequency level single-thread T&ASLE model for both processor types on each of the ODROID XU3 boards using the mSD card as OS driver.** The models are generated and validated by the custom methodology using the cBench workload for the two processor types on the ODROID XU3 development platform with the mSD card as OS driver. The *Average Percent Error* at each available CPU frequency for the models is presented on the graph. The model references corresponding to the legend are given in Table 5.16.

Table 5.16: **[Model reference and total average error for the data in Figure 5.25.** The experiment execution *Reference* is presented for each *Model Code* from the legend. *Total Average Error* values are obtained for each model on the two processor types (when available) by using their prediction error values across all frequency levels.

Model Code	Reference	Total Average Error	
		ARM Cortex-A15	ARM Cortex-A7
(1)	Run 1	18.34%	15.08%
(2)	Run 2	18.62%	18.37%
(3)	Run 3	19.56%	17.95%
(4)	Run 4	18.33%	20.40%
(5)	Run 5	18.37%	19.38%
(6)	Mean	18.64%	18.24%

Figure 5.26 shows the same results as when comparing the different platforms during the reproducibility experiment. The PMU events are consistently stable, due to using cset with the methodology to minimise OS overhead. Despite this there is a lot of variation in the workload

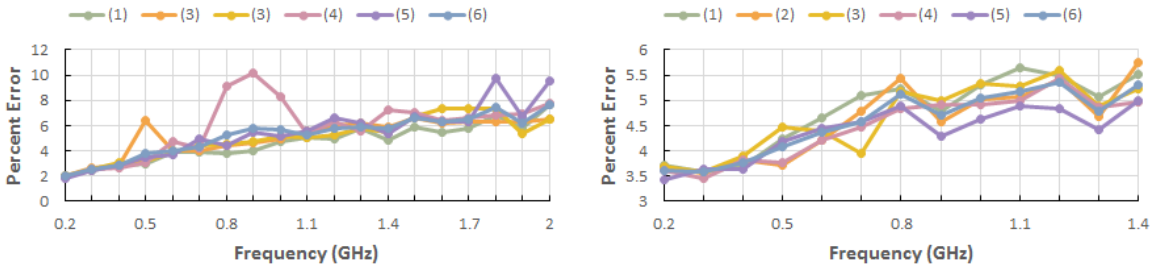
runtime and average power between the runs for both the Cortex-A15 and Cortex-A7. This means that making consistent models for this memory system is a big challenge.



(a) ARM Cortex-A15 Data (b) ARM Cortex-A7 Data

Figure 5.26: **Difference between single-thread workload executions on both processor types on the ODROID XU3 board using the mSD card as OS driver.** The data presented is the *Average* and *Maximum Percent Difference* between all of the cBench workload executions for both processor types on the same ODROID XU3 development platform with the mSD card as OS driver. The presented metrics are the workload *Total Runtime*, CPU *Average Power*, CPU *Average Temperature* and the collected hardware events for the *T&ASLE* model.

In the end the initial hypothesis that the mSD is very unstable and not very suitable for power modelling holds true.



(a) ARM Cortex-A15 Models (b) ARM Cortex-A7 Models

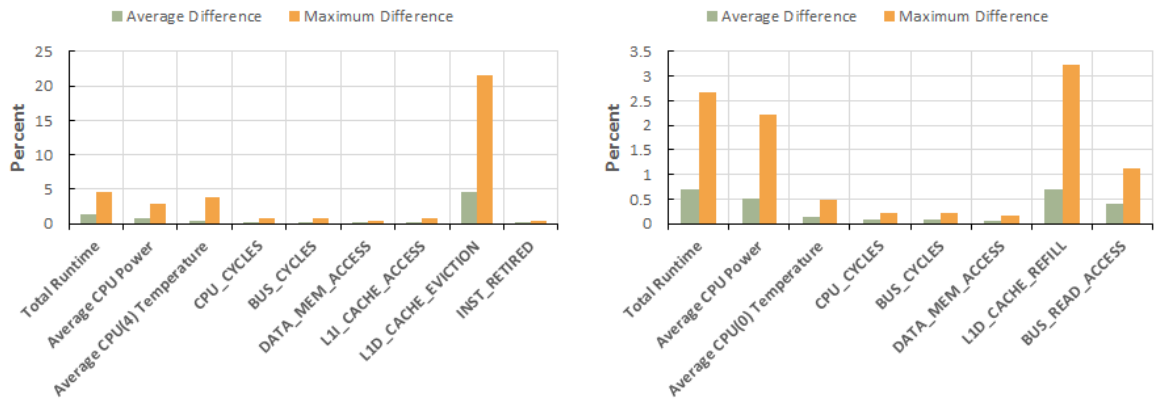
Figure 5.27: **Comparison between the repeatedly generated per-frequency level single-thread T&ASLE model for both processor types on each of the ODROID XU3 boards using the eMMC card as OS driver.** The models are generated and validated by the custom methodology using the cBench workload for the two processor types on the ODROID XU3 development platform with the eMMC card as OS driver. The *Average Percent Error* at each available CPU frequency for the models is presented on the graph. The model references corresponding to the legend are given in Table 5.17.

5.6.2 Investigating eMMC card stability

The results from the mSD experiment are compared to the ones for the eMMC card and it is identified that the same PMU events can capture system behaviour much better. Figure 5.27 and Table 5.17 show that despite some variability on the ARM Cortex-A15, overall the eMMC card can produce much better power models.

Table 5.17: **Model reference and total average error for the data in Figure 5.27.** The experiment execution *Reference* is presented for each *Model Code* from the legend. *Total Average Error* values are obtained for each model on the two processor types (when available) by using their prediction error values across all frequency levels.

Model Code	Reference	Total Average Error	
		ARM Cortex-A15	ARM Cortex-A7
(1)	Run 1	4.65%	4.77%
(2)	Run 2	5.16%	4.59%
(3)	Run 3	5.04%	4.65%
(4)	Run 4	5.93%	4.48%
(5)	Run 5	5.44%	4.37%
(6)	Mean	5.24%	4.57%



(a) ARM Cortex-A15 Data (b) ARM Cortex-A7 Data

Figure 5.28: **Difference between single-thread workload executions on both processor types on the ODROID XU3 board using the eMMC card as OS driver.** The data presented is the *Average* and *Maximum Percent Difference* between all of the cBench workload executions for both processor types on the same ODROID XU3 development platform with the eMMC card as OS driver. The presented metrics are the workload *Total Runtime*, CPU *Average Power*, CPU *Average Temperature* and the collected hardware events for the *T&ASLE* model.

Final analysis shows that the eMMC variability is very low and the card is very stable. The results show below 5% variation for the ARM Cortex-A15 and 3% variation for the ARM Cortex-A7 on almost all collected points of reference. The only things with high variability are

some cache-related PMU events, which can reduce model performance consistency. This issue is addressed later on in Section 5.7.

The evident conclusion of the reproducibility and the memory system comparison experiments is that the eMMC card is much more suitable for power modelling, since it requires less workload executions in order to produce accurate models. Similar conclusions as [107] have been derived in that the type of memory used on the mobile platform greatly influences performance and energy efficiency. Any further analysis is beyond the scope of this thesis. For all future experiments only the eMMC card is used. From this point on this work focuses on refining the methodology and there was no other opportunity to revisit the reproducibility experiments and improve upon the mSD models.

5.6.3 Addressing the temperature variability

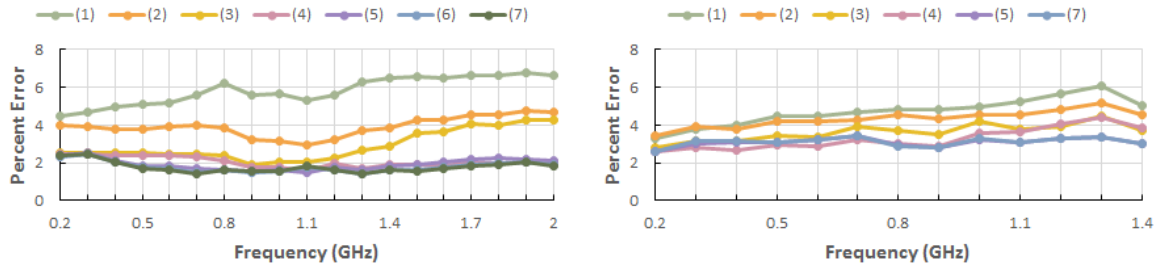
Given all of this, the ODROID-XU3 is a great platform, but it does have one big limitation, which was left unresolved. The platform itself has a big cooling unit in order to prevent it from overheating, so the effects of temperature on the power models are unable to be explored. This investigation is mentioned in Section 3.3, where an overview of the ODROID XU3 board is presented. For the purposes of the experiments the fan was switched on at highest setting during the whole course of the workloads and temperature was not included in the model events.

5.7 Further development - complete exploration of the PMU event set

This Section briefly outlines the results of applying the machine learning methods on the concatenated data. An approach has been developed, which uses **CPU_CYCLES** as base event and tries to add 1 more PMU event each iteration of the search algorithm until the desired number of events is reached or no more event that improve the accuracy can be found. This is labelled as ***bottom-up*** search. The pseudocode version of the algorithm is presented in Appendix C.3. Figure 5.29 and Table 5.18 represent the effects of the search algorithm with each event added. Note that **CPU_CYCLES** is always present as a first event in the models since it has a dedicated PMU register as well as being highly correlated to power on its own, as is shown by model code (1).

The model is computed using OLS method in ***octave*** and follows the equations listed below. I can be seen that the Cortex-A15 model does not use the full 7 registers available in the PMU for concurrent collection, since it has achieved a local optima at 6 events.

The ***per-frequency*** models are given in equation 5.16 and 5.17 and the mathematical example expression for the ***intra-core*** model is given in equation 5.19. The reasoning behind model 5.19 is to allow the ability to predict average power from the events from one frequency



(a) ARM Cortex-A15 Models

(b) ARM Cortex-A7 Models

Figure 5.29: **Breakdown of the generated per-frequency level single-thread model errors at each iteration of the automatic event selection algorithm for both processor types on the ODROID XU3 board.** The models are generated and validated by the custom methodology using the cBench workload for the two processor types on the ODROID XU3 development platform with the eMMC card as OS driver. The *Average Percent Error* at each available CPU frequency for the models is presented on the graph. The model references corresponding to the legend are given in Table 5.18.

Table 5.18: **Model reference and total average error for the data in Figure 5.29.** The model *Reference* is presented for each *Model Code* from the legend. *Total Average Error* values are obtained for each model on the two processor types (when available) by using their prediction error values across all frequency levels.

Model Code	Reference	Total Average Error	
		ARM Cortex-A15	ARM Cortex-A7
(1)	CPU_CYCLES	5.83%	4.72%
(2)	1 Event	3.91%	4.34%
(3)	2 Events	2.90%	3.63%
(4)	3 Events	2.06%	3.28%
(5)	4 Events	1.92%	3.09%
(6)	5 Events	1.79%	-
(7)	CPU_CYLES + Events	1.76%	3.10%

level for another on the same core, hence its chosen name. The main distinction is that it only predict average power an not real-time power like the ***per-frequency*** models which can only be used on one level. A way to think about this is that the ***per-frequency*** model gives a detailed analysis on the application at runtime (0.5s interval) if the frequency is unchanged, and the ***intra-core*** model gives an estimation of the average power on various frequency levels based on a window of PMU events. The ***intra-core*** model uses a simple approach to scale the current incoming PMU events by using the property that the events of each data sample are approximately as proportional as their averages for the whole runtime of the workload. Thus turning real-time information to information about average data. The event scaling method is demonstrated in Equation 5.18, where ev_{1_f1} and ev_{1_f2} represent the value of hardware

event ev_1 at data samples on two CPU frequencies $f1$ and $f2$. After the scaling factor has been identified by using the training set then the ***intra-core*** model can be validated using one frequency level from the test set as *f1* and the other as *f2* in equation 5.19. This special model can be extended into a power-aware scheduler for DVFS.

$$\begin{aligned} P_{CPU\_A15} = {} & \alpha_0 + \alpha_1 \times CPU\_CYCLES + \alpha_2 \times L1I\_CACHE\_ACCESS \\ & + \alpha_3 \times L1D\_CACHE\_ACCESS + \alpha_4 \times BUS\_CYCLES \\ & + \alpha_5 \times BUS\_PERIPH\_ACCESS + \alpha_6 \times BRANCH\_SPEC\_EXEC\_RET \end{aligned} \tag{5.16}$$

$$\begin{aligned} P_{CPU\_A7} = {} & \alpha_0 + \alpha_1 \times CPU\_CYCLES + \alpha_2 \times BUS\_READ\_ACCESS \\ & + \alpha_3 \times L2D\_CACHE\_REFILL + \alpha_4 \times UNALIGNED\_LOAD\_STORE \\ & + \alpha_5 \times BUS\_CYCLES \end{aligned} \tag{5.17}$$

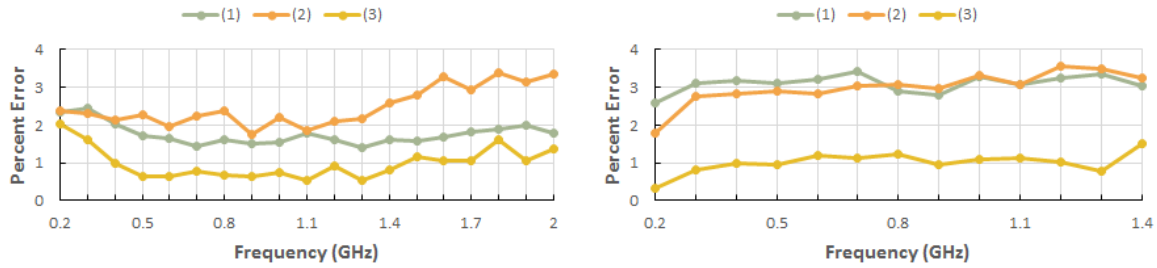
$$\begin{aligned} \frac{ev_{1\_f1}}{ev_{1\_f2}} &= \frac{\overline{ev_{1\_f1}}}{\overline{ev_{1\_f2}}} \\ ev_{1\_f1}\frac{\overline{ev_{1\_f2}}}{\overline{ev_{1\_f1}}} &= ev_{1\_f2} \\ \frac{\overline{ev_{1\_f2}}}{\overline{ev_{1\_f1}}} &= Event\ Scaling\ Factor \end{aligned} \tag{5.18}$$

$$P_{CPU\_f1} = \alpha_0 + \alpha_1 \times ev_{1\_f2} \times \frac{\overline{ev_{1\_f1}}}{\overline{ev_{1\_f2}}} + \alpha_2 \times ev_{2\_f2} \times \frac{\overline{ev_{2\_f1}}}{\overline{ev_{2\_f2}}} + \ldots + \alpha_n \times ev_{n\_f2} \times \frac{\overline{ev_{n\_f1}}}{\overline{ev_{n\_f2}}} \tag{5.19}$$

Table 5.19: **Model reference and total average error for the data in Figure 5.30.** The model *Reference* is presented for each *Model Code* from the legend. *Total Average Error* values are obtained for each model on the two processor types (when available) by using their prediction error values across all frequency levels.

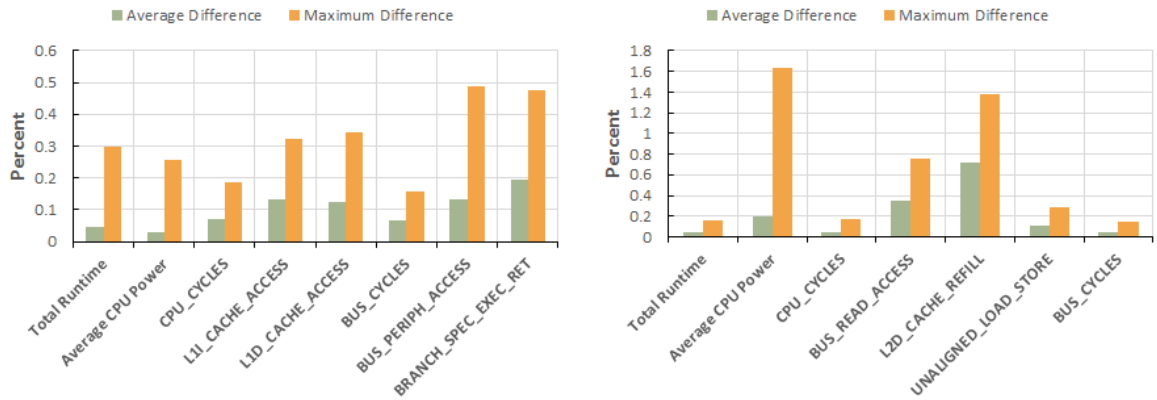
Model Code	Reference	Total Average Error	
		ARM Cortex-A15	ARM Cortex-A7
(1)	Automatic	1.76%	3.10%
(2)	Collected	2.49%	2.99%
(3)	Intra-Core	0.99%	1.01%

Figure 5.30 and Table 5.19 represent the final outcome of the machine learning assisted methodology, with equation number in the figure and corresponding names given in the table. The experiment data shows that there is not much of a difference between the automatic search using concatenated data and the hardware validated model with all events collected at the same time. It can also be observed that the ***intra-core*** model has higher accuracy, but



(a) ARM Cortex-A15 Models (b) ARM Cortex-A7 Models

Figure 5.30: **Comparison between the generated per-frequency level single-thread Automatic, Collected and Intra-Core models for both processor types on the ODROID XU3 board.** The models are generated and validated by the custom methodology using the cBench workload for the two processor types on the ODROID XU3 development platform with the eMMC card as OS driver. The *Average Percent Error* at each available CPU frequency for the models is presented on the graph. The model references corresponding to the legend are given in Table 5.19.



(a) ARM Cortex-A15 Data (b) ARM Cortex-A7 Data

Figure 5.31: **Difference between single-thread workload executions on both processor types on the ODROID XU3 board.** The data presented is the *Average* and *Maximum Percent Difference* between all of the cBench workload executions for both processor types on the ODROID XU3 development platforms with the eMMC card as OS driver. The presented metrics are the workload *Total Runtime*, CPU *Average Power*, CPU *Average Temperature* and the collected hardware events for the *collected* model.

bear in mind it only approximates the average power and cannot give detailed application profiling during execution like the standard ***per-frequency*** model can. The developed models are also well within the target accuracy, so the methodology can make accurate models for the single-thread case.

Figure 5.31 shows the workload runtime, average power and PMU event variability be-

tween the experiment runs. It is very low, because only stable events have been chosen and the methodology has low measurement overhead. Low variability is critical when building replicable models, even on a different set of data. Some memory events, which might have high correlation to power, also have a high degree of variability which might make the model not usable in certain circumstances, hence the choice to only use events with less than 5% variability in the search tree. Out of the 67 PMU events available for the Cortex-A15 and the 42 available for the Cortex-A7 only 55 for the Cortex-A15 and 31 for the Cortex-A7 are considered for possible selection for the ***per-frequency*** model. This is the reason the models achieve such low event variability and model error.

5.8 Model validation and application

The final part in the model evaluation is comparison against other published work. There are quite a few other researchers working in this area as well as some models built on the same research platform. The models are compared against the previous work in Nikov et al [21], which did not include the machine learning methods, as well as Pricopi et al [50] and Walker et al [62] which both utilise PMU events for their models and are both developed for ARM CPUs. This is done by using the custom methodology and their set of PMU events to train and validate their models. Their reported accuracy in the respective publications is also taken into account to see if there is a significant difference in the used methodologies. The developed approach uses a significantly larger number of samples for the data set compare to others and the entire frequency range is explored in much greater detail with the ***per-frequency*** models. Also, to the researcher's knowledge, this work is the only one to present a true power model for the heterogeneous system with the ***intra-core*** model, instead of developing separate models for the different types of processing units.

Pricopi [50]

This model is labelled as equation code (1) in Figure 5.32 and Table 5.20. They have a mechanistic model for the Cortex-A15, which utilises experience and a deeper understanding of the architecture to select the list of PMU events used. They have not produced a model for the Cortex-A7 on the justification that the CPU core does not exhibit much variation in its power dissipation and can be approximated by a single number. Their work is done on an experimental platform and on a single CPU frequency, hence the simplified power profile. Nevertheless this is one of the earliest PMU based models available for the ARMv7 architecture and provides great insight into the use of PMU

events for power modelling.

$$P_{CPU\_A15} = \alpha_0 + \alpha_1 \times \frac{INST\_SPEC\_EXEC}{CPU\_CYCLES} + \alpha_2 \times \frac{INST\_SPEC\_EXEC\_INT}{INST\_SPEC\_EXEC} + \alpha_3 \times \frac{INST\_SPEC\_EXEC\_VFP}{INST\_SPEC\_EXEC} + \alpha_4 \times \frac{L1D\_CACHE\_ACCESS}{INST\_SPEC\_EXEC} + \alpha_5 \times \frac{L2D\_CACHE\_ACCESS}{INST\_SPEC\_EXEC} + \alpha_6 \times \frac{L2D\_CACHE\_REFILL}{INST\_SPEC\_EXEC} \tag{5.20}$$

Walker [62]

Labelled as equation code (2) in Figure 5.32 and Table 5.20. The research team from University of Southampton have published a model for the Cortex-A15 on the same platform, the ODROID-XU3. They use a similar machine learning method to traverse the list of available PMU events, but their methodology uses a different workload and does not utilise some of this work's approaches to minimise overhead and event variability.

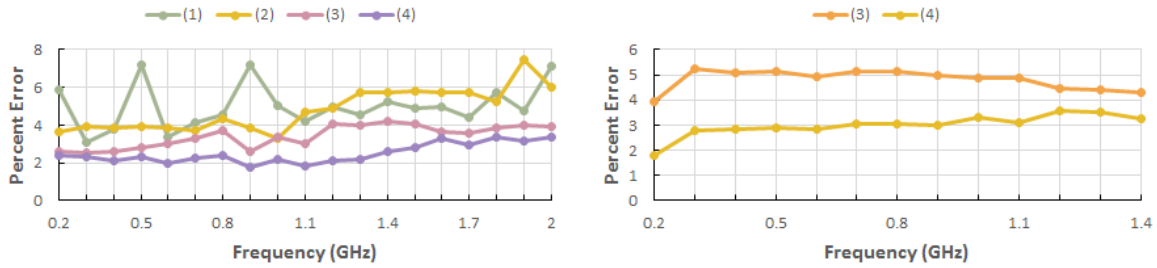
$$\begin{aligned} P_{CPU\_A15} = {} & \alpha_0 + \alpha_1 \times CPU\_CYCLES + \alpha_2 \times INST\_SPEC\_EXEC \\ & + \alpha_3 \times L2D\_READ\_ACCESS + \alpha_4 \times UNALIGNED\_ACCESS \\ & + \alpha_5 \times INST\_SPEC\_EXEC\_INT + \alpha_6 \times L1I\_CACHE\_ACCESS \\ & + \alpha_7 \times BUS\_ACCESS \end{aligned} \tag{5.21}$$

Old model [21]

Labelled as equation code (3) in Figure 5.32 and Table 5.20. In the previous work, an intuitive approach was used to select the PMU events. A previous version of the methodology was also used, which had higher overhead and variability. This is why improvement in the model accuracy can be seen using the new methodology even when using the same set of events. This is a clear indication that a great fundamental improvement in the methodology was achieved, not just in the developed power models.

$$\begin{aligned} P_{CPU} = {} & \alpha_0 + \alpha_1 \times CPU\_CYCLES + \alpha_2 \times L1D\_CACHE\_ACCESS \\ & + \alpha_3 \times L1I\_CACHE\_ACCESS + \alpha_4 \times INST\_RETIRED \\ & + \alpha_5 \times DATA\_MEM\_ACCESS \end{aligned} \tag{5.22}$$

From Figure 5.32 and Table 5.20 clearly indicate that the latest model, labelled as equation code (4), has the best performance. Some inconsistencies can also be seen in the reported accuracy of the other models compared to the measured one. This is due to the different methodology, but mostly due to the variation in workload. This is again another highlight that developing a methodology should be the focus of power modelling research since the many factors involved prevent there to be a universal solution to this complex problem. Instead research should be focusing on developing flexible solutions, which the industry can tailor and use for their own configurations and target applications. This work also demonstrates that



(a) ARM Cortex-A15 Models

(b) ARM Cortex-A7 Models

Figure 5.32: **Comparison between the generated per-frequency level single-thread collected model and other published work for both processor types on the ODROID XU3 board.** The models are generated and validated by the custom methodology using the cBench workload for the two processor types on the ODROID XU3 development platform with the eMMC card as OS driver. The *Average Percent Error* at each available CPU frequency for the models is presented on the graph. The model references corresponding to the legend are given in Table 5.20.

machine learning can be great for this king of optimisation problem, where the platform is used as a *black box* and its more accurate and more robust than using intuition and mechanistic information.

Table 5.20: **Model reference and total average error for the data in Figure 5.32.** The model *Reference* is presented for each *Model Code* from the legend. *Reported Average Error* values are taken from the related publication (when available). *Total Average Error* values are obtained for each model on the two processor types (when available) by using their prediction error values across all frequency levels.

Model Code	Reference	Reported Average Error		Total Average Error	
		ARM C.-A15	ARM C.-A7	ARM C.-A15	ARM C.-A7
(1)	Pricopi	1.2%	-	5.01%	-
(2)	Walker	2.8%	3.8%	4.81%	-
(3)	Nikov	-	-	3.41%	4.81%
(4)	New S-T	-	-	2.49%	2.99%

5.9 Summary of results

This chapter outlines the entire journey into perfecting the methodology for development of single-thread models on the *big.LITTLE* SoC. The initial models built on the ODROID XU+E and ARM Versatile Express Motherboard with CoreTile TC2 Daughterboard development platforms are thoroughly described. The methodology evolution, when using the ODROID XU3 development board, has also been presented. A key milestone is the deployment of

per-frequency models, which greatly improve prediction accuracy by reducing the system complexity the model needs to capture. Another significant contribution is the use of an automatic search algorithm to identify the best possible set of PMU events and achieve sub 3% model prediction error on both the ARM Cortex-A15 and ARM Cortex-A7 processors.

Some uncommon topics are also investigated, such as development platform variability as well as SoC temperature and memory system impact on performance and power. The difference between the eMMC and mSD memory card has been investigated into great depth with the conclusion that the eMMC memory card is much more stable and from that point on is used as OS driver for all later experiments.

It is claimed that the results and experiments provide unique insight into single-thread models for HES and have contributed greatly to the general area of power management.

CHAPTER 6

MULTI-THREAD MODELS

After investigating the single-thread models, the research focuses on exploring the multi-thread case. For these purposes a different workload with a different system set-up was used, detailed in the following chapter.

It begins with Section 6.1 describing the initial efforts in developing multi-thread power models on the ODROID XU3 board with information about the different system configurations and corresponding models.

Afterwards Chapter 6.2 explains the additional techniques and search algorithms that were developed in an effort to further improve model accuracy. Unfortunately the multi-thread power models, developed for ARM Cortex-A15 processor do not meet the required accuracy target. The causes of this and possible solutions to overcoming the model limitation are briefly presented.

Chapter 6.4 discusses the results of the comparison between the custom ***per-frequency*** level power models and related work, validated using the developed methodology.

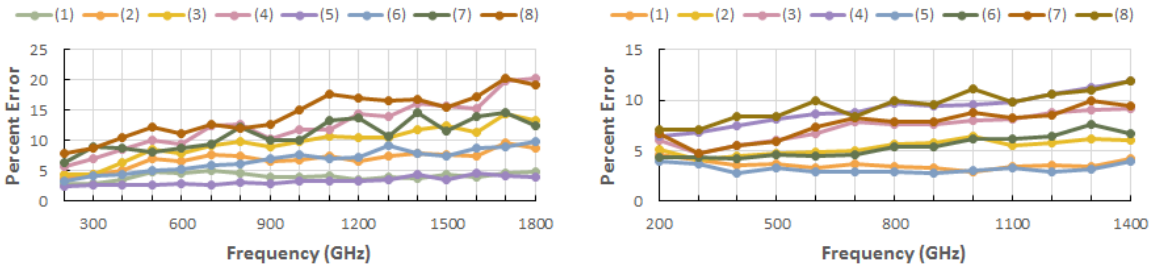
6.1 Initial results

In order to develop multi-thread models the PARSEC 3.0 benchmark suite was configured and used. Details about the workload are given in Section 4.1.2.2. The methodology was used to find out the most optimal PMU events for the multi-thread case using the ***bottom-up*** event search algorithm. Equations 6.1 and 6.2 show the mathematical expression for each best model. Both multi-thread models use very different events compared to the best performing single-thread models computed in Section 5.7. The power modelling equation for the ARM Cortex-A15 uses all 7 available counters in the PMU. It is interesting to note that the algorithm

identified branch and cache-related events to be the most useful to the model. The power model for the ARM Cortex-A7 uses all 5 available PMU counters and similarly to the Cortex-A15 model relies on cache-related events. In addition to the events the number of cores are also included in the model variables. The main reason for this is to distinguish between different configurations, but the event also smooths out the percent error prediction.

$$\begin{aligned} P_{CPU\_A15} = {} & \alpha_0 + \alpha_1 \times num\_cores + \alpha_2 \times CPU\_CYCLES \\ & + \alpha_3 \times L1D\_READ\_ACCESS + \alpha_4 \times BRANCH\_MISPRED \\ & + \alpha_5 \times BARRIER\_SPEC\_EXEC\_DMB + \alpha_6 \times L2D\_INVALIDATE \\ & + \alpha_7 \times BRANCH\_SPEC\_EXEC\_IMM\_BRANCH + \alpha_8 \times BUS\_CYCLES \end{aligned} \tag{6.1}$$

$$\begin{aligned} P_{CPU\_A7} = {} & \alpha_0 + \alpha_1 \times num\_cores + \alpha_2 \times CPU\_CYCLES \\ & + \alpha_3 \times L1I\_CACHE\_ACCESS + \alpha_4 \times L1D\_CACHE\_WB \\ & + \alpha_5 \times DATA\_READS + \alpha_6 \times IMMEDIATE\_BRANCHES \end{aligned} \tag{6.2}$$



(a) ARM Cortex-A15 Models (b) ARM Cortex-A7 Models

Figure 6.1: **Comparison between the generated per-frequency level multi-thread models for both processor types for different system configurations on the ODROID XU3 board.** The models are generated and validated by the custom methodology using the PARSEC 3.0 workload for the two processor types on the ODROID XU3 development platform with the eMMC card as OS driver. The *Average Percent Error* at each available CPU frequency for the models is presented on the graph. The model references corresponding to the legend are given in Table 6.1.

The train data set from all 4 CPU core configurations is used in order to fit the model coefficients. As stated earlier, the initial hypothesis for the multi-thread case was that having dedicated models for each core configuration (1,2,3 or 4 running processor cores in parallel) would yield the best performing models. It can be observed from Figure 6.1 and Table 6.1 that using the the full training data from all configuration produces a better overall model even when tested on the individual configuration case data. A dedicated model only produces a lower average percent error in the 1 core and 2 cores configurations, but it is not enough

Table 6.1: **Model reference and total average error for the data in Figure 6.1.** The model *Reference* is presented for each *Model Code* from the legend. *Total Average Error* values are obtained for each model on the two processor types (when available) by using their prediction error values across all frequency levels for the corresponding system configuration.

Model Code	Reference	Total Average Error	
		ARM Cortex-A15	ARM Cortex-A7
(1)	All Core Bottom-Up 1 Core	4.12%	3.65%
(2)	All Core Bottom-Up 2 Cores	6.97%	5.37%
(3)	All Core Bottom-Up 3 Cores	9.70%	7.33%
(4)	All Core Bottom-Up 4 Cores	12.69%	9.13%
(5)	Dedicated Bottom-Up 1 Core	3.34%	3.22%
(6)	Dedicated Bottom-Up 2 Cores	6.81%	5.41%
(7)	Dedicated Bottom-Up 3 Cores	11.06%	7.64%
(8)	Dedicated Bottom-Up 4 Cores	14.33%	9.50 %

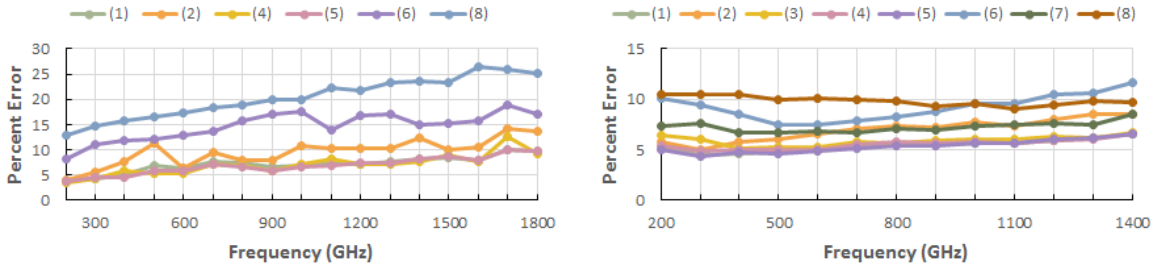
to overcome the significantly higher error in the other cases. This is because the ***dedicated*** models have different sets of events for each configuration, identified by the automatic search algorithm and actually using the data from all configurations for the ***all core*** model overall produces a more stable set of events.

6.2 Extended methodology

After investigating the ***all core vs. dedicated*** models case the research goes on to experimenting with a few more optimisation criteria for the search algorithm. Minimising the model error standard deviation and the PMU event cross correlation are two other approaches that could potentially improve model accuracy [62]. They do this by enabling the traversal of a different search tree and thus overcome any possible local minima that the automated search algorithm might be *trapped* in. A second automatic search method is also developed, which uses a ***top-down*** approach. The algorithm starts off by first making a model using all the available events and then slowly removing the event which causes this model to improve the most. This way the search tree is trimmed from the top, hence the name. Often this algorithm will get *stuck* on a larger set of events that can physically be collected concurrently on the PMU (7 for the Cortex-A15 and 5 for the Cortex-A7), so the rest of the search is done using an ***exhaustive*** approach which identifies all the combinations of 7 or 5 events from the already pruned search tree and extracts only the best performing combination. The reason the ***exhaustive*** method is not used on its own is because it takes a very long time to complete using the full events list. For example, for the Cortex-A15, all the combinations of 6 events out of the total list of 54 usable events need to be investigated, which means a total of 25827165 combinations (the justification about why **CPU_CYCLES** is always the first event of the model and the details about the usable events list are present in section 5). This is a very large number of

combination to go through, which is why these techniques have been developed in order to improve the probability of identifying the most *optimal* solution. The details of the ***top-down*** and ***exhaustive*** algorithms are presented in Appendix C.4 and C.5 respectively.

In addition to these investigations, a model generated only for the *multi-core* case on data from 2 cores, 3 cores and 4 cores configurations has been explored. The results of these experiments are shown in Figure 6.2 and Table 6.2. Only the most significant experiment cases are shown. The results highlight that the ***bottom-up*** algorithm produces the most accurate models, while taking much less computation time. Also minimising the error standard deviation or PMU event cross correlation does not improve model accuracy as evidenced by model codes (6), (7) and (8). Removing the **num_cores** event from the model as shown in model code (4) produces a slightly better power model for the Cortex-A15, but worse for the Cortex-A7. Also from Figure 6.2 a slight error spike can be observed at the higher frequencies so the conclusion is that it does not improve the base ***all core bottom-up*** model. The only other better model is model code (5) which removes the practical event limit and is allowed to use the best available PMU events. This model uses 9 events for the Cortex-A15 and again 9 events for the Cortex-A7 including **num_cores** and **CPU_CYCLES**. This indicates, that for such complex embedded systems a higher number of PMU counters might be beneficial when using system event based models for predicting power consumption.



(a) ARM Cortex-A15 Models

(b) ARM Cortex-A7 Models

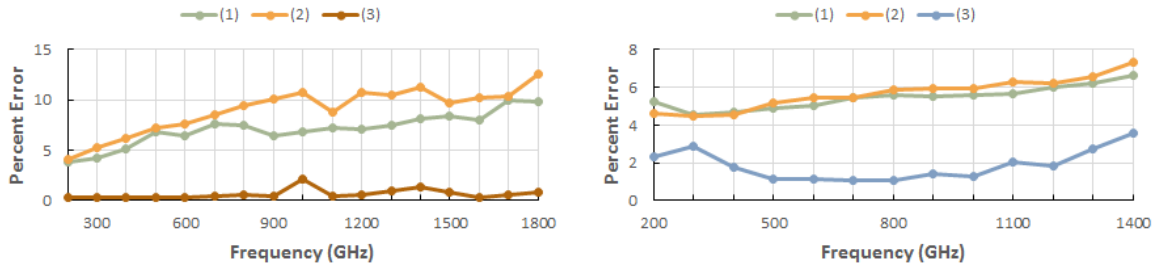
Figure 6.2: **Comparison between the different automatic event selection methods for generating per-frequency level multi-thread models for both processor types on the ODROID XU3 board.** The models are generated and validated by the custom methodology using the PARSEC 3.0 workload for the two processor types on the ODROID XU3 development platform with the eMMC card as OS driver. The *Average Percent Error* for the models at each available CPU frequency over the corresponding system configurations is presented on the graph. The model references corresponding to the legend are given in Table 6.2.

After identifying that the ***all core bottom-up per-frequency*** model is overall the best performing one it is validated using data from the concurrent real-time collection of the PMU events used in the model. Also as described in section 5.7 the event scaling technique is used to obtain the ***intra-core*** model. However, the **num_cores** event is left unscaled, since it is just

Table 6.2: **Model reference and total average error for the data in Figure 6.2.** The model *Reference* is presented for each *Model Code* from the legend. *Total Average Error* values are obtained for each model on the two processor types (when available) by using their prediction error values across all frequency levels and corresponding system configurations.

Model Code	Reference	Total Average Error	
		ARM C.-A15	ARM C.-A7
(1)	All Core Bottom-Up	7.12%	5.46%
(2)	Multicore Only Bottom-Up	9.55%	7.01%
(3)	All Core Top-Down + Exhaustive	-	5.91%
(4)	All Core Bottom-Up No Core Count	7.06%	5.57%
(5)	All Core Bottom-Up No Event Limit	6.92%	5.35%
(6)	All Core Bottom-Up Std.Dev.	14.68%	9.22%
(7)	All Core Top-Down + Exhaustive Std.Dev.	-	7.28%
(8)	All Core Bottom-up Cross Corr.	20.35%	9.86%

the static number of enabled cores and there is no need to adjust it like the other PMU events. The finalised models validated on real hardware are shown in Figure 6.3 and Table 6.3. As expected the collected model is slightly less accurate than the one using the concatenated PMU event data, especially for the higher frequency levels of the Cortex-A15.



(a) ARM Cortex-A15 Models

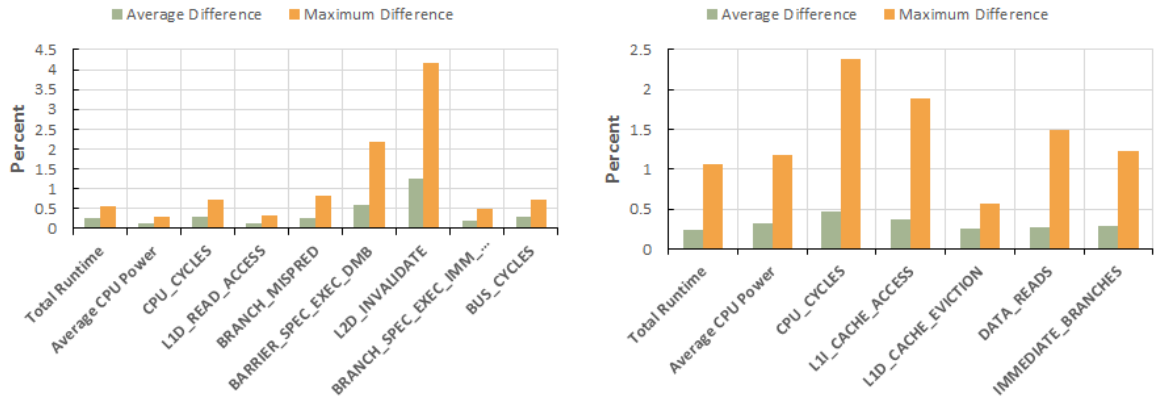
(b) ARM Cortex-A7 Models

Figure 6.3: **Comparison between the generated per-frequency level multi-thread Automatic, Collected and Intra-Core models for both processor types on the ODROID XU3 board.** The models are generated and validated by the custom methodology using the PARSEC 3.0 workload for the two processor types on the ODROID XU3 development platform with the eMMC card as OS driver. The *Average Percent Error* for the models at each available CPU frequency over all system configurations is presented on the graph. The model references corresponding to the legend are given in Table 6.3.

Figure 6.4 summarises the event variability of the events for each CPU cluster model. It demonstrates that good methodology stability is achieved even in the multi-thread case, with less that 5% maximum difference between workload executions for both processor types. This is mainly caused by some cache related events, which still contribute greatly to model accuracy, so should not be excluded.

Table 6.3: **Model reference and total average error for the data in Figure 6.3.** The model *Reference* is presented for each *Model Code* from the legend. *Total Average Error* values are obtained for each model on the two processor types (when available) by using their prediction error values across all frequency levels and system configurations.

Model Code	Reference	Total Average Error	
		ARM Cortex-A15	ARM Cortex-A7
(1)	Automatic	7.12%	5.46%
(2)	Collected	9.03%	5.68%
(3)	Intra-Core	0.65%	1.85%



(a) ARM Cortex-A15 Data (b) ARM Cortex-A7 Data

Figure 6.4: **Difference between multi-thread workload executions on both processor types on the ODROID XU3 board.** The data presented is the *Average* and *Maximum Percent Difference* between all of the PARSEC 3.0 workload executions for both processor types on the ODROID XU3 development platforms with the eMMC card as OS driver. The presented metrics are the workload *Total Runtime*, CPU *Average Power*, CPU *Average Temperature* and the collected hardware events for the *collected* model.

Unfortunately, according to the target accuracy metrics presented in Table 4.4 in Chapter 4, the ARM Cortex-A15 multi-threaded model has a higher error than required for successful use in real-time. The final ***collected*** models show 9.03% and 5.68% error for the Cortex-A15 and Cortex-A7, which for the former is higher than the 7.57% target. How these targets are defined and calculated is explained in detail in Subsection 4.3.3. Since the model is computed using the automated search approach under best possible conditions (low methodology overhead and reduced OS impact on experiment) the conclusion is that at this state of the work this is the best model the methodology can produce. A possible limitation is the fixed number of counters available in the PMU, which do not provide enough information/events to successfully describe the system behaviour in the multi-thread case. Another limitation could be the available events themselves. More specialised hardware events for the multi-threaded case could also provide useful information to the model. Methods to improve the multi-thread

per-frequency power models for the ARM Cortex-A15 remain a topic of future work. However accurate predictions of the average power can still be performed for both processor types as shown by the ***intra-core*** model.

6.3 Comparison to related work

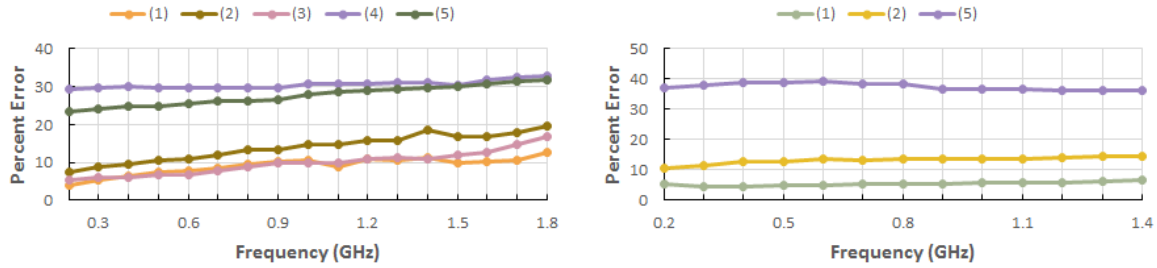
In this section the PMU event-based models are compared to other related work. It is demonstrated how the comprehensive automatic event search methods improve accuracy with respect to methods that require *intuitive choice*. A future topic of research is to explore if there is any other approach out there that will satisfy the accuracy requirement. In addition to the models described in section 5.8 an extra one is included in the comparison, namely the PMU event based power model described in Rethinagiri et al [56].

Rethinagiri [56] This model is labelled as equation code (6) in Figure 6.5 and Table 6.4. The interesting thing about this model is the heavy emphasis on Cache events. In the comparison the *single-core* version of their model is used. The authors have developed a **dual-core** specific model, tested on the Cortex-A9, however that model requires collecting the **IPC** and **L1_CACHE_MISS/REFILL** event counts for each Core. This could only be done in simulation since the dedicated PMU Unit in the SoC collects the events per CPU cluster, which includes all 4 available cores. Because of this individual cores cannot be isolated and the data needed for the **dual-core** model cannot be collected. When tested on the platform their *single-core* model performed quite poorly on both the Cortex-A15 and Cortex-A7 with average error above 25%. This is mainly due to the use of of cache-related events in their model and their high variability, especially on the multi-thread configurations.

$$\begin{aligned} P_{CPU} = \alpha_0 + \alpha_1 \times \frac{INST\_RETIRED}{CPU\_CYCLES} \\ + \alpha_2 \times (L1I\_CACHE\_REFILL + L1D\_CACHE\_REFILL) \\ + \alpha_3 \times L2D\_CACHE\_REFILL \end{aligned} \tag{6.3}$$

In addition to all the published models comparison is also done against the single-thread model as described in section 5 . Figure 6.5 and Table 6.4 clearly showcase that having a specifically trained model for the multi-thread case definitely improves accuracy. Compared to the related work the model still performs best, with Walker et al [62], labelled as equation code (3), coming in at a close second for the Cortex-A15. This again solidifies the result that automatic search methods definitely improve the accuracy of PMU based models. However due to the high complexity of multi-thread systems and the limited number of performance event counters available through the PMU the resulting models might not be accurate enough for the purposes of power prediction in real-time. Despite this the PMU based ***intra-core***

models can approximate average power for a given workload quite accurately and have an advantage to just using a physical characteristic like CPU frequency.



(a) ARM Cortex-A15 Models (b) ARM Cortex-A7 Models

Figure 6.5: **Comparison between the generated per-frequency level multi-thread collected model and other published work for both processor types on the ODROID XU3 board.** The models are generated and validated by the custom methodology using the PARSEC 3.0 workload for the two processor types on the ODROID XU3 development platform with the eMMC card as OS driver. The *Average Percent Error* for the models at each available CPU frequency over all system configurations is presented on the graph. The model references corresponding to the legend are given in Table 6.4.

Table 6.4: **Model reference and total average error for the data in Figure 6.5.** The model *Reference* is presented for each *Model Code* from the legend. *Reported Average Error* values are taken from the related publication (when available). *Total Average Error* values are obtained for each model on the two processor types (when available) by using their prediction error values across all frequency levels and system configurations.

Model Code	Reference	Reported Average Error		Total Average Error	
		ARM C.-A15	ARM C.-A7	ARM C.-A15	ARM C.-A7
(1)	Multi-Thread	-	-	9.03%	5.68%
(2)	Single-Thread	-	-	13.93%	13.23%
(3)	Walker	2.8%	3.8%	9.79%	-
(4)	Pricopi	1.2%	-	30.56%	-
(5)	Rethinagiri	2.4%	1.24%	27.74%	37.51%

6.4 Summary of results

This chapter shows how the work from the previous Chapter 5 is advanced and the methodology is adapted for multi-thread power model generation on the ODROID XU3 board. It is demonstrated that despite the great success achieved with the single-thread case, the system behaviour in the multi-thread scenario is much more complex.

Several different system configurations and models have been explored in addition to two extra methods for automatic event selection, namely ***top-down*** and ***exhaustive*** search, in order

to further improve model accuracy. Unfortunately the outcome is that the produced models for the ARM Cortex-A15 do not meet the target accuracy requirement for use in real-time. Despite this the generated models are compared to related work using the methodology and they still outperform them. This indicates that multi-thread power modelling for *big.LITTLE* has not been completely solved. Some possible solutions to improving the model for the Cortex-A15 are identified, but due to the limited time-frame of this project have not been pursued and are left as future work.

CHAPTER 7

HETEROGENEOUS MODELS

In this chapter the ***intra-core*** models in sections 5 and 6 are extended in order to be able to predict the average power between the two CPU core types - the ARM Cortex-A15 and ARM Cortex-A7 on the ODROID XU3 board. These new types of models are named ***inter-core***. Their function is to use the PMU events from one CPU cluster in order to predict the average power on the other cluster. The justification for this is to explore power models for heterogeneous systems and see if they can possibly be extended to power-aware schedulers. Current scheduling for the *big.LITTLE* system includes thread migration based on CPU utilisation percentage and CPU temperature. To the researcher's knowledge there is no other solution that utilises PMU events to predict average power.

Section 7.1 further develops this concept and introduces the key techniques to developing these models. It also gives an example scenario in order to illustrate their use and significance.

Afterwards Sections 7.2 and 7.3 detail the developed single-thread and multi-thread ***inter-core*** models and highlight their excellent performance.

Section 7.4 describes the research collaboration with Federal University of Rio Grande do Norte in Brazil. In this work their own type of heterogeneous models, which use a different type of mathematical expression and are not PMU event-based, are generated and validated using the adapted methodology on the ODROID XU3 board. This demonstrates that the developed approach is highly configurable and reliable.

7.1 Model purpose and significance

The basis of the models is calculating the **Event Scaling Factors** used in the ***intra-core*** models, but between the events of the two CPU clusters. Equations 7.1 and 5.19 are extended in order to showcase how this approach applies to the ***inter-core*** models. Equation 7.1 demonstrates

how the **Event Scaling Factors** factors are computed for event ev between $core\_A$ and $core\_B$ by using the mean values of the data samples. Equation 7.2 gives details on how to use the Cortex-A7(LITTLE) PMU events to predict the average power for the Cortex-A15(big). Figure 7.1 gives a theoretical example of how the heterogeneous models can be used to guide a power aware scheduler. The models are specifically designed to identify the most energy efficient system configuration at runtime, using information from only one of the processing clusters.

The models are generated in three steps. First the data from the train set from both CPU clusters is used to compute the **Event Scaling Factors**. Then the target CPU cluster train set is used to compute the power model using the selected PMU events. Finally the scaled PMU events from the test set of the initial CPU cluster are used in the computed power model. In order to evaluate the model accuracy, the model error, using the scaled events, is compared against the average power from the test set for the target CPU cluster. This is because there is no equivalent sensor reading for the sample power of the other CPU cluster for the events of the initial cluster.

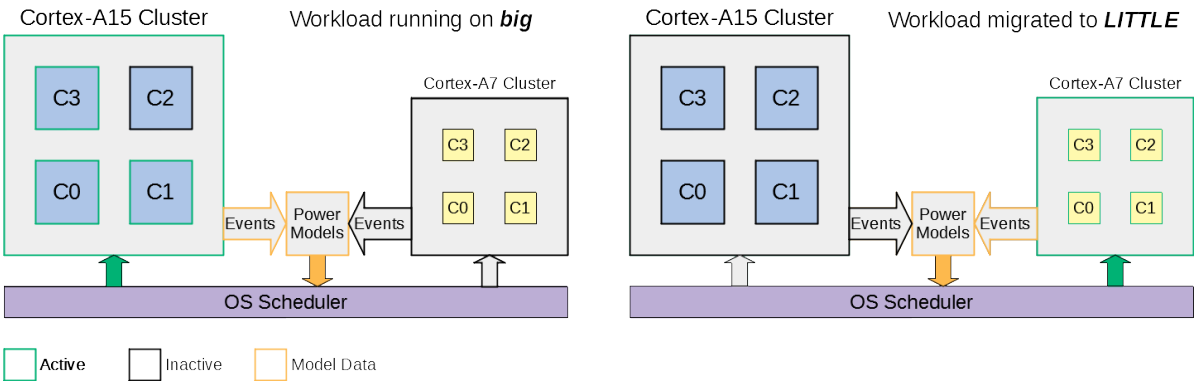


Figure 7.1: **Example usage of the heterogeneous models as a guide to a power-aware scheduler.** In this example the information from the power models is used to guide the OS scheduler. The workload is initially executed on 3 ARM Cortex-A15 cores. The model uses PMU event samples from the execution on the Cortex-A15 cluster to determine the CPU power usage for the Cortex-A7 cluster. It determines that the workload can run more efficiently on the Cortex-A7 cluster and the OS scheduler uses this information to migrate the workload threads.

For the models only the common events available to both the Cortex-A7 PMU and the Cortex-A15 PMU are used. This means the optimal models identified previously are not reused, but instead OLS is used to refit and validate specific dedicated ***inter-core*** models for both single-thread and multi-thread cases. A narrow list of 17 common PMU events between the Cortex-A15 and the Cortex-A7 is used, which is taken from the analysed list of low-variability events, used in Section 5.7. As with the ***per-frequency*** and ***intra-core*** models **CPU_CYCLES** is

the base event used in the model generation.

$$\begin{aligned} \frac{ev_{core\_A}}{ev_{core\_B}} &= \frac{\overline{ev_{core\_A}}}{\overline{ev_{core\_B}}} \\ ev_{core\_A}\frac{\overline{ev_{core\_B}}}{\overline{ev_{core\_A}}} &= ev_{core\_B} \\ \frac{\overline{ev_{core\_B}}}{\overline{ev_{core\_A}}} &= Event\ Scaling\ Factor \end{aligned} \tag{7.1}$$

$$\begin{aligned} P_{CPU\_big} = \alpha_0 + \alpha_1 \times ev_{1\_LITTLE} \times \frac{\overline{ev_{1\_big}}}{ev_{1\_LITTLE}} + \alpha_2 \times ev_{2\_LITTLE} \times \frac{\overline{ev_{2\_big}}}{ev_{2\_LITTLE}} \\ + ... + \alpha_n \times ev_{n\_LITTLE} \times \frac{\overline{ev_{n\_big}}}{ev_{n\_LITTLE}} \end{aligned} \tag{7.2}$$

Because of the nature of the ***inter-core*** model the events from any of the frequencies of one CPU cluster are scalable to the events of any the frequencies the other CPU cluster, therefore the models are able to predict between any two frequency levels. This explores the entire transition space of the heterogeneous system.

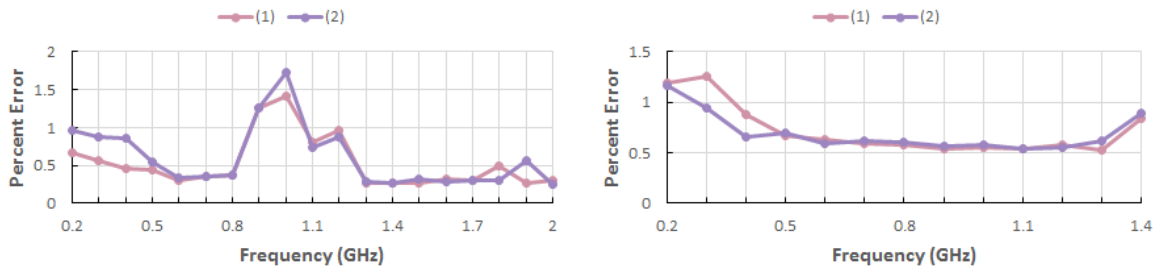
7.2 Single-thread case

The same methodology used for the ***per-frequency intra-core*** models detailed in the previous section is also used for the single-thread case. cBench is used as the workload with the benchmark split same as Appendix B.2. The same ***bottom-up*** automatic search algorithm that was identified to be the most optimal in Section 6.2 is used on the list of common PMU events. Equation 7.3 and 7.4 list the identified optimal events for the LITTLE to big and big to LITTLE models respectively. It is interesting to note that more events are needed from the LITTLE core to predict the power of the big core than vice versa. The events are quite different between the two situations and different from the per-frequency single-thread model.

$$\begin{aligned} P_{CPU\_A15} = &\alpha_0 + \alpha_1 \times CPU\_CYCLES_{\_A7} \times \frac{\overline{CPU\_CYCLES_{\_A15}}}{\overline{CPU\_CYCLES_{\_A7}}} \\ &+ \alpha_2 \times EXCEPTION\_RETURN_{\_A7} \times \frac{\overline{EXCEPTION\_RETURN_{\_A15}}}{\overline{EXCEPTION\_RETURN_{\_A7}}} \\ &+ \alpha_3 \times BRANCH\_MISPRED_{\_A7} \times \frac{\overline{BRANCH\_MISPRED_{\_A15}}}{\overline{BRANCH\_MISPRED_{\_A7}}} \\ &+ \alpha_4 \times L2D\_CACHE\_WB_{\_A7} \times \frac{\overline{L2D\_CACHE\_WB_{\_A15}}}{\overline{L2D\_CACHE\_WB_{\_A7}}} \\ &+ \alpha_5 \times BUS\_ACCESS_{\_A7} \times \frac{\overline{BUS\_ACCESS_{\_A15}}}{\overline{BUS\_ACCESS_{\_A7}}} \end{aligned} \tag{7.3}$$

$$
\begin{aligned}
P_{CPU\_A7} = \alpha_0 &+ \alpha_1 \times CPU\_CYCLES_{\_A15} \times \frac{\overline{CPU\_CYCLES_{\_A7}}}{\overline{CPU\_CYCLES_{\_A15}}} \\
&+ \alpha_2 \times L1I\_CACHE\_ACCESS_{\_A15} \times \frac{\overline{L1I\_CACHE\_ACCESS_{\_A7}}}{\overline{L1I\_CACHE\_ACCESS_{\_A15}}} \\
&+ \alpha_3 \times BRANCH\_PRED_{\_A15} \times \frac{\overline{BRANCH\_PRED_{\_A7}}}{\overline{BRANCH\_PRED_{\_A15}}}
\end{aligned}
\tag{7.4}
$$

The results from the methodology are detailed in Figure 7.2 and Table 7.1. It can be seen that the models perform quite well for their scenarios with the average error being very low. Bear in mind this is due to the fact that the models are only validated against the average power for the test set, so some variation, which can influence the model, is lost. However this simplified model can be successfully used to predict average power at frequency level. This can be used in a scheduling algorithm, where the decision to switch between the different processors of the heterogeneous system can be influenced by the coarse power model. The models have the ability to predict the average power from the events between any two frequencies of the two cores, due to the event scaling technique. The reported error percent number is the average of all the many to one predictions for that frequency of the target CPU cluster. This means that the ability to predict the average power of the target CPU is measured from all possible points. For example, the reported error for 1.4 GHz for the Cortex-A7 is the average of the model errors from using the scaled Cortex-A15 PMU events at 0.2, 0.3, 0.4 ... to 2.0 GHz, so its an average of 18 numbers. The models achieve less than 1% average error in the validated final model for both ***inter-core*** single-thread situations, which is more than satisfactory.



(a) LITTLE to big Models

(b) big to LITTLE Models

Figure 7.2: **Comparison between the generated per-frequency level single-thread inter-core Automatic and Collected models for both processor transition types on the ODROID XU3 board.** The models are generated and validated by the custom methodology using the cBench workload for the two processor transition types on the ODROID XU3 development platform with the eMMC card as OS driver. The *Average Percent Error* at each available CPU frequency for the models is presented on the graph. The model references corresponding to the legend are given in Table 7.1.

Table 7.1: **Model reference and total average error for the data in Figure 7.2.** The model *Reference* is presented for each *Model Code* from the legend. *Total Average Error* values are obtained for each model on the two processor types (when available) by using their prediction error values across all frequency levels.

Model Code	Reference	Total Average Error	
		LITTLE to big	big to LITTLE
(1)	Automatic	0.53%	0.72%
(2)	Collected	0.60%	0.69%

7.3 Multi-thread case

Several key milestones have been accomplished during this research with the main focus being the development of a robust methodology to generate accurate run-time power models on a heterogeneous system. In order to demonstrate the effectiveness of the unique approach to modelling custom power-models have been developed and their results have been compared to other published work. The main focus is using PMU events and OLS linear regression to try and capture system behaviour. This approach has been proven to be effective in non-heterogeneous systems before for various other architectures. This means that the methodology can be adapted to another architecture and a different set of system events, but the underlying methods would still work well.

Continuing from the previous work, described in Nikov et al [21], the ***per-frequency level*** power models are further developed and it is demonstrated how they can be used to predict the average power with coarse sampling granularity between any two CPU frequencies. The previous single-thread workload and the ODROID-XU3 development platform are reused, but drastically improve the methodology by using control tools that greatly reduce measurement overhead and platform variability. The full list of available PMU events is also explored for the model and automatic search algorithms are also used to identify the best possible models for the two CPU core types of the platform. Afterwards a parallel workload is modified and used with the improved methodology to develop models for the multi-thread case. Fine system control mechanisms are used to explore all available scenarios (1 core, 2 cores, 3 cores and 4 cores) in order to identify the best model that fits all of them. Several different optimisation criteria for the power model are also explored, including standard deviation of the model error as well as PMU event cross correlation. A key limitation with the PMU event based model has been identified reducing its accuracy for the multi-thread case.

The most interesting contribution of this work, however is the unique approach to exploiting the capabilities of a heterogeneous system. Power models have been developed that use the PMU events of one CPU core type to predict the average power of another. This is tested on the platform with good success, achieving less than 5% error even for the multi-thread case. However, this is done with sacrificed real-time usability, since fewer measurements are used to

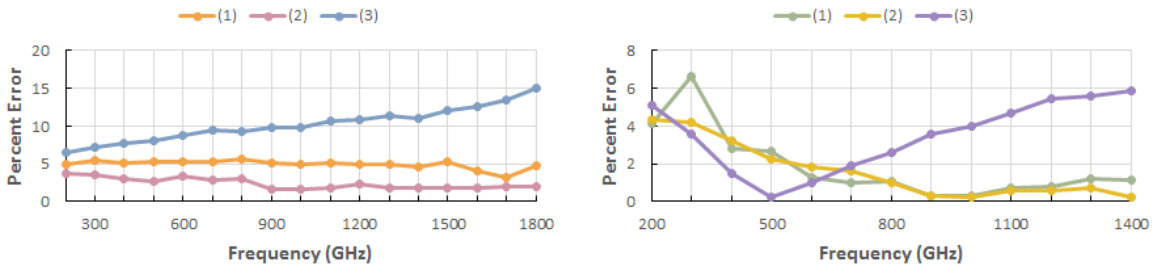
test the model. This model is designed only to give a general prediction of the average power, instead of at small intervals like the ***per-frequency runtime*** models.

This work offers some key insights into predicting the behaviour of modern heterogeneous systems and can serve as a stepping stone for further research into intelligent advanced power-aware scheduling. It is believed that this work can help solve the issue with energy efficiency in the Mobile and IoT industry.

The multi-thread ***inter-core*** model uses the same ideas from the single-thread case as described in the previous section, but with the multi-thread workload and core configuration specifics. The same PARSEC 3.0 train and test set split is used in addition to reusing the same technique of isolating the 1 core, 2 cores, 3 cores and 4 Cores situations. For the construction of the model the ***bottom-up*** automatic search algorithm is used, like in the previous Section 7.2. The data from all 4 core configurations is used as one big set, so one set of events and coefficients are obtained for each frequency level. It is noted that the generated models use the **num_cores** event of either the target core or the core the PMU events are collected from. In the situation where testing is done using all possible core configuration and only the average error of all situations is of interest, then just using the number of enabled cores of the origin core, for which the PMU events are collected, is sufficient. However in a specialised case where predicting from a specific scenario is required, the scaled events from the initial core and the number of enabled cores (for the scenario being tested) of the target core will need to be used. This is the only intricate part of the ***inter-core*** multi-thread models. Equations 7.5 and 7.6 detail the finalized models for the LITTLE to big and big to LITTLE scenario respectively.

$$
\begin{aligned}
P_{CPU\_A15} = {} & \alpha_0 + \alpha_1 \times num\_cores\_A7 / num\_cores\_A15 \\
& + \alpha_2 \times CPU\_CYCLES_{\_A7} \times \frac{\overline{CPU\_CYCLES_{\_A15}}}{\overline{CPU\_CYCLES_{\_A7}}} \\
& + \alpha_3 \times EXCEPTION\_TAKEN_{\_A7} \times \frac{\overline{EXCEPTION\_TAKEN_{\_A15}}}{\overline{EXCEPTION\_TAKEN_{\_A7}}} \\
& + \alpha_4 \times L2D\_CACHE\_WB_{\_A7} \times \frac{\overline{L2D\_CACHE\_WB_{\_A15}}}{\overline{L2D\_CACHE\_WB_{\_A7}}} \\
& + \alpha_5 \times BRANCH\_MISPRED_{\_A7} \times \frac{\overline{BRANCH\_MISPRED_{\_A15}}}{\overline{BRANCH\_MISPRED_{\_A7}}} \\
& + \alpha_6 \times EXCEPTION\_RETURN_{\_A7} \times \frac{\overline{EXCEPTION\_RETURN_{\_A15}}}{\overline{EXCEPTION\_RETURN_{\_A7}}}
\end{aligned}
\tag{7.5}
$$

$$
\begin{aligned}
P_{CPU\_A7} = & \alpha_0 + \alpha_1 \times num\_cores\_A15/num\_cores\_A7 \\
& + \alpha_2 \times CPU\_CYCLES_{\_A15} \times \frac{\overline{CPU\_CYCLES_{\_A7}}}{\overline{CPU\_CYCLES_{\_A15}}} \\
& + \alpha_3 \times EXCEPTION\_TAKEN_{\_A15} \times \frac{\overline{EXCEPTION\_TAKEN_{\_A7}}}{\overline{EXCEPTION\_TAKEN_{\_A15}}} \\
& + \alpha_4 \times EXCEPTION\_RETURN_{\_A15} \times \frac{\overline{EXCEPTION\_RETURN_{\_A7}}}{\overline{EXCEPTION\_RETURN_{\_A15}}}
\end{aligned}
\tag{7.6}
$$



(a) LITTLE to big Models

(b) big to LITTLE models

Figure 7.3: **Comparison between the generated per-frequency level multi-thread inter-core Automatic, Collected and Single-Thread Events models for both processor transition types on the ODROID XU3 board.** The models are generated and validated by the custom methodology using the PARSEC 3.0 workload for the two processor transition types on the ODROID XU3 development platform with the eMMC card as OS driver. The *Average Percent Error* for the models at each available CPU frequency over all system configurations is presented on the graph. The model references corresponding to the legend are given in Table 7.2.

Table 7.2: **Model reference and total average error for the data in Figure 7.3.** The model *Reference* is presented for each *Model Code* from the legend. *Total Average Error* values are obtained for each model on the two processor types (when available) by using their prediction error values across all frequency levels and system configurations.

Model Code	Reference	Total Average Error	
		LITTLE to big	big to LITTLE
(1)	Automatic	4.94%	1.85%
(2)	Collected	2.39%	1.62%
(3)	Single-Thread Events	10.20%	3.47%

The resulting model equations are validated and compared against the model events identified in the single-thread case. The results are presented in Figure 7.3 and Table 7.2. It is evident that for the multi-thread case, dedicated specific models have a much better accuracy that just reusing the events for the ***inter-core*** single-thread models. It can also be noted that the validated model performs better than the one computed from the concatenated events. In contrast to the ***per-frequency*** and ***intra-core*** models in previous chapters, collecting the PMU

events concurrently on the hardware results in a smoother graph. This is a result of the better mapping of the PMU samples to the power sensor samples in the coarse-grained models due to lower variation. The multi-thread ***inter-core*** models are accurate enough to distinguish between the different energy levels successfully, so the coarser granularity in this case helps improve the results of the multi-thread ***per-frequency*** and ***intra-core*** models in Section 6.2. It can be observed that it is harder to predict the Cortex-A15 power from the Cortex-A7 PMU events that the other way around, since the Cortex-A15 has a much wider power range and requires more than the available 5 PMU event registers to fully capture that. Overall the performance of the heterogeneous models is quite satisfactory with both the single-thread and the multi-thread ones having low enough error to facilitate a possible use in a PMU event based power-aware scheduler.

7.4 Collaboration

The final part of this research consists of a collaboration with researchers from Federal University of Rio Grande do Norte in Brazil. This is part of an ongoing two-year joint project between university of Bristol and Universidade Federal do Rio Grande do Norte exploring energy efficiency in parallel systems. They have developed a purely mathematical model describing the relationship between the two core types, which can be used to predict the energy consumption at different energy levels. The developed methodology on the ODROID-XU3 platform was proposed in order to validate their methods.

The aim is to try and predict the behaviour of the system with all 8 CPU cores working at the same time. Since the CPU consists of two clusters, each using a different type of CPU core and operating at different frequencies. The initial difficulty was to identify a workload that will scale for the two types of cores. This is essential, since just using a standard heterogeneous workload causes the faster Cortex-A15 CPU cores to wait for the Cortex-A7 CPU cores to finish. This resulted in hardly any improvement compared to just using the Cortex-A15, despite 4 more CPU cores being utilised. In the end a modified version of a suitable benchmark from the PARSEC 3.0 workload was optimised for dynamic heterogeneous workloads, namely *parsec.blackscholes*. The energy/frequency relationship for the two benchmarks used for the experiment can be found on Appendix E. The same non-linear relationship, observed in the research experiments with cBench and PARSEC 3.0, is clearly present, but this time with respect to changing the frequency of the two CPU core clusters independently. This provides another strong point in favour of complex power models. The resulting performance and energy model is validated using coarse measurements, one for each frequency level per benchmark. The data is presented in Figure 7.4. Overall, the model performance is satisfactory with 1.6% prediction error for the performance and 4.6% prediction error for the energy consumption model.

This work has been presented as *The energy consumption benefits of DynamIQ for heterogeneous*

parallel workloads [22] at **ARM Research Summit 2017**. Further details about the mathematical expressions and calibration methodology are contained in the poster document.

Figure 7.4: **Performance of the generated heterogeneous workload execution time and CPU energy models on the ODROID XU3 board.** The models are generated and validated by the custom methodology using a modified *parsec.blackscholes* benchmark as workload for all eight cores working at the same time on the ODROID XU3 development platform with the eMMC card as OS driver. The data presented is the *measured* and *estimated* workload *Execution time* and CPU *Energy*. The presented *estimated* values are the predictions of the custom generated models for various frequency pairs of the two processing clusters.

7.5 Summary of Results

This chapter describes the most significant contribution of this research, namely the development of the ***per-frequency intra-core*** power models. These models can predict average power between processing clusters. A modified version of the PMU event scaling technique from chapter 5 is used in order to achieve this. The models can only predict power at a coarse level, but it is argued that these models serve as a general guide to the most optimal energy level for switching. Once the task has been migrated the runtime per-frequency models can be used again. The key concepts and principles are demonstrated thoroughly. The generated models for the single-thread and multi-thread cases are also presented in detail.

The recent collaboration with Federal University of Rio Grande do Norte is also outlined.

Their mathematical multi-core model is validated on the ODROID XU3 platform using the developed methodology. This shows that the developed workflow is not constrained to just one type of modelling, but is flexible enough to b used in a different context.

CHAPTER 8

CONCLUSIONS

The final chapter in this thesis summarises the primary objectives of this work in Section 8.1. The next Section 8.2 lists the 6 key research outcomes and contributions to the field of power modelling. An outline for continuation of this research is also presented in Section 8.3, highlighting any outstanding problems and milestones as future work. The final remarks of the researcher are included in the closing Section 8.4.

8.1 Summary of research objectives

The complicated energy demands of the Mobile industry offer ample research objectives, thus drawing in many researchers. Literature survey has identified many contributions out there that propose possible solutions for improving device energy efficiency, however most of them are very niche and are not suitable for widespread adoption. A commercially available approach for solving the increasing performance demand, while still considering energy efficiency are heterogeneous embedded platforms. An already established solution is the ARM *big.LITTLE* SoC which combines a high-performance processor with a power-efficient one, connected via a Cache Coherent Interconnect. This allows quick dynamic task migration to adjust the system performance based on demand while minimising energy consumption. Such heterogeneous computing techniques are nothing new, but applying them to the embedded market seems to be the way forward.

This research is aimed at evaluating *big.LITTLE* as a platform for energy-efficient computing. The existing software scheduler, developed by ARM for the system, relies on CPU utilisation thresholds to identify when to migrate tasks between the two processors. It is argued that a dedicated scheduler which optimises energy usage metrics can achieve better power efficiency

and utilisation of the system capabilities.

This work investigates possible approaches and identifies that hardware performance counter-based power models can provide the necessary metrics that can help guide at run-time such a novel scheduler. From this point on this work involves developing a methodology for fitting and validation of PMU-based power models on *big.LITTLE* platforms. After comparing the initial work to other relevant power models, it was concluded that there is no one-size-fits-all solution, so the key objective becomes to make the proposed methodology robust, yet flexible enough to be applicable to all possible scenarios. Therefore, the workloads, the model generation algorithms and even the experiment data collection scripts can be easily reconfigured.

In order to validate the methodology a number of accurate power models have been produced and thoroughly compared against the state-of-the-art. Novel techniques were used to develop models that are able to capture the heterogeneous behaviour of the platform by predicting the average power of one core, while using the PMU events of the other. These models can be utilised to improve the *big.LITTLE* heterogeneous software scheduler, which remains an area of future work.

8.2 Key contributions

This research has achieved several critical milestone, both in terms of the developed methodology as well as the validated power models. A list of the most important contributions is given below:

Accurate runtime per-frequency models - The first breakthrough was achieved when it was realized that by computing individual model coefficients for each frequency level the model accuracy improved significantly. This is due to the fact that the PMU events used for model generation can vary greatly between energy levels. Restricting these allows the linear regression algorithm to fit a much tighter set of data. This allowed the developed models to go from 25% and 20% down to 8% and 4% average error on the ARM Cortex-A15 and ARM Cortex-A7 processor respectively. These models we subsequently compared to other related work on *big.LITTLE* and found to be more accurate.

Automated model generation using custom algorithms - The second milestone is the adoption of automated techniques to try and improve the models even further. To achieve this an advanced method of collecting the system information and then combining the experiment data in one big data set was developed. Several different machine learning algorithms and optimisation criteria, such as event cross-correlation were studied. It is shown that automated approaches, though more difficult to implement initially, greatly outperform traditional *intuitive* techniques for PMU event selection.

Critical analysis of platform variability - During the course of this research factors contributing to variability of platform energy consumption and performance were also explored. Manufacturing process can vary greatly between different die implementations of the same SoC, which can impact power consumption significantly. Other factors were also studied, such as differences in the types of flash memory cards, commonly used in ES. This knowledge was used to minimize the utilized development platform incongruity to around 1% average power variance between experiment runs.

Evaluation of single-thread and multi-thread power models - Single and multi-thread use cases were independently analysed and the results were compared. Using the latest optimisation techniques developed in this research, models with less than 3% average error for the single-thread case were designed. However it was identified that the multi-threaded scenario is much more complicated resulting in an error of between 9% and 6% on the Cortex-A15 and Cortex-A7 respectively. Analysis indicates that the limited number of PMU event counters available for the *big.LITTLE* platform are not sufficient to capture multi-thread behaviour on the ARM Cortex-A15. This is in contrast to a number of published models, which claim very high accuracy using PMU events, specifically on multi-thread platforms. The author is confident in the rigorousness of the proposed approach and willing to challenge the existing notion that multi-threaded power modelling for the *big.LITTLE* platform has been solved.

Intra and inter-core power models - A specific technique to scale PMU events for use in model equations was developed. The key principle was to use the initial ***per-frequency*** level models to obtain average power information for any frequency level by using the PMU events collected at run-time, scaling them and inserting them in the model equation of the target frequency level. This is a way to show that the models can be used for DVFS on the same processor type, by reusing the same PMU event sample to obtain estimations of the average CPU power at all frequency levels. This information can be used to identify the most optimal CPU frequency for efficient task execution and apply DVFS. Due to the nature of the technique, the scaled events are able to approximate the average power for the target frequency, in contrast to the ***per-frequency*** models, which can predict dynamic power at every PMU event sample interval. Because the granularity of the prediction is coarse, providing only a single value for a frequency level, a prediction error for average power of around 1% for both for the single-thread and multi-thread case was achieved. These models are called ***intra-core***, because of the ability to predict average power between the frequency levels of the same processing core type. Afterwards this technique was extended to scale the PMU events of the two processor types on the *big.LITTLE* SoC. Distinct models were trained to use runtime information from one cluster to predict the workload behaviour of the other. This is also done at a very coarse estimation of average power for every frequency level, so it

only serves as a rough guide. The results show model error at 1% for the single-thread case and around 2% for the multi-thread case. The models are identified as ***inter-core***, because of the ability to predict average power between the two processing core types. In contrast to the ***intra-core*** models which reuse the PMU events and coefficients of the ***per-frequency*** models, the ***inter-core*** models are fitted using a list of common PMU events for both processor types on the *big.LITTLE* platform using the developed automated search algorithms, resulting in completely different models.

Methodology flexibility and reconfigurability - Finally the flexibility and reconfigurability of the methodology was demonstrated though a collaboration with Federal University of Rio Grande do Norte in Brazil, in which their multi-core models were validated using the developed approach on the HARDKERNEL ODROID-XU3 platform. The researchers use a mathematical approach to derive the energy usage and performance of the whole system and report 1.6% prediction error for performance and 4.6% error for energy consumption. This shows the researched methodology is not constrained to just PMU-based power modelling, but can be adjusted to other scenarios.

8.3 Future work

The most obvious continuation avenue of this work is the extension of the ***intra*** and ***inter-core*** power models into a power-aware scheduling solution for *big.LITTLE*. The models have been designed to be quick and responsive and to provide usable information in the context of scheduling. This research has given some examples of how the models can be extended for different scenarios.

There are also quite a few interesting questions that have been left unanswered during this research. The possibility of revisiting the multi-thread case and identifying how to improve model accuracy is the most obvious one. A possible approach is to use specific mechanistic information of the platform, such as pipeline depth and other architectural details to provide even more information to the model. Other research has shown that this approach is effective [55] [58], but it comes at the cost of increased model complexity and platform specificity. Another argument would be to extend the ARM PMU for future *big.LITTLE* implementations and allow for the collection of more counters concurrently. In addition, research could be directed towards adding support for specific energy-usage-capturing events for the multi-threaded scenario, not included in the current available list. This would require a very deep understanding of the ARM architecture and heavy use of simulators to explore scenarios and events.

Another interesting experiment to revisit would be to reproduce the platform variability study with the more stable eMMC cards in an effort to produce a model that can capture the platform variability successfully, perhaps with on-demand refitting. This would be beneficial

in trying to develop a solution for the Mobile market, since the model would be able to adjust itself and overcome manufacturing and platform variations that cause deviation in CPU energy usage.

8.4 Final remarks

This work details four years of research in the area of power modelling and analysis for the ARM *big.LITTLE* SoC. Initial success was found by using novel techniques to produce accurate single-thread models for the HARDKERNEL ODROID-XU3 development platform. Capturing of the multi-thread behaviour on the target board proved to be a much greater challenge even when validating other published models, despite them having reported very high accuracy. It seems there is no consistent metric for power modelling and any model can look good under the right circumstances. It is understood that due to platform, workload and technology variation a universal solution is not achievable at this moment. Therefore this work has focused on developing a methodology and use the generated models as a case study, rather than a definitive answer. Nevertheless the research has produced several highly accurate models, including ones that can predict average CPU power using information between the two processor types on the development board, which is a novel approach to power modelling on HES. The methodology and techniques are designed to be extended to a power-aware scheduler as a future continuation of this work.

APPENDIX A

AVAILABLE PMU EVENTS FOR THE *big.LITTLE* SYSTEM

Event Name	Raw ID	
	Cortex-A15	Cortex-A7
SW_INCR	r000	r000
L1I_CACHE_REFILL	r001	r001
L1I_TLB_REFILL	r002	r002
L1D_CACHE_REFILL	r003	r003
L1D_CACHE_ACCESS	r004	r004
L1D_TLB_REFILL	r005	r005
DATA_READS	-	r006
DATA_WRITES	-	r007
INST_RETIRED	r008	r008
EXCEPTION_TAKEN	r009	r009
EXCEPTION_RETURN	r00A	r00A
CID_WRITE_RETIRED	r00B	r00B
SW_CHANGE_PC	-	r00C
IMMEDIATE_BRANCHES	-	r00D
PROCEDURE_RETURNS	-	r00E
UNALIGNED_LOAD_STORE	-	r00F
BRANCH_MISPRED	r010	r010
CPU_CYCLES	r011	r011
BRANCH_PRED	r012	r012
DATA_MEM_ACCESS	r013	r013

Event Name	**Raw ID**	
	Cortex-A15	**Cortex-A7**
L1I_CACHE_ACCESS	r014	r014
L1D_CACHE_WB	r015	r015
L2D_CACHE_ACCESS	r016	r016
L2D_CACHE_REFILL	r017	r017
L2D_CACHE_WB	r018	r018
BUS_ACCESS	r019	r019
LOCAL_MEMORY_ERROR	r01A	-
INST_SPEC_EXEC	r01B	-
TTBR_WRITE_RETIRED	r01C	-
BUS_CYCLES	r01D	r01D
L1D_READ_ACCESS	r040	-
L1D_WRITE_ACCESS	r041	-
L1D_READ_REFILL	r042	-
L1D_WRITE_REFILL	r043	-
L1D_WB_VICTIM	r046	-
L1D_WB_CLEAN_COHERENCY	r047	-
L1D_INVALIDATE	r048	-
L1D_TLB_READ_REFILL	r04C	-
L1D_TLB_WRITE_REFILL	r04D	-
L2D_READ_ACCESS	r050	-
L2D_WRITE_ACCESS	r051	-
L2D_READ_REFILL	r052	-
L2D_WRITE_REFILL	r053	-
L2D_WB_VICTIM	r056	-
L2D_WB_CLEAN_COHERENCY	r057	-
L2D_INVALIDATE	r058	-
BUS_READ_ACCESS	r060	r060
BUS_WRITE_ACCESS	r061	r061
BUS_NORMAL_ACCESS	r062	-
BUS_NOT_NORMAL_ACCESS	r063	-
BUS_NORMAL_ACCESS_2	r064	-
BUS_PERIPH_ACCESS	r065	-
DATA_MEM_READ_ACCESS	r066	-
DATA_MEM_WRITE_ACCESS	r067	-
UNALIGNED_READ_ACCESS	r068	-

Event Name	Raw ID	
	Cortex-A15	Cortex-A7
UNALIGNED_WRITE_ACCESS	r069	-
UNALIGNED_ACCESS	r06A	-
INST_SPEC_EXEC_LDREX	r06C	-
INST_SPEC_EXEC_STREX_PASS	r06D	-
INST_SPEC_EXEC_STREX_FAIL	r06E	-
INST_SPEC_EXEC_LOAD	r070	-
INST_SPEC_EXEC_STORE	r071	-
INST_SPEC_EXEC_LOAD_STORE	r072	-
INST_SPEC_EXEC_INTEGER_INST	r073	-
INST_SPEC_EXEC_SIMD	r074	-
INST_SPEC_EXEC_VFP	r075	-
INST_SPEC_EXEC_SOFT_PC	r076	-
BRANCH_SPEC_EXEC_IMM_BRANCH	r078	-
BRANCH_SPEC_EXEC_RET	r079	-
BRANCH_SPEC_EXEC_IND	r07A	-
BARRIER_SPEC_EXEC_ISB	r07C	-
BARRIER_SPEC_EXEC_DSB	r07D	-
BARRIER_SPEC_EXEC_DMB	r07E	-
IRQ_EXCEPTION_TAKEN	-	r086
FIQ_EXCEPTION_TAKEN	-	r087
EXTERNAL_MEMORY_REQUEST	-	r0C0
NONCACHE_EXTERNAL_MEMORY_REQUEST	-	r0C1
PREFETCH_LINEFILL	-	r0C2
PREFETCH_LINEFILL_DROPPED	-	r0C3
ENTERING_READ_ALLOC	-	r0C4
READ_ALLOC	-	r0C5
Reserved	-	r0C6
ETM_EXT_OUT_0	-	r0C7
ETM_EXT_OUT_1	-	r0C8
DATA_WRITE_STALL	-	r0C9
DATA_SNOOPED	-	r0CA

APPENDIX B

POWER MODELLING WORKLOADS TRAIN TEST SET SPLITS

B.1 cBench Initial *Partial* Uneven Workset Split

	Train Set	Test Set
Benchmarks	automotive_qsort1	automotive_bitcount
	automotive_susan_c	bzip2d
	automotive_susan_e	consumer_jpeg_c
	automotive_susan_s	network_dijkstra
	bzip2e	office_ghostscript
	consumer_jpeg_d	security_blowfish_d
	consumer_tiff2bw	telecom_adpcm_c
	consumer_tiff2rgba	
	consumer_tiffdither	
	consumer_tiffmedian	
	network_patricia	
	office_ispell	
	office_rsynth	
	office_stringsearch1	
	security_blowfish_e	
	security_pgp_d	
	security_pgp_e	
	security_rijndael_d	
	security_rijndael_e	
	security_sha	

	Train Set	Test Set
	telecom_adpcm_d	
	telecom_CRC32	
	telecom_gsm	

B.2 cBench Workset Split

	Train Set	Test Set
Benchmarks	telecom_CRC32	consumer_jpeg_d
	consumer_tiffdither	security_blowfish_e
	telecom_gsm	security_pgp_d
	bzip2d	office_ghostscript
	consumer_tiffmedian	network_dijkstra
	consumer_jpeg_c	security_blowfish_d
	office_stringsearch1	automotive_susan_e
	office_ispell	automotive_qsort1
	automotive_susan_s	automotive_bitcount
	security_pgp_e	security_rijndael_e
	telecom_adpcm_d	telecom_adpcm_c
	automotive_susan_c	bzip2e
	security_sha	network_patricia
	security_rijndael_d	office_rsynth
	consumer_tiff2rgba	consumer_tiff2bw

B.3 PARSEC Workset Split

	Train Set	Test Set
Benchmarks	parsec.facesim	parsec.dedup
	splash2x.radiosity	parsec.freqmine
	splash2x.raytrace	parsec.streamcluster
	splash2x.water_nsquared	splash2x.barnes
		splash2x.fmm

APPENDIX C

PSEUDOCODE SAMPLES

C.1 Experiment Data Concatenation

```
1: procedure DATACAT(F1, F2, ..., FN)
2:     F1Line,F2Line,...FNLine ← 2        ▷ First line of data file is column header, merging starts from second line (first line with data).
3:     while F1Line < EOFLine do          ▷ First file is used as synchronization anchor.
4:         READ F1Run,F1Freq,F1Bench      ▷ The three synchronisation points. Only points from the same stage of experiment can be merged.
5:         for i in 2 to N do             ▷ Go through data files one at a time.
6:             READ FiRun,FiFreq,FiBench  ▷ Extract synchronisation points at FiLine.
7:             if F1Run! = FiRun then     ▷ Try and synchronise the experiment Run.
8:                 F1Sync ← READ next sync F1Line ▷ Next F1Line at which Run matches.
9:                 FiSync ← READ next sync FiLine ▷ Next FiLine at which Run matches.
10:                if F1Sync = FiSync = NULL then
11:                    RETURN             ▷ If no possible synchronization, end procedure.
12:                else if F1Sync = NULL then   ▷ If only FiSync found.
13:                    FiLine ← FiSync    ▷ Move FiLine pointer ahead.
14:                    BREAK              ▷ Break and synchronize next file.
15:                else if FiSync = NULL then   ▷ If only F1Sync found.
16:                    F1Line ← F1Sync    ▷ Move F1Line pointer ahead.
17:                    BREAK              ▷ Break and synchronize next file.
18:                else                   ▷ Find pointer synchronization distance for both files.
19:                    F1Diff ← F1Sync − F1Line
20:                    FiDiff ← FiSync − FiLine
21:                    if F1Diff < FiDiff then   ▷ Find shortest synchronization distance.
22:                        F1Line ← F1Sync   ▷ If F1Sync shorter, move F1Line.
23:                        BREAK          ▷ Break and synchronize next file.
```

```
                else
                    FiLine ← FiSync                    ▷ If FiSync shorter, move FiLine.
                    BREAK                              ▷ Break and synchronize next file.
                end if
            end if
        end if
        if F1Freq! = FiFreq then                       ▷ After synchronizing data Run, do Freq.
            ...
        end if
        if F1Bench! = FiBench then                     ▷ After also synchronizing data Freq, do Bench.
            ...
        end if
    end for
    for i in 1 to N do
        TempData ← READ FiData                         ▷ Get synchronized data points from all files.
        FiLine ← FiLine + 1 READ FiData                ▷ Move data pointers to next sample.
    end for
    SyncData ← AVERAGE TempData                        ▷ Average the physical data (temperature,
current, power) from synchronized files and concatenate the PMU events.
    WRITE SyncData                                     ▷ Store processed data point.
  end while
end procedure
```

C.2 Per-Frequency Model Generation

```
1: procedure PERFREQGEN(File1,FIle2,List)
2:   READ FreqList from File1                          ▷ Extract FreqList from File1.
3:   for F in FreqList do      ▷ Individual model fitting for every frequency in FreqList.
4:     Train ← READ from File1                         ▷ Extract train set data from File1
5:     Test ← READ from File2                          ▷ Extract test set data from File1
6:     Model ← FIT(Train,F,List)                       ▷ Fit model using event List on Train data.
7:     Error ← VALIDATE (Test,F,Model)                 ▷ Validate fitted model on Test data.
8:   end for
9:   ModelError ← AVERAGE Error ▷ Final model performance metric is average Error
over all frequencies in FreqList.
10:  RETURN ModelErrror
11: end procedure
```

C.3 Bottom-Up Automatic Event Selection

```
1: procedure BOTUPEVENTSEL(File1, File2, Pool,Num)
2:     READ FreqList from File1                          ▷ Extract FreqList from File1.
3:     List ← NULL                                       ▷ Initiate event List.
4:     while Num > 0 do                                  ▷ Continue search until Num events reached.
5:         EventAdd ← NULL                               ▷ Improving event identifier EventAdd.
6:         for TempEvent in Pool do                      ▷ Search all events in Pool.
7:             TempList ← List + TempEvent               ▷ Temporary event list TempList using all
   events in List and event under test TempEvent.
8:             for F in FreqList do                      ▷ Use PerFreqGen to fit and validate model.
9:                 Train ← READ from File1
10:                Test ← READ from File2
11:                Model ← FIT(Train,F,TempList)
12:                Erorr ← VALIDATE(Test,F,Model)
13:            end for
14:            TempError ← AVERAGE Error
15:            if MinError = NULL then                   ▷ Use first event metrics as initial reference.
16:                EventAdd ← TempEvent
17:                MinError ← TempError
18:            else
19:                if TempError < MinError then          ▷ Compare against stored metrics.
20:                    EventAdd ← TempEvent              ▷ Overwrite if tested event improves model.
21:                    MinError ← TempError
22:                end if
23:            end if
24:        end for
25:        if EventAdd ≠ NULL then                       ▷ After searching all events, look for new EventAdd.
26:            List ← List + EventAdd                    ▷ Add to final event List.
27:            Pool ← Pool − EventAdd                    ▷ Remove from event Pool.
28:            Num ← Num − 1                             ▷ Adjust event counter Num.
29:        else
30:            RETURN List                               ▷ If no improving event, return existing list.
31:        end if
32:    end while
33: end procedure
```

C.4 Top-Down Automatic Event Selection

```
1: procedure TOPDOWNEVENTSEL(File1, File2, Pool,Num)
2:     READ FreqList from File1                          ▷ Extract FreqList from File1.
3:     for F in FreqList do                              ▷ Use PerFreqGen to fit and validate model.
4:         Train ← READ from File1
5:         Test ← READ from File2
6:         Model ← TRAIN(Train,F,Pool)                   ▷ Starting model using all events in Pool.
7:         Erorr ← TEST(Test,F,Model)
8:     end for
9:     TempError ← AVERAGE Error
10:    MinError ← TempError                              ▷ Use starting model metrics as initial reference.
11:    List ← Pool                                       ▷ Initiate event List with all events in Pool.
12:    while TRUE do                                     ▷ Initiate event search. Break condition is inside the loop.
13:        EventRemove ← NULL                            ▷ Improving event identifier EventRemove.
14:        for TempEvent in Pool do                      ▷ Search all the events in Pool.
15:            TempList ← Pool − TempEvent               ▷ Temporary event list TempList using all
    events in Pool except event under test EventRemove.
16:            for F in FreqList do                      ▷ Use PerFreqGen to fit and validate model.
17:                ...
18:            end for
19:            TempError ← AVERAGE Error
20:            if TempError < MinError then              ▷ Compare against stored metrics.
21:                EventRemove ← TempEvent               ▷ Overwrite if tested event improves model.
22:                MinError ← TempError
23:            end if
24:        end for
25:        if EventRemove ≠ NULL then ▷ After searching all events, look for EventRemove
26:            Pool ← Pool − EventRemove                 ▷ Remove from Pool.
27:            SizePool ← SIZE Pool                      ▷ Recalculate Pool size.
28:            if SizePool = Num then                    ▷ Check Pool size.
29:                List ← Pool                           ▷ If Pool reaches Num, update and return List
30:                RETURN List
31:            end if                                    ▷ If Num not reached continue search.
32:        else
33:            List ← Pool                               ▷ If no improving event, return Pool as List.
34:            RETURN List
35:        end if
36:    end while
37: end procedure
```

C.5 Exhaustive Automatic Event Selection

```
1: procedure EXHEVENTSEL(File1, File2, Pool,Num)
2:     READ FreqList from File1                          ▷ Extract FreqList from File1.
3:     EventCombinations ← COMBINATIONS(Pool,Num)  ▷ Compute all combinations
   of Num events from Pool.
4:     for Count in EventCombinations do                 ▷ Search all event combinations.
5:         TempList ← EventCombinations[Count] ▷ Temporary event list TempList using
   events in EventCombinations[Count].
6:         for F in FreqList do                  ▷ Use PerFreqGen to fit and validate model.
7:             ...
8:         end for
9:         TempError ← AVERAGE Error
10:        if MinError = NULL then                ▷ Use first list metrics as initial reference.
11:            MinError ← TempError
12:            List ← TempList
13:        else
14:            if TempError < MinError then              ▷ Compare against stored metrics.
15:                MinError ← TempError       ▷ Overwrite if tested events improve model.
16:                List ← TempList
17:            end if
18:        end if
19:    end for
20:    RETURN List                          ▷ After checking all combinations, return List.
21: end procedure
```

C.6 Inter-Core Model Generation

1: **procedure** INTERCOREGEN($File1$,$FIle2$,$File3$, $File4$,$List$)
2: **READ** $FreqList1$ from $File1$ ▷ Extract $FreqList1$ from $File1$.
3: **READ** $FreqList2$ from $File3$ ▷ Extract $FreqList2$ from $File3$.
4: **for** $F2$ in $FreqList2$ **do** ▷ Outer loop is every target processor frequency.
5: **for** $F1$ in $FreqList1$ **do** ▷ Inner loop is every origin processor frequency. Individual model fitting for every frequency combination.
6: $Train1 \leftarrow$ **READ** from $File1$ ▷ Extract origin train set data from $File1$.
7: $Test1 \leftarrow$ **READ** from $File2$ ▷ Extract origin test set data from $File2$.
8: $Train2 \leftarrow$ **READ** from $File3$ ▷ Extract target train set data from $File3$.
9: $Test2 \leftarrow$ **READ** from $File4$ ▷ Extract target test set data from $File4$.
10: $ESF \leftarrow$ **CALCULATE**($Train1$,$F1$,$Train2$,$F2$) ▷ Compute Event Scaling Factors(ESF) using the origin and target train sets.
11: $Model \leftarrow$ **FIT**($Train2$,$F2$,$List$) ▷ Fit model using event $List$ on target processor $Train2$ data.
12: $F1Error \leftarrow$ **VALIDATE**($Test1$,$F1$,$Test2$,$F2$,$Coeff$,$Model$) ▷ Validate fitted model using origin processor $Test1$ events, scaled with ESF, on target $Test2$ data.
13: **end for**
14: $F2Error \leftarrow$ **AVERAGE** $F1Error$ ▷ Intermediate model performance metric is average $F1Error$ over all origin processor frequencies in $FreqList1$ for target processor frequency $F2$. Many to one relationship.
15: **end for**
16: $ModelError \leftarrow$ **AVERAGE** $F2Error$ ▷ Final model performance metric is average $F2Error$ over all target processor frequencies in $FreqList2$. Many to many relationship.
17: **RETURN** $ModelErrror$
18: **end procedure**

APPENDIX D

POWER MODEL COEFFICIENTS

D.1 ODROID XU+E Cortex-A15 Power Model Coefficients

Model	Constant	Cycles	Instructions	CPI	IPC	CR	CM
Full Cycles	2,91	-4,10E-12					
Full Instructions	2,95		-6,15E-12				
Full CPI	2,89			-1,29E-01			
Full IPC	2,30				4,21E-01		
Full CR	2,97					-1,73E-11	
Full CM	2,94						-2,93E-09
Full IPC, CR, CM	2,79				1,53E-01	-3,86E-12	-2,33E-09
Partial IPC, CR, CM	2,77				1,73E-01	-5,65E-12	-2,11E-09

D.2 ODROID XU+E Cortex-A7 Power Model Coefficients

Model	Constant	Cycles	Instructions	CPI	IPC	CR	CM
Full Cycles	0,10	-6,39E-14					
Full Instructions	0,10		-6,18E-14				
Full CPI	0,10			-3,80E-03			
Full IPC	0,09				9,63E-03		
Full CR	0,10					-1,55E-13	
Full CM	0,10						-3,54E-11
Full Cycles, CR, CM	0,10	-7,50E-14				2,67E-13	-3,49E-11
Full Cycles, Instructions, CR, CM	0,10	-7,81E-14	9,87E-15			2,60E-13	-3,46E-11
Full CPI, CR, CM	0,10			-2,84E-04		4,84E-14	-3,72E-11
Partial Cycles, CR, CM	0,10	-6,58E-14				1,83E-13	-3,01E-11

D.3 Versatile Express with TC2 Cortex-A15 Power Model Coefficients

Model	Constant	Cycles	Instructions	CPI	IPC	CR	CM
Full Cycles	1,55	-2,56E-12					
Full Instructions	1,55		-3,02E-12				
Full CPI	1,53			-7,49E-02			
Full IPC	1,20				2,33E-01		
Full CR	1,73					-2,31E-11	
Full CM	1,59						-1,99E-09
Full Instructions, CR, CM	1,59		8,21E-12			-2,63E-11	-1,36E-09
Full IPC, CR, CM	1,69				1,69E-02	-1,28E-11	-1,46E-09
Partial Instructions, CR, CM	1,59		1,22E-11			-4,10E-11	-8,71E-10

D.4 Versatile Express with TC2 Cortex-A7 Power Model Coefficients

Model	Constant	Cycles	Instructions	CPI	IPC	CR	CM
Full Cycles	0,21	-6,89E-14					
Full Instructions	0,21		-6,54E-14				

Model	Constant	Cycles	Instructions	CPI	IPC	CR	CM
Full CPI	0,21			-2,88E-03			
Full IPC	0,19				9,86E-03		
Full CR	0,21					-4,33E-13	
Full CM	0,21						-4,32E-11
Full IPC, CR, CM	0,21				-4,95E-04	-1,09E-13	-4,01E-11
Full CR, CM	0,21					-1,08E-13	-3,96E-11
Partial IPC, CR, CM	0,21				-6,86E-03	-1,94E-13	-3,79E-11
Partial CM	0,20						-3,78E-11

D.5 ODROID-XU3 Cortex-A15 Single-Thread Power Model Coefficients

F.	Constant	r011	r014	r004	r01D	r065	r079
2	1.45E+00	2.87E-07	1.23E-09	6.50E-10	-1.44E-06	-8.28E-06	8.94E-10
1.9	1.31E+00	-1.22E-07	1.06E-09	5.49E-10	6.08E-07	-9.27E-06	3.26E-09
1.8	1.13E+00	-3.10E-07	9.54E-10	5.01E-10	1.55E-06	-8.23E-06	1.93E-10
1.7	9.50E-01	-6.75E-08	8.62E-10	4.93E-10	2.70E-07	-5.48E-06	1.81E-09
1.6	8.40E-01	1.77E-08	8.09E-10	4.37E-10	-7.07E-08	-4.97E-06	1.30E-09
1.5	6.97E-01	-2.10E-07	7.62E-10	3.88E-10	8.41E-07	-2.71E-06	1.69E-09
1.4	6.59E-01	-1.12E-07	7.06E-10	3.89E-10	4.48E-07	-2.93E-06	8.86E-10
1.3	5.74E-01	-1.43E-07	6.93E-10	3.80E-10	4.28E-07	-2.13E-06	1.21E-09
1.2	4.27E-01	-1.54E-08	6.46E-10	3.70E-10	4.66E-08	1.04E-07	1.89E-09
1.1	3.46E-01	-2.10E-07	6.44E-10	3.03E-10	6.31E-07	9.50E-07	1.78E-09

F.	Constant	r011	r014	r004	r01D	r065	r079
1	2.71E-01	-5.93E-08	5.85E-10	3.30E-10	1.78E-07	1.71E-06	1.99E-09
0.9	2.78E-01	-5.38E-08	5.49E-10	3.17E-10	1.62E-07	6.12E-07	1.87E-09
0.8	2.31E-01	-3.13E-08	5.16E-10	2.99E-10	9.41E-08	8.82E-07	2.08E-09
0.7	2.19E-01	-3.18E-08	5.05E-10	2.95E-10	9.57E-08	7.05E-07	1.77E-09
0.6	1.90E-01	-2.96E-08	5.11E-10	2.98E-10	8.89E-08	8.87E-07	1.61E-09
0.5	1.68E-01	-1.94E-08	5.13E-10	2.95E-10	5.82E-08	1.15E-06	1.60E-09
0.4	1.52E-01	-2.40E-09	5.04E-10	2.96E-10	7.05E-09	1.78E-06	1.53E-09
0.3	9.94E-02	-1.09E-08	4.96E-10	2.94E-10	3.27E-08	5.06E-06	1.14E-09
0.2	8.07E-02	3.34E-09	4.83E-10	2.98E-10	-1.04E-08	5.48E-06	7.77E-10

D.6 ODROID-XU3 Cortex-A7 Single-Thread Power Model Coefficients

F.	Constant	r011	r060	r017	r00F	r01D
1.4	2.30E-01	-5.27E-08	-1.56E-08	3.72E-08	5.51E-09	2.11E-07
1.3	1.87E-01	1.00E-08	-1.42E-08	3.45E-08	8.69E-09	-2.98E-08
1.2	1.76E-01	2.25E-08	-1.28E-08	3.52E-08	6.17E-09	-6.74E-08
1.1	1.47E-01	1.25E-08	-1.13E-08	3.23E-08	6.46E-09	-3.73E-08
1	1.41E-01	-1.41E-08	-1.01E-08	2.78E-08	6.70E-09	4.23E-08
0.9	1.17E-01	-2.47E-08	-8.48E-09	2.36E-08	7.17E-09	7.40E-08
0.8	8.97E-02	2.37E-10	-7.38E-09	2.29E-08	6.47E-09	-6.58E-10
0.7	8.13E-02	-1.20E-09	-6.45E-09	2.13E-08	6.80E-09	3.58E-09
0.6	6.91E-02	-2.47E-09	-5.49E-09	1.91E-08	5.92E-09	7.34E-09
0.5	5.76E-02	5.77E-10	-4.74E-09	1.77E-08	6.18E-09	-1.85E-09
0.4	4.95E-02	5.51E-10	-4.11E-09	1.64E-08	5.70E-09	-1.80E-09
0.3	4.58E-02	1.82E-10	-4.35E-09	1.82E-08	5.22E-09	-8.43E-10
0.2	2.76E-02	4.37E-10	-3.56E-09	1.49E-08	3.63E-09	-1.50E-09

D.7 ODROID-XU3 Cortex-A15 Multithread Power Model Coefficients

F.	Constant	Cores(#)	r011	r040	r010	r07E	r058	r078	r01D
1.8	5.98E-1	2.08E-1	9.38E-7	2.78E-10	-3.35E-8	-1.91E-7	1.76E-6	3.51E-9	-4.69E-6
1.7	5.02E-1	1.02E-1	7.71E-7	6.93E-10	-3.09E-8	-9.53E-8	-8.86E-6	2.47E-9	-3.08E-6
1.6	4.51E-1	5.72E-2	7.79E-7	2.40E-10	-2.67E-8	-9.63E-8	-9.22E-6	2.78E-9	-3.11E-6

F.	Constant	Cores(#)	r011	r040	r010	r07E	r058	r078	r01D
1.5	4.11E-1	4.81E-2	3.88E-7	3.78E-10	-2.43E-8	-8.93E-8	-1.22E-5	2.25E-9	-1.55E-6
1.4	3.53E-1	7.79E-2	5.70E-7	2.44E-10	-2.61E-8	-1.14E-7	-6.77E-6	2.41E-9	-2.28E-6
1.3	3.15E-1	8.55E-2	8.06E-7	9.80E-11	-2.43E-8	-1.08E-7	-3.81E-6	2.40E-9	-2.42E-6
1.2	2.80E-1	6.98E-2	6.21E-7	1.85E-11	-2.42E-8	-1.04E-7	-1.64E-6	2.38E-9	-1.86E-6
1.1	2.47E-1	4.54E-2	4.76E-7	2.37E-10	-2.08E-8	-7.57E-8	-2.27E-6	1.83E-9	-1.42E-6
1	2.25E-1	5.67E-2	3.92E-7	-2.57E-10	-2.31E-8	-1.03E-7	-6.46E-6	2.49E-9	-1.17E-6
0.9	1.89E-1	2.45E-2	3.96E-7	-1.40E-10	-2.06E-8	-7.52E-8	-1.75E-6	2.12E-9	-1.19E-6
0.8	1.70E-1	2.93E-2	4.11E-7	-6.24E-11	-1.82E-8	-7.84E-8	-2.57E-6	1.76E-9	-1.23E-6
0.7	1.43E-1	2.58E-2	2.70E-7	1.48E-10	-1.76E-8	-6.45E-8	-2.27E-6	1.40E-9	-8.09E-7
0.6	1.25E-1	1.59E-2	1.77E-7	1.27E-10	-1.75E-8	-5.97E-8	-2.07E-6	1.40E-9	-5.30E-7
0.5	1.06E-1	1.16E-2	1.90E-7	2.38E-10	-1.67E-8	-5.41E-8	-1.27E-6	1.25E-9	-5.70E-7
0.4	8.18E-2	1.12E-2	1.07E-7	3.13E-10	-1.60E-8	-4.82E-8	-1.41E-6	1.07E-9	-3.19E-7
0.3	6.24E-2	3.82E-3	6.08E-8	4.20E-10	-1.59E-8	-3.65E-8	-4.52E-7	9.73E-10	-1.81E-7
0.2	4.38E-2	2.35E-3	1.44E-8	4.86E-10	-1.54E-8	-2.85E-8	-3.91E-7	9.02E-10	-4.15E-8

D.8 ODROID-XU3 Cortex-A7 Multithread Power Model Coefficients

F.	Constant	Cores(#)	r011	r014	r015	r006	r00D
1.4	9.54E-02	1.46E-02	9.30E-11	3.34E-10	2.56E-09	2.28E-10	-2.55E-10
1.3	8.04E-02	1.14E-02	8.38E-11	3.15E-10	2.80E-09	1.98E-10	-2.47E-10
1.2	6.79E-02	1.21E-02	7.72E-11	2.61E-10	2.07E-09	2.01E-10	-2.18E-10
1.1	5.89E-02	1.08E-02	7.23E-11	2.71E-10	2.21E-09	1.00E-10	-2.23E-10
1	5.03E-02	8.83E-03	6.98E-11	2.40E-10	1.97E-09	1.33E-10	-2.50E-10
0.9	4.08E-02	6.83E-03	6.79E-11	2.03E-10	1.82E-09	1.51E-10	-1.74E-10
0.8	3.07E-02	4.12E-03	6.92E-11	1.97E-10	1.68E-09	9.39E-11	-1.94E-10
0.7	2.38E-02	3.59E-03	6.52E-11	1.72E-10	1.66E-09	1.17E-10	-1.53E-10
0.6	1.72E-02	3.52E-03	6.08E-11	1.87E-10	1.85E-09	5.89E-11	-1.95E-10
0.5	1.31E-02	2.49E-03	6.10E-11	1.49E-10	1.55E-09	8.33E-11	-1.56E-10
0.4	1.08E-02	1.97E-03	6.59E-11	1.49E-10	1.60E-09	6.68E-11	-1.67E-10
0.3	8.60E-03	1.49E-03	7.00E-11	1.46E-10	1.62E-09	6.41E-11	-1.83E-10
0.2	6.10E-03	9.64E-04	7.90E-11	1.36E-10	1.60E-09	5.72E-11	-1.97E-10

D.9 ODROID-XU3 Cortex-A15 Single-Thread Inter-Core Power Model Coefficients

F.	Constant	r011	r00A	r010	r018	r019
2	1.72E+00	5.02E-11	5.66E-06	4.47E-08	4.36E-06	-1.91E-07
1.9	1.38E+00	6.99E-11	5.21E-06	3.83E-08	3.23E-06	-1.43E-07
1.8	1.28E+00	-5.09E-11	4.21E-06	3.42E-08	3.26E-06	-1.42E-07
1.7	1.08E+00	1.28E-11	4.14E-06	3.19E-08	2.76E-06	-1.21E-07
1.6	1.01E+00	-6.45E-11	3.49E-06	2.90E-08	2.72E-06	-1.19E-07
1.5	9.69E-01	-1.87E-10	3.25E-06	2.65E-08	2.64E-06	-1.15E-07
1.4	9.20E-01	-2.61E-10	2.96E-06	2.54E-08	2.53E-06	-1.09E-07
1.3	8.57E-01	-2.80E-10	2.98E-06	2.36E-08	2.33E-06	-9.94E-08
1.2	6.80E-01	-1.41E-10	2.72E-06	2.17E-08	1.86E-06	-7.85E-08
1.1	5.24E-01	8.31E-12	2.80E-06	2.01E-08	1.24E-06	-5.19E-08
1	4.33E-01	5.35E-11	2.93E-06	1.78E-08	8.47E-07	-3.37E-08
0.9	3.71E-01	5.23E-11	2.72E-06	1.67E-08	6.82E-07	-2.66E-08
0.8	3.19E-01	3.53E-11	2.48E-06	1.57E-08	6.22E-07	-2.33E-08
0.7	2.95E-01	-2.57E-11	2.55E-06	1.51E-08	5.90E-07	-2.26E-08
0.6	2.34E-01	5.20E-11	2.65E-06	1.52E-08	4.63E-07	-1.71E-08
0.5	1.90E-01	8.85E-11	2.72E-06	1.48E-08	2.13E-07	-5.98E-09
0.4	1.38E-01	1.79E-10	2.57E-06	1.43E-08	6.63E-08	1.17E-09
0.3	9.34E-02	2.82E-10	2.50E-06	1.37E-08	-8.23E-08	8.33E-09
0.2	7.42E-02	2.17E-10	2.32E-06	1.37E-08	7.21E-10	5.57E-09

D.10 ODROID-XU3 Cortex-A7 Single-Thread Inter-Core Power Model Coefficients

F.	Constant	r011	r014	r012
1.4	2.26E-01	-4.32E-11	3.79E-10	-3.20E-10
1.3	1.91E-01	-3.71E-11	3.66E-10	-3.43E-10
1.2	1.70E-01	-4.79E-11	3.20E-10	-3.16E-10
1.1	1.47E-01	-4.80E-11	2.99E-10	-3.02E-10
1	1.34E-01	-6.17E-11	2.79E-10	-3.21E-10
0.9	1.13E-01	-5.71E-11	2.54E-10	-3.06E-10
0.8	8.69E-02	-5.63E-11	2.29E-10	-2.85E-10
0.7	7.95E-02	-7.35E-11	2.10E-10	-2.85E-10
0.6	6.59E-02	-7.49E-11	1.85E-10	-2.54E-10

F.	Constant	r011	r014	r012
0.5	6.33E-02	-1.16E-10	1.70E-10	-2.50E-10
0.4	5.43E-02	-1.23E-10	1.59E-10	-2.55E-10
0.3	5.38E-02	-2.07E-10	1.75E-10	-2.91E-10
0.2	4.38E-02	-2.71E-10	1.42E-10	-2.02E-10

D.11 ODROID-XU3 Cortex-A15 Multithread Inter-Core Power Model Coefficients

F.	Constant	Cores(#)	r011	r009	r018	r010	r00A
1.8	6.22E-01	1.24E-01	1.26E-09	-8.41E-05	-3.58E-07	1.46E-09	4.07E-05
1.7	5.50E-01	9.43E-02	1.17E-09	-7.16E-05	-2.86E-07	2.02E-09	3.41E-05
1.6	4.65E-01	8.24E-02	1.09E-09	-7.12E-05	-2.37E-07	3.16E-09	3.45E-05
1.5	4.07E-01	5.28E-02	1.03E-09	-7.18E-05	-2.08E-07	7.13E-10	3.54E-05
1.4	3.54E-01	9.22E-02	9.24E-10	-5.13E-05	-1.87E-07	-9.73E-10	2.34E-05
1.3	3.20E-01	5.92E-02	9.44E-10	-5.52E-05	-1.63E-07	-1.54E-09	2.62E-05
1.2	2.96E-01	5.62E-02	8.74E-10	-3.23E-05	-1.42E-07	-3.78E-09	1.40E-05
1.1	2.52E-01	3.90E-02	8.53E-10	-4.78E-05	-1.32E-07	-2.61E-09	2.29E-05
1	2.30E-01	2.24E-02	8.12E-10	-5.05E-05	-1.09E-07	-6.49E-10	2.45E-05
0.9	2.03E-01	2.77E-02	7.50E-10	-4.17E-05	-9.46E-08	-2.84E-09	1.97E-05
0.8	1.74E-01	1.94E-02	7.13E-10	-2.37E-05	-7.46E-08	-3.52E-09	1.04E-05
0.7	1.48E-01	1.08E-02	7.17E-10	-3.14E-05	-6.38E-08	-3.05E-09	1.48E-05
0.6	1.27E-01	5.94E-03	7.35E-10	-2.69E-05	-5.23E-08	-2.92E-09	1.24E-05
0.5	1.07E-01	5.33E-03	7.44E-10	-3.31E-05	-4.65E-08	-3.78E-09	1.58E-05
0.4	8.31E-02	2.49E-03	7.75E-10	-3.04E-05	-3.44E-08	-4.21E-09	1.44E-05
0.3	6.29E-02	1.15E-03	7.92E-10	-2.41E-05	-2.43E-08	-4.35E-09	1.11E-05
0.2	4.39E-02	3.10E-04	8.12E-10	-1.96E-05	-1.64E-08	-4.64E-09	8.71E-06

D.12 ODROID-XU3 Cortex-A7 Multithread Inter-Core Power Model Coefficients

F.	Constant	Cores(#)	r011	r009	r00A
1.4	1.17E-01	2.12E-02	1.70E-10	4.10E-06	-3.16E-06
1.3	9.59E-02	1.77E-02	1.61E-10	2.18E-06	-1.96E-06
1.2	7.98E-02	1.73E-02	1.46E-10	-1.32E-06	1.83E-08
1.1	6.73E-02	1.43E-02	1.39E-10	7.18E-07	-9.82E-07

F.	Constant	Cores(#)	r011	r009	r00A
1	5.77E-02	1.22E-02	1.32E-10	4.44E-07	-8.17E-07
0.9	4.74E-02	1.05E-02	1.26E-10	3.94E-07	-7.39E-07
0.8	3.51E-02	7.22E-03	1.22E-10	-1.13E-06	1.62E-07
0.7	2.71E-02	5.32E-03	1.20E-10	-6.75E-07	-1.69E-08
0.6	1.97E-02	4.29E-03	1.17E-10	-5.34E-07	-5.17E-08
0.5	1.48E-02	3.42E-03	1.11E-10	-8.65E-07	1.57E-07
0.4	1.22E-02	2.61E-03	1.15E-10	-1.05E-06	2.92E-07
0.3	9.41E-03	1.72E-03	1.21E-10	-6.99E-07	1.23E-07
0.2	6.48E-03	9.49E-04	1.29E-10	-1.58E-06	6.49E-07

APPENDIX E

MODIFIED PARSEC BLACKSCHOLES FOR HETEROGENEOUS EXECUTION ON 8 CORES

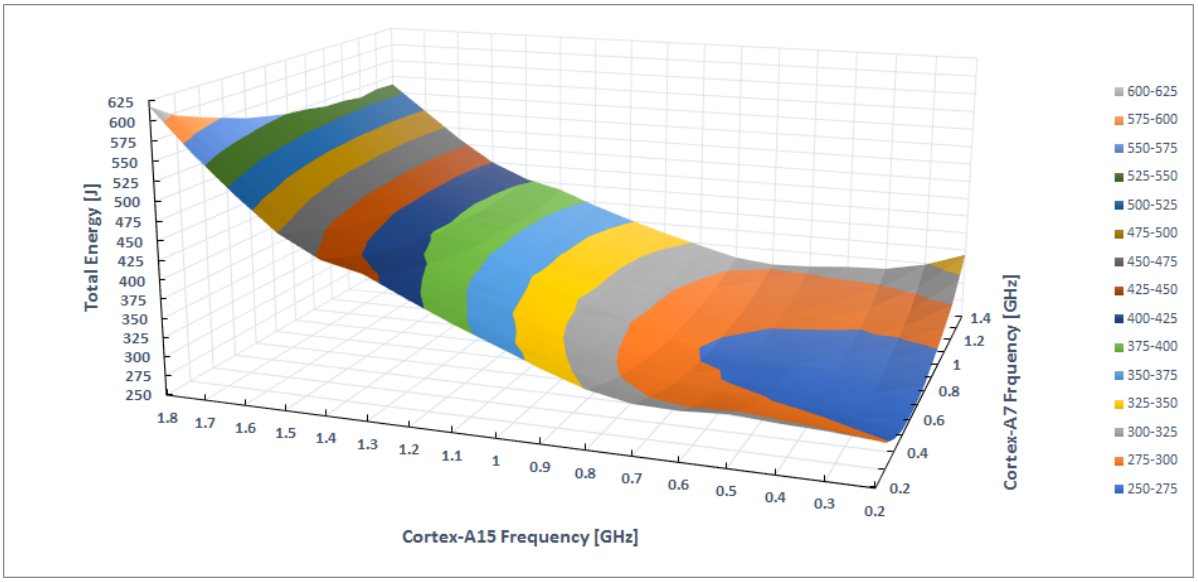


Figure E.1: PARSEC Blackscholes E/F Curve

BIBLIOGRAPHY

[1] N. Heuveldop *et al.*, "Ericsson mobility report," *Ericsson AB, Technol. Emerg. Business, Stockholm, Sweden, Tech. Rep. EAB-17*, vol. 5964, 2017.

[2] S. Borkar and A. A. Chien, "The future of microprocessors," *Communications of the ACM*, vol. 54, no. 5, pp. 0–5, 2011. [Online]. Available: http://search.ebscohost.com/login.aspx?direct=true{&}db=bth{&}AN=60863975{&}site=ehost-live

[3] J. E. Stone, D. Gohara, and G. Shi, "Opencl: A parallel programming standard for heterogeneous computing systems," *Computing in science & engineering*, vol. 12, no. 3, pp. 66–73, 2010.

[4] C. Nvidia, "Programming guide," 2010.

[5] A. Munshi, B. Gaster, T. G. Mattson, and D. Ginsburg, *OpenCL programming guide*. Pearson Education, 2011.

[6] http://www.arm.com/products/processors/technologies/biglittleprocessing.php, [Online; accessed 10-Oct-2013].

[7] https://www.arm.com/about/newsroom/arm-unveils-its-most-energy-efficient-application-proces
php, [Online; accessed 21-Oct-2014].

[8] https://www.arm.com/files/pdf/big_LITTLE_Technology_the_Futue_of_Mobile.pdf, [Online; accessed 2-Feb-2014].

[9] R. Kumar, D. Tullsen, P. Ranganathan, N. Jouppi, and K. Farkas, "Single-ISA heterogeneous multi-core architectures for multithreaded workload performance," *Proceedings. 31st Annual International Symposium on Computer Architecture, 2004.*, 2004.

[10] M. D. Hill and M. R. Marty, "Amdahl's law in the multicore era," *Computer 7*, no. April, pp. 33–38, 2008.

[11] J. L. Gustafson, "Reevaluating amdahl's law," *Communications of the ACM*, vol. 31, no. 5, pp. 532–533, 1988.

[12] S. Borkar, "Thousand core chips: a technology perspective," in *Proceedings of the 44th annual Design Automation Conference*. ACM, 2007, pp. 746–749.

[13] D. H. Woo and H.-H. S. Lee., "Extending Amdahl's law for energy-efficient computing in the many-core era," *Computer 12*, pp. 24–31, 2008.

[14] http://www.hardkernel.com/main/products/prdt_info.php?g_code=G137463363079, [Online; accessed 10-Oct-2013].

[15] https://www.arm.com/files/pdf/Motherboard_Express_uATX.pdf, [Online; accessed 20-Oct-2014].

[16] https://static.docs.arm.com/dui0447/j/DUI0447.pdf, [Online; accessed 15-Feb-2014].

[17] https://static.docs.arm.com/ddi0503/i/DDI0503I_v2p_ca15_a7_tc2_trm.pdf, [Online; accessed 20-Feb-2014].

[18] http://www.hardkernel.com/main/products/prdt_info.php?g_code=G140448267127, [Online; accessed 12-March-2015].

[19] http://ctuning.org/wiki/index.php?title=CTools:CBench/, [Online; accessed 19-Oct-2014].

[20] http://parsec.cs.princeton.edu/index.htm, [Online; accessed 02-May-2017].

[21] K. Nikov, J. L. Nunez-Yanez, and M. Horsnell, "Evaluation of hybrid run-time power models for the ARM big. Little architecture," *Proceedings - IEEE/IFIP 13th International Conference on Embedded and Ubiquitous Computing, EUC 2015*, pp. 205–210, 2015.

[22] https://www.researchgate.net/publication/319914261_The_energy_consumption_benefits_of_DynamIQ_for_heterogeneous_parallel_workloads, [Online; accessed 19-Sep-2017].

[23] L. Benini, A. Bogliolo, and G. De Micheli, "A survey of design techniques for system-level dynamic power management," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 8, no. 3, pp. 299–316, 2000.

[24] http://www.acpi.info/, [Online; accessed 19-Sep-2014].

[25] http://www.uefi.org/sites/default/files/resources/ACPI%206_2_A_Sept29.pdf, [Online; accessed 10-Feb-2018].

[26] M. Pedram, "Power optimization and management in embedded systems," in *Proceedings of the 2001 Asia and South Pacific Design Automation Conference*. ACM, 2001, pp. 239–244.

[27] http://chipdesignmag.com/sld/blog/2014/03/12/system-level-power-budgeting/, [Online; accessed 10-Aug-2014].

[28] https://www.apache-da.com/company/events/ATF2011, [Online; accessed 10-Aug-2014].

[29] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," *IEEE Micro*, vol. 32, pp. 122–134, 2012.

[30] M. Shafique, S. Garg, T. Mitra, S. Parameswaran, and J. Henkel, "Dark silicon as a challenge for hardware/software co-design," *Proceedings of the 2014 International Conference on Hardware/Software Codesign and System Synthesis - CODES '14*, pp. 1–10, 2014. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2656075.2661645

[31] K. De Vogeleer, G. Memmi, P. Jouvelot, and F. Coelho., "The energy/frequency convexity rule: Modeling and experimental validation on mobile devices." *Parallel Processing and ...*, pp. 793–803, 2014. [Online]. Available: isi:000180067200055{%}5Cnhttp://link.springer.com/chapter/10.1007/3-540-48086-2{_}55{%}5Cnhttp://link.springer.com/10.1007/3-540-48086-2

[32] K. Rangan, G. Wei, and D. Brooks, "Thread motion: fine-grained power management for multi-core systems," *ACM SIGARCH Computer Architecture ...*, pp. 302–313, 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1555793

[33] K. Choi, R. Soma, and M. Pedram, "Fine-grained dynamic voltage and frequency scaling for precise energy and performance tradeoff based on the ratio of off-chip access to on-chip computation times," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 1, pp. 18–28, 2005.

[34] D. Shelepov and J. S. Alcaide, "HASS: a scheduler for heterogeneous multicore systems," *... Operating Systems ...*, pp. 66–75, 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1531804

[35] J. Henning, "SPEC CPU2000: measuring CPU performance in the New Millennium," *Computer (Long. Beach. Calif).*, vol. 33, no. 7, pp. 28–35, 2000. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=869367

[36] M. Curtis-maury, A. Shah, F. Blagojevic, D. S. Nikolopoulos, B. R. D. Supinski, and M. Schulz, "Prediction Models for Multi-dimensional Power-Performance Optimization on Many Cores Categories and Subject Descriptors," pp. 250–259.

[37] L. Dagum and R. Menon, "Openmp: an industry standard api for shared-memory programming," *IEEE computational science and engineering*, vol. 5, no. 1, pp. 46–55, 1998.

[38] D. H. Bailey, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, S. K. Weeratunga, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. a. Fatoohi, P. O. Frederickson, and T. a. Lasinski, "The NAS parallel benchmarks—summary and preliminary results," *Proceedings of the 1991 ACM/IEEE conference on Supercomputing - Supercomputing '91*, pp. 158–165, 1991. [Online]. Available: http://ieeexplore.ieee.org/xpl/freeabs{_}all.jsp?arnumber=5348941

[39] C. Imes and H. Hoffmann, "Minimizing energy under performance constraints on embedded platforms: resource allocation heuristics for homogeneous and single-ISA heterogeneous multi-cores," *ACM SIGBED Review*, vol. 11, no. 4, pp. 49–54, 2015.

[40] C. Bienia and K. Li, "Parsec 2.0: A new benchmark suite for chip-multiprocessors," *5th Annual Workshop on Modeling, Benchmarking and Simulation*, pp. 1–9, 2009. [Online]. Available: http://www-mount.ece.umn.edu/{~}jjyi/MoBS/2009/program/02E-Bienia.pdf

[41] K. V. Craeynest, "Scheduling Heterogeneous Multi-Cores through Performance Impact Estimation (PIE) type I type II type III," vol. 00, no. c, pp. 213–224, 2012.

[42] J. L. Henning, "SPEC CPU2006 benchmark descriptions." *ACM SIGARCH Computer Architecture News*, vol. 34, no. 4, pp. 1–17, 2006. [Online]. Available: http://scholar.google.com/scholar?hl=en{&}btnG=Search{&}q=intitle:No+Title{#}0

[43] W.-T. Hsieh, C.-C. Shiue, and C.-N. Liu, "Efficient power modelling approach of sequential circuits using recurrent neural networks," *IEE Proceedings - Computers and Digital Techniques*, vol. 153, no. 2, p. 78, 2006. [Online]. Available: http://digital-library.theiet.org/content/journals/10.1049/ip-cdt{_}20045147

[44] I. Takouna, W. Dawoud, and C. Meinel, "Accurate Mutlicore Processor Power Models for Power-Aware Resource Management," *2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing*, pp. 419–426, dec 2011. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6118753

[45] T. Sherwood, E. Perelman, G. Hamerly, S. Sair, and B. Calder, "Discovering and Exploiting Program Phases," *IEEE Micro*, vol. 23, pp. 84–93, 2003.

[46] C. Isci and M. Martonpsi, "Phase characterization for power: Evaluating control-flow-based and event-counter-based techniques," *Proceedings - International Symposium on High-Performance Computer Architecture*, vol. 2006, pp. 122–133, 2006.

[47] R. Bertran, M. Gonzalez, X. Martorell, N. Navarro, and E. Ayguade, "Decomposable and Responsive Power Models for Multicore Processors using Performance Counters Cat-

egories and Subject Descriptors," *Proceedings of the 24th ACM International Conference on Supercomputing*, pp. 147–158, 2010.

[48] M. J. Pazzani and S. D. Bay, "The Independent Sign Bias : Gaining Insight from Multiple Linear Regression," 1981.

[49] J. Nunez-Yanez and G. Lore, "Enabling accurate modeling of power and energy consumption in an ARM-based System-on-Chip," *Microprocessors and Microsystems*, vol. 37, no. 3, pp. 319–332, may 2013. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0141933113000021

[50] M. Pricopi, T. S. Muthukaruppan, V. Venkataramani, T. Mitra, and S. Vishin, "Power-performance modeling on asymmetric multi-cores," *2013 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*, pp. 1–10, sep 2013. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6662519

[51] K. Singh, M. Bhadauria, and S. a. McKee, "Real time power estimation and thread scheduling via performance counters," *ACM SIGARCH Computer Architecture News*, vol. 37, no. 2, p. 46, 2009.

[52] R. Rodrigues, A. Annamalai, I. Koren, and S. Kundu, "A study on the use of performance counters to estimate power in microprocessors," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 60, no. 12, pp. 882–886, 2013.

[53] M. Guthaus and J. Ringenberg, "MiBench: A free, commercially representative embedded benchmark suite," *... , 2001. WWC-4. ...*, pp. 3–14, 2001. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs{_}all.jsp?arnumber=990739

[54] C. Lee, M. Potkonjak, and W. H. Mangione-Smith, "Mediabench: a tool for evaluating and synthesizing multimedia and communicatons systems," in *Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture*. IEEE Computer Society, 1997, pp. 330–335.

[55] 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," *Journal of Systems Architecture*, vol. 53, no. 10, pp. 689–702, oct 2007. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S1383762107000161

[56] S. K. Rethinagiri, O. Palomar, R. Ben Atitallah, S. Niar, O. Unsal, and A. C. Kestelman, "System-level power estimation tool for embedded processor based platforms," *Proceedings of the 6th Workshop on Rapid Simulation and Performance*

Evaluation Methods and Tools - RAPIDO '14, pp. 1–8, 2014. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2555486.2555491

[57] C. Dismuke and R. Lindrooth, "Ordinary least squares," *Methods and Designs for Outcomes Research*, vol. 93, pp. 93–104, 2006.

[58] S. Eyerman, K. Hoste, and L. Eeckhout, "Mechanistic-empirical processor performance modeling for constructing CPI stacks on real hardware," *(Ieee Ispass) Ieee International Symposium on Performance Analysis of Systems and Software*, pp. 216–226, apr 2011. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5762738

[59] H. Jacobson and A. Buyuktosunoglu, "Abstraction and microarchitecture scaling in early-stage power modeling," *... (HPCA), 2011 IEEE ...*, pp. 394–405, 2011. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs{_}all.jsp?arnumber=5749746

[60] Y. Zhang, X. Wang, X. Liu, Y. Liu, Ł. Zhuang, and F. Zhao, "Towards better CPU power management on multicore smartphones," *Proceedings of the Workshop on Power-Aware Computing and Systems - HotPower '13*, pp. 1–5, 2013. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2525526.2525849

[61] M. J. Walker, S. Diestelhorst, A. Hansson, A. K. Das, S. Yang, B. M. Al-hashimi, and G. V. Merrett, "Accurate and Stable Run-Time Power Modeling for Mobile and Embedded CPUs," *Ieee Transactions on Computer Aided Design of Integrated Circuits and Systems*, pp. 1–14, 2015.

[62] ——, "Accurate and Stable Run-Time Power Modeling for Mobile and Embedded CPUs," *Ieee Transactions on Computer Aided Design of Integrated Circuits and Systems*, vol. 36, no. 1, pp. 1–14, 2017.

[63] M. Kim, H. Kim, H. Chung, and K. Lim, "Samsung exynos 5410 processor-experience the ultimate performance and versatility," *White Paper*, 2013.

[64] P. Greenhalgh, "Big. little processing with arm cortex-a15 & cortex-a7," *ARM White paper*, vol. 17, 2011.

[65] B. Jeff, "Advances in big. little technology for power and energy savings," *ARM White paper*, p. 33, 2012.

[66] https://developer.arm.com/products/system-ip/corelink-interconnect/corelink-cache-coherent-interconnect-family", note =.

[67] https://developer.arm.com/docs/ddi0470/latest/preface, [Online; accessed 13-Jun-2016].

[68] https://developer.arm.com/products/architecture/amba-protocol/amba-4", note =.

[69] H.-D. Cho, P. D. P. Engineer, K. Chung, and T. Kim, "Benefits of the big. little architecture," *EETimes, Feb*, 2012.

[70] H. Chung, M. Kang, and H.-D. Cho, "Heterogeneous multi-processing solution of exynos 5 octa with arm® big. little™ technology," *Samsung White Paper*, 2012.

[71] K. Krewell, "Cortex-a53 is arm's next little thing," *Microprocessor Report*, vol. 11, no. 5, pp. 12–2, 2012.

[72] J. Bolaria, "Cortex-a57 extends arm's reach," *Microprocessor Report*, vol. 11, no. 5, pp. 12–1, 2012.

[73] https://www.arm.com/files/pdf/ARM_Qualcomm_White_paper_Final.pdf, [Online; accessed 1-Jul-2015].

[74] http://www.arm.com/products/processors/cortex-a/cortex-a15.php, [Online; accessed 10-Oct-2013].

[75] http://infocenter.arm.com/help/topic/com.arm.doc.ddi0438c/DDI0438C_cortex_a15_r2p0_trm.pdf, [Online; accessed 10-Dec-2013].

[76] T. Lanier, "Exploring the design of the cortex-a15 processor," *URL: http://www. arm. com/files/pdf/atexploring the design of the cortex-a15. pdf (visited on 12/11/2013)*, 2011.

[77] http://www.arm.com/products/processors/cortex-a/cortex-a7.php, [Online; accessed 10-Oct-2013].

[78] http://infocenter.arm.com/help/topic/com.arm.doc.ddi0464d/DDI0464D_cortex_a7_mpcore_r0p3_trm.pdf, [Online; accessed 10-Dec-2013].

[79] https://perf.wiki.kernel.org/index.php/Main_Page", note =.

[80] R. Randhawa, "Software Techniques for ARM big.LITTLE Systems," p. 9, 2013. [Online]. Available: http://www.arm.com/files/downloads/Software{_}Techniques{_}for{_}ARM{_}big.LITTLE{_}Systems.pdf

[81] M. Poirier, "In kernel switcher: A solution to support arm's new big. little technology," in *Embedded Linux Conference*, 2013.

[82] B. Jeff, "big. little technology moves towards fully heterogeneous global task scheduling," *White Paper.*, 2013.

[83] https://www.arm.com/files/pdf/Heterogeneous_Multi_Processing_Solution_of_Exynos_5_Octa_with_ARM_bigLITTLE_Technology.pdf", note =.

[84] http://www.ti.com/product/ina231, [Online; accessed 14-Sep-2014].

[85] P. Greenhalgh, "big . LITTLE Processing with," no. September 2011, pp. 1–8, 2012.

[86] http://web.eece.maine.edu/~vweaver/projects/perf_events/, [Online; accessed 13-Nov-2013].

[87] https://community.arm.com/processors/b/blog/posts/cortex-a15-and-cortex-a7-big-little-hardware-from-arm, [Online; accessed 09-Feb-2018].

[88] https://snapshots.linaro.org/ubuntu/pre-built/lsk-vexpress/, [Online; accessed 1-Mar-2014].

[89] Y. Shin, K. Shin, P. Kenkare, R. Kashyap, H. J. Lee, D. Seo, B. Millar, Y. Kwon, R. Iyengar, M. S. Kim, A. Chowdhury, S. I. Bae, I. Hong, W. Jeong, A. Lindner, U. Cho, K. Hawkins, J. C. Son, and S. H. Hwang, "28nm high-κ metal-gate heterogeneous quad-core CPUs for high-performance and energy-efficient mobile application processor," *Digest of Technical Papers - IEEE International Solid-State Circuits Conference*, vol. 56, pp. 154–155, 2013.

[90] http://cloc.sourceforge.net/, [Online; accessed 10-Feb-2018].

[91] http://manpages.ubuntu.com/manpages/precise/man1/cset.1.html, [Online; accessed 19-Jul-2015].

[92] https://linux.die.net/man/1/cpufreq-info, [Online; accessed 10-Feb-2018].

[93] https://linux.die.net/man/1/cpufreq-set, [Online; accessed 10-Feb-2018].

[94] C. Bienia, "BENCHMARKING MODERN MULTIPROCESSORS," 2011. [Online]. Available: http://parsec.cs.princeton.edu/publications/bienia11benchmarking.pdf

[95] https://github.com/kranik/DATACOLLECT/tree/master/ARMPM_datacollect/ODROID_XU3", note =.

[96] http://www.netlib.org/benchmark/dhry-c, [Online; accessed 20-Oct-2013].

[97] http://www.netlib.org/benchmark/whetstonec, [Online; accessed 20-Oct-2013].

[98] http://www.netlib.org/linpack/, [Online; accessed 20-Oct-2013].

[99] J. Clemons, H. Zhu, S. Savarese, T. Austin, and A. Arbor, "MEVBench : A Mobile Computer Vision Benchmarking Suite," pp. 91–102, 2011.

[100] J. Stratton and C. Rodrigues, "Parboil: A revised benchmark suite for scientific and commercial throughput computing," ... *Computing*, 2012. [Online]. Available: http://impact.crhc.illinois.edu/shared/docs/impact-12-01.parboil.pdf

[101] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," *2009 IEEE International Symposium on Workload Characterization (IISWC)*, pp. 44–54, oct 2009. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5306797

[102] http://www.phoronix-test-suite.com/, [Online; accessed 20-Oct-2013].

[103] https://github.com/kranik/BUILDMODEL/tree/master/ARMPM_buildmodel", note =.

[104] D. A. Belsley, E. Kuh, and R. E. Welsch, *Regression diagnostics: Identifying influential data and sources of collinearity*. John Wiley & Sons, 2005, vol. 571.

[105] M. H. Kutner, C. Nachtsheim, and J. Neter, *Applied linear regression models*. McGraw-Hill/Irwin, 2004.

[106] W. C. Black, B. J. Babin, R. E. Anderson *et al.*, *Multivariate data analysis*, vol. 5, no. 3.

[107] H. Kim, N. Agrawal, and C. Ungureanu, "Revisiting storage for smartphones," *ACM Transactions on Storage (TOS)*, vol. 8, no. 4, p. 14, 2012.

[108] https://wiki.linaro.org/Flash%20memory, [Online; accessed 16-Mar-2017].

[109] https://wiki.linaro.org/WorkingGroups/KernelArchived/Projects/FlashCardSurvey, [Online; accessed 17-Sep-2015].

[110] https://wiki.linaro.org/WorkingGroups/KernelArchived/Specs/StoragePerfEMMC?highlight=, [Online; accessed 14-Jul-2015].

[111] https://wiki.linaro.org/WorkingGroups/KernelArchived/Specs/StoragePerfMMC-FS-compare, [Online; accessed 14-Mar-2016].