



Universidade Federal do Rio Grande do Norte
Centro de Tecnologia
Programa de Pós-Graduação em Engenharia Elétrica e de
Computação



Performance-Energy Trade-offs Prediction and Runtime Selection for Parallel Applications on Heterogeneous Multi-processing Systems

Demetrios A. M. Coutinho

Advisor: Prof. Dr. Samuel Xavier de Souza

Thesis submitted in partial fulfilment of the **Doctor of Science** requirements in the Graduate Program in Electrical Engineering and Computer Engineering of UFRN (area of concentration: Computer Engineering).

PPgEEC order number: M000
July 2021

Universidade Federal do Rio Grande do Norte - UFRN
Sistema de Bibliotecas - SISBI
Catalogação de Publicação na Fonte. UFRN - Biblioteca Central Zila Mamede

Coutinho, Demetrios A. M.

Performance-energy trade-offs prediction and runtime selection for parallel applications on heterogeneous multi-processing systems / Demétrios Araújo Magalhães Coutinho. - 2021.

111 f.: il.

Tese (doutorado) - Federal University of Rio Grande do Norte, Center of Technology, Graduate Program in Electrical and Computer Engineering, Natal, RN, 2021.

Orientador: Prof. Dr. Samuel Xavier de Souza.

1. Heterogeneous multi-processing - Tese. 2. Energy efficiency - Tese. 3. Pareto frontier - Tese. 4. Runtime energy manager - Tese. 5. Performance, power and energy model - Tese. I. Souza, Samuel Xavier de. II. Título.

RN/UF/BCZM

CDU 004.932

Acknowledgements

I would like to express my sincere gratitude to my PhD advisor, Professor Samuel Xavier, who provided me with outstanding mentorship to help me grow professionally and personally.

I would like to express my sincere gratitude to professor Kerstin Eder for her assistance and insightful feedback, which pushed me to sharpen my thinking and bring my work to a higher level.

I would like to offer my special thanks to PostDoc researcher Daniele de Sensi for sharing his expertise and the encouragement extended to me.

I would like to thank the research team at TSL-UoB for the excellent hospitality and great moments, especially Abanoub Ghobrial and Kyriakos Georgiou.

I would like to thank my lab mates at LAPPS/UFRN for the treasured coffee breaks conversations, especially Anderson Braulio, Alex Furtunato, Carla Santana, Italo Assis and Roger Araujo.

I would like to thank my dear friends Aluisio Igor, Bonfim Aquino, Diego Cirilo, Elenilson Vieira and Jefferson Queiroga for lifting my spirit throughout this journey.

I would like to thank my dear friends of “Toca da treta” for a cherished time spent together in social settings before the pandemic.

I am deeply grateful to my wife, who has been unwaveringly supportive and has been by my side through all my toils, absences and impatience.

I would like to thank my parents and sister for all their love and belief.

I want to thank Coordenação de Aperfeiçoamento de Pessoal de Nivel Superior - Brasil (CAPES) for financially helping part of this work.

Abstract

In the multi-core era, the size of the software operation space, i.e. hardware configurations (number of cores and operating frequency) that provide different software performance and energy consumption, is significantly larger. It becomes even more complex to choose a configuration that optimizes heterogeneous processors' performance and energy consumption. Heterogeneous multi-core architectures offer flexibility in different core types and voltage and frequency pairings, defining a vast design space to explore. Furthermore, energy efficiency solutions are crucial on smaller devices as they can lead to longer battery life and a better user experience, including more complex applications. This thesis proposes a methodology to find performance-energy trade-offs for single parallel applications with dynamically balanced workloads running on Heterogeneous Multicore Processing (HMP) systems with a single instruction-set architecture (ISA). Our method devises novel analytical models for performance and power consumption whose parameters can be fitted using only a few strategically sampled offline measurements. These models are then used to estimate an application's performance and energy consumption for the whole configuration space. In turn, these offline predictions define the choice of estimated Pareto-optimal configurations of the model, which are used to inform the configuration that the application should execute. The methodology was validated on an ODROID-XU3 board for eight programs from the PARSEC Benchmark, Phoronix Test Suite and Rodinia applications. Energy savings of up to 59.77%, 61.38% and 17.7% were observed compared to the performance, ondemand and powersave Linux governors, respectively, with higher or similar performance. This method aims to provide an optimal start point for a runtime energy manager to make better decisions according to the given application's performance and energy consumption requirements and running system. Therefore, this thesis also proposes a strategy using the Pareto-optimal configuration selected by our models as an appropriate start point for a runtime support framework called Nornir. This framework performs a local search dynamically for a more desirable configuration of cores and frequency adapting to workload fluctuations and external interference. Also, we extend our power model to predict the whole device's consumption, i.e. the sum of all internal components' consumption. This hybrid approach was employed on an ODROID-XU3 board on two multi-thread applications. Preliminary results shows that Nornir starting with Pareto configuration can achieve up to 50% of energy savings compared to random starting configurations. On average, we observed that the default Linux governors consumed up to $1.62\times$ more energy than Nornir using Pareto.

Keywords: heterogeneous multi-processing; energy efficiency; performance, power and energy model; Pareto frontier; runtime energy manager.

Resumo

Na era multicore, o tamanho do espaço de operação do software, ou seja, configurações de hardware (número de núcleos e frequência operacional) que fornecem diferentes desempenhos e consumos energéticos de um software, é significativamente maior. Além disso, torna-se ainda mais complexo escolher uma configuração que otimize o desempenho e o consumo de energia de processadores heterogêneos. Arquiteturas heterogêneas de vários núcleos oferecem flexibilidade em diferentes tipos de núcleos e de tensão e frequência, definindo um vasto espaço de design a ser explorado. Além do mais, soluções de eficiência energética é crucial em dispositivos menores, pois pode levar a uma vida útil mais longa da bateria e uma melhor experiência do usuário, incluindo aplicativos mais complexos. Esta tese propõe uma metodologia para encontrar balanços entre performance e energia para aplicações paralelas com cargas de trabalho balanceadas dinamicamente, em sistemas multiprocessadores heterogêneo com uma arquitetura de conjunto de instruções único (ISA). Modelos analíticos são elaborados para prever desempenho e consumo energético de aplicações cujos parâmetros podem ser ajustados usando apenas algumas medições estrategicamente amostradas. Esses modelos são então usados para estimar o desempenho de um aplicativo e o consumo de energia para todo o espaço de configuração. Por sua vez, essas previsões realizadas off-line são utilizadas para estimar as configurações ótimas de Pareto, que informam a configuração ideal que o aplicativo deve executar definida pelos modelos. A metodologia foi validada em uma placa ODROID-XU3 contando com oito aplicativos do PARSEC Benchmark, Phoronix Test Suite e Rodinia. Foram observadas economias de energia de até 59,77%, 61,38% e 17,7% em comparação com os governadores do Linux performance, ondemand e powersave, respectivamente, com desempenho superior ou semelhante. Este método visa fornecer um ponto de partida ideal para um gerenciador de energia para tomar melhores decisões de acordo com o desempenho do aplicativo e requisitos de consumo de energia numa determinada arquitetura de hardware. Dessa forma, esta tese também propõe uma estratégia usando a configuração Pareto-ótima selecionada por nossos modelos como um ponto de partida apropriado para um *runtime support framework*, chamado Nornir. Esse framework realiza dinamicamente uma busca local por uma configuração mais desejável de núcleos e frequência adaptando às flutuações da carga de trabalho e interferência externa. Além disso, estendemos nosso modelo de energia para prever o consumo de todo o dispositivo, ou seja, a soma do consumo de todos os componentes internos do hardware. Obteve-se resultados preliminares desta abordagem híbrida numa placa ODROID-XU3 em dois aplicativos de múltiplas threads. Nornir começando com a configuração de Pareto

pode alcançar até 50% de economia de energia em comparação com configurações iniciais aleatórias. Em média, observamos que os governadores padrão do Linux consomem até $1,62\times$ mais energia do que Nornir usando Pareto.

Palavras-chave: multiprocessamento heterogêneo; eficiência energética; modelo de desempenho, potência e energia; Fronteira de Pareto; gerenciador de energia em tempo de execução.

Contents

Table of Contents	i
List of Figures	v
List of Tables	vii
List of Symbols and Abbreviations	1
I Aim and Scope	1
1 Introduction	2
1.1 Performance and Energy Trade-offs Challenge	3
1.2 Objectives	5
1.3 Scope and Contributions	6
1.4 Publications	7
1.5 Thesis Structure	8
2 Background	9
2.1 Embedded Multi-Processor System	9
2.1.1 Processing Cores Count	10
2.1.2 Types of Multi-core Architectures	12
2.1.3 ARM big.LITTLE Architecture	13
2.1.4 Parallel Processing at Multi-core Systems	15
2.2 Software-Energy Management	16
2.2.1 DVFS and Power Consumption	17
2.2.2 Linux Power Governors	18
2.2.3 Energy Management Approaches and Aspects	19
2.3 Experimental Setup and Applications	27
2.3.1 Odroid-XU3 Platform	27
2.3.2 Odroid SmartPower2	29
2.3.3 Benchmark Applications	29

II	Finding Optimal Performance and Energy Trade-offs Methodology	33
3	Finding Optimal Performance and Energy Trade-off	34
3.1	General Overview	34
3.2	Application Performance Modelling	36
3.3	Power Consumption Modelling	38
3.4	Energy Modelling	40
4	Evaluation and Results	41
4.1	Hardware Design Parameters	41
4.1.1	The Perf Parameter	42
4.1.2	Power Consumption Parameters	43
4.2	Applications	44
4.3	Pareto Frontier Validation	44
4.4	Comparison against DVFS Governors	48
4.4.1	Normalized Comparison	48
III	An Offline and Online Hybrid Approach for Selecting Optimal Configurations	52
5	Selecting Optimal Configurations	53
5.1	Runtime Measurements and the <code>perf</code> Parameter	54
5.2	Fine Power Modelling and Whole-System Consumption	55
5.3	Energy Modelling	56
5.4	Runtime Selection	57
5.4.1	Analyze-Plan Algorithm	58
6	Evaluation and Results	62
6.1	Modelling Fitting Results	62
6.1.1	Power Consumption Parameters	63
6.1.2	Performance Parameters	63
6.1.3	Modelling Validation	65
6.1.4	Pareto Frontier Validation	67
6.2	Hybrid Approach Validation	71
6.2.1	Pareto X Non-Pareto	71
6.2.2	Linux Governors Comparison	74
IV	Conclusions and Future Research Directions	78
7	Conclusion	79
8	Future Work	81

List of Figures

2.1	General Organization of an Embedded multi-processor with multiples processing elements.	10
2.2	Historical trends in CPU performance.	11
2.3	Hot-spot power density scaling with feature size.	11
2.4	Simplified representation of symmetric multi-core processors.	13
2.5	Simplified representation of asymmetric multi-core processors.	13
2.6	ARM big.LITTLE System.	14
2.7	Conventional ARM big.LITTLE and DynamIQ System comparison.	15
2.8	Speedup curves varying the number of processors/cores for different portions of parallel code in an application, using Amdahl's law	17
2.9	Thread mapping examples on a eight core heterogeneous architecture.	21
2.10	ODROID-XU3 board highlighting Samsung Exynos5422 chip internal organization.	27
2.11	Odroid SmartPower2 board detail.	29
2.12	ODROID-XU3 connected to Odroid SmartPower2.	30
3.1	Flowchart of the methodology applied for predicting the optimal performance and energy efficiency trade-offs of a parallel application.	35
4.1	Power consumption from LITTLE and big clusters for every stress-ng method.	42
4.2	Performance parameter perf fitted for the architecture using a fixed number of operations of the stress-ng tool for one core of each cluster.	43
4.3	Estimated and measured Pareto frontier compared with the default governors, and all possible modeled configurations.	47
4.5	Normalized energy consumption by the measured Pareto configuration that provides the least energy consumption (MPLE) for each application.	49
4.6	Normalized performance by the measured Pareto configuration that provides the highest performance (MPHP) for each application.	49
4.4	Estimated and measured Pareto frontier compared with the default governors removing all possible modeled configurations.	51
5.1	Flowchart highlights the extended steps to suit our approach of selection configurations using Pareto configurations at runtime. Also, it shows the main changed features from the power and energy modelling.	53
5.2	General Architecture of Nornir Framework.	57

5.3	An example of Nelder–Mead simplex search over the Himmelblau’s function	59
5.4	An example of Nornir’s threads mapping policy.	61
6.1	Measured and Modelled power consumption of 100 Halton and random configurations ordered by modelled power.	64
6.2	Measured and modelled metric values comparison for 50 Halton and random configurations for Black-Scholes application ordered by modelled time.	66
6.3	Measured and modeled metric values comparison for 50 halton and random configurations for Bodytrack application ordered by modeled time.	67
6.4	Estimated and measured Pareto frontier compared with the default governors and all possible modelled configurations.	69
6.5	Measured execution time and energy consumption for HP and LE Pareto, Nornir HP and LE with Pareto and random configurations.	72
6.6	Boxplot graph for 25 random configurations of Nornir HP and LE for Black-Scholes and Bodytrack applications.	73
6.7	The available Linux governors’ execution time and energy consumption considering all cores big and LITTLE combination for Black-Scholes	75
6.8	The available Linux governors’ execution time and energy consumption considering all cores big and LITTLE combination for Bodytrack	76
6.9	Normalized execution time and energy consumption by the respective least measured value.	77

List of Tables

2.1	Related work categorized by various aspects.	26
2.2	Voltage and Frequency relation for Exynos5422	28
2.3	Characteristic details of each application based on	32
4.1	The fitted parallel fraction values of each application.	44
4.2	Pareto Frontier energy and performance variations concerning the measured values and the total number of Pareto configurations. . . .	45
4.3	The average standard deviation and mean absolute percentage error of performance and energy consumption of the estimated Pareto frontier against measured data.	48
4.4	Percentage of performance gains and energy savings when compared to the measured Pareto configurations that give the highest performance (MPHP) and the least energy consumption (MPLE), respectively, for all applications.	50
5.1	Main requirements specified in this thesis.	58
6.1	Mean absolute percentage error (MAPE) between the measured and modelled time, power and energy between values, including the average standard deviation of the measured data for Black-Scholes and Bo-dytrack	68
6.2	Pareto Frontier energy and performance variations concerning the measured values, mean absolute percentage error (MAPE) between the measured and modelled performance and energy consumption of the Pareto Frontier and average standard deviation of the measured data for each application.	70
6.3	The percentage of performance gains and energy savings when compared to the MPHP and MPLE, respectively, for each application. . .	70

Part I

Aim and Scope

Chapter 1

Introduction

The report from a non-profit organization, shift project, affirms that Information and Communication Technology (ICT) energy consumption increases by about 9% per year (period 2015 to 2020) (Ferreboeuf et al. 2019). It is above the expected case scenario predicted in 2015 by Andrae and Edler (Andrae & Edler 2015) of 4% and below their worst-case scenario (10%). This includes large systems such as data centres and telecommunication networks and small systems as terminals (computers, tablets, smartphones, SmartTVs) and IoT sensors (Internet of Things). Furthermore, the ratio of digital and global energy consumption (E-Ratio) has risen from 1.9% in 2013 to 2.7% in 2017 and reached 3.1% in 2019. E-ratio has exponential growth and can reach 6% in the worst case in which the progress of energy efficiency is decelerated. The lack of energy efficiency limits the functionality of the smallest devices powered by batteries. In addition, it limits the applicability of the largest systems due to the economic and environmental costs of energy.

Roughly, we can classify the energy efficiency solutions into three different levels: chip, system and program level—the chip-level concerns developing technologies to produce high-performance and low-power CMOS. Nevertheless, the scaling of transistors to smaller dimensions is not awaited to reduce the switching energy significantly (Fagas et al. 2017). Next, the system-level (architecture and microarchitecture) involves the various technologies integration of the existing processing devices. The energy consumption of a CMOS technology design depends on how specific the hardware architecture is, optimized for a single or a few tasks, or implemented in a general way to perform many different calculations. For instance, Applications Specific Integrated Circuits (ASICs) is an Integrated Circuit (IC) chip optimized for a particular task providing the lowest energy consumption. However, such devices produce no flexibility and cannot be reprogrammed. For general-purpose computing that must execute various tasks, minimize energy consumption is significantly more difficult. Finally, the program level (software) affects the amount of energy consumption in a system in various ways. The software techniques include optimization on the instruction level, compiler level, and operating system level. Possibly the most significant is how the system operates at the request of software as processor cores, data communication, or displays can draw more or less energy. The potential to minimize energy consumption is much more prominent when targeting software because it controls most of the hardware functionalities (Eder et al. 2016). For instance, it is

estimated that up to 80% of an embedded system’s total energy consumption is due to software-related activities (Luo et al. 2009). Indeed, poorly written or managed software can make even the most energy-efficient hardware behave very inefficiently regarding energy consumption.

Eventually, in the next Section, we explain the interwoven relationship between performance and energy consumption of a processor in the multi-core era, how it affects parallel software and the problem statement. In Sections 1.2 and 1.3, we state the objectives and the contributions of this thesis, respectively. In Section 1.4, we list the publications that cover different parts of this thesis, and finally, in Section 1.5, we outline the structure of this manuscript, which is divided into Part I (introduction and background), Part II/Part III (methodologies and results) and Part IV (conclusion).

1.1 Performance and Energy Trade-offs Challenge

For decades computing relied on single-core, which means that optimal configuration was only one core and maximum frequency. Instead of minimizing energy consumption, the main concern was to improve computational capacity, reducing processing time. Moreover, energy efficiency only turned into a genuine concern because power density grew faster than the delivered performance gains (Patterson & Hennessy 2014, Fagas et al. 2017). Thus, primarily motivated by its incapacity to increase single-core performance and power- and thermal-related restrictions, the industry shifted towards multi-core processors. The multi/many-core architectures have also been shown as an encouraging solution for energy-efficient computing (Vangal et al. 2008, Truong et al. 2009).

The theoretical performance of the newer generations of processors expects to grow in the multi-core era (Georgiou et al. 2017). However, harvesting this performance requires an extra programmer’s effort to use the additional processing elements efficiently (Coutinho & Xavier-de-Souza 2013). Furthermore, energy efficiency becomes more challenging because coding for parallel hardware performance is necessary but not enough. Several configurations of the same architecture can deliver similar performance while yielding very different energy profiles. Even with an optimized parallel code for a given parallel architecture and its cores to maximum performance, it can result in energy-optimal configurations with equivalent performance being completely neglected.

Each configuration poses a different trade-off between performance and power, impacting energy (power $\times$ time) in different ways. The dissipated power increases linearly with the number of cores and cubically by scaling up the operating frequency (Tan et al. 2009, Benini et al. 2000, Pouwelse et al. 2003). A parallel software’s performance often sub-linearly increases with the operating frequency due to the memory wall’s effect (Furtunato et al. 2020). Also, power and performance are related to the microarchitecture complexity of the multi-core chip design. According to Pollack’s rule, performance growth is limited by the square root of the increase in microarchitecture complexity (Pollack 1999). So, adding complexity to a

core often means more power consumption but diminishing returns in a performance boost. One way to improve the balance between performance and power consumption is to use different single and multi-core processors, which differ in terms of complexity, on a single chip. Single Instruction-Set Architecture (Single-ISA) Heterogeneous Multi-Processing (HMP) architecture is a CMP (Chip Multi-processor) that comprises cores of varying size, performance, and complexity coherently interconnected. Furthermore, this kind of architecture is known for delivering a significantly higher performance-power ratio than the homogeneous architecture (Kumar et al. 2004, Mittal 2014, Fagas et al. 2017).

There exist many commercially available single-ISA HMP designs in the embedded and mobile world (Pham et al. 2006, Chitlur et al. 2012). ARM big.LITTLE is the first attempt to leverage dark silicon in a commercial product (Modarressi & Sarbazi-Azad 2018), and it is one of the commercial widely used heterogeneous architectures (Nguyen 2013). It consists of two clusters with two types of processing cores, each one containing one or more cores. Cluster type *big* is composed of higher performance larger cores that are also more power-hungry. Cluster type *LITTLE* consists of smaller cores that are slower and more energy-efficient. The increased sophistication of this type of architecture delivers a challenging task to develop better energy management solutions. It is increasingly more complicated to find the number of cores, operating frequency, and voltage that optimize performance and energy consumption to meet the requirements of a given application (Wang et al. 2015, Koufaty et al. 2010, Saez et al. 2017). Running a multi-threaded application only on big cores may not justify the performance gain related to energy consumption. Also, using only small cores may not be the best choice for reducing energy consumption, as the application may suffer a significant increase in execution time. Besides, it is possible to scale up or scale down the core frequencies to improve the performance or save energy. Each arrangement, or configuration, of operating frequency and the number of available cores of each cluster type to execute a given application may produce a different performance-energy trade-off.

A simple approach to obtaining one or more configurations in HMPs that provide the most beneficial performance-energy trade-offs is to collect a given application's performance and energy consumption in all available configurations. However, in current heterogeneous systems, this would require a higher cost to be performed online when compared to their counterparts. For instance, a typical heterogeneous processor as Samsung Exynos 7420 (Samsung 2018) with ARM big.LITTLE architecture has an operation space of 4032 points¹. Contrasting with the homogeneous multi-processing systems, which achieves only hundreds of different operation points (De Sensi 2016, Furtunato et al. 2020). Such a large number of configurations could be unfeasible for online approaches since, in many cases, the search process for the best configuration can outlast the execution itself. Even if executed offline, although it could be possible to evaluate all configuration, the expected overheads

¹Exynos 7420 has eight cores (4 big + 4 LITTLE), 14 and 12 levels of frequency for big and LITTLE cores. Total number of possible configurations is given by (# possible counts of big) $\times$ (5 $\times$ 14) $\times$ (# possible counts of LITTLE) $\times$ (5 $\times$ 12) - (possible frequencies)(12 $\times$ 14) = 4032

makes the method unfeasible.

Therefore, finding the energy-optimal software operation so that both performance and energy consumption are most effective requires a careful configuration choice. It is a relevant problem, and many solutions have been proposed as described in Section 2.2.3. Ultimately, this thesis's scope consists of finding operation points (processor configuration) whose performance and energy are optimal for a given parallel software at a Single-ISA HMP system.

1.2 Objectives

The thesis is divided into two parts. The first part details the central methodology developed throughout this research. The second is a combination of the estimated Pareto configurations with a dynamic reconfiguration platform for parallel applications.

1. Finding Optimal Performance and Energy Trade-offs Methodology

Develop an offline methodology to find optimal performance and energy trade-offs tailored to a parallel application with dynamically balanced workloads for a given single-ISA HMP.

- Create agnostic power model for single-ISA two-cluster processing chip devices;
- Create a performance model for parallel applications;
- Measure power and execution time data from several applications;
- Validate Pareto curve in hardware;
- Compare methodology against standard Linux dynamic voltage and frequency scaling (DVFS) governors.

2. An Offline and Online Hybrid Approach for Selecting Optimal Configurations

Explore estimated Pareto configurations to dynamically find optimal performance and power trade-offs during the application's execution, considering the entire embedded system's energy consumption.

- Extend the power model to include the entire embedded device;
- Validate the estimated energy consumption and performance among the configurations selected by the Pareto Frontier;
- Compare the hybrid approach starting with a Pareto configuration against random guesses;
- Compare the hybrid approach starting with a Pareto against standard Linux DVFS governors.

1.3 Scope and Contributions

To address the above objectives, the main contributions that have been made during the course of this research are briefly introduce.

1. **A practical methodology to find Pareto-optimal performance and energy trade-offs for HMP**

Considering the vast diversity of performance-energy trade-offs that the large configuration space of HMP systems yields for a given application, the understanding of these trade-offs becomes critical to energy-efficient software development and operation (Jin et al. 2017). We developed a novel methodology to find the best performance and energy trade-off configurations of parallel applications with dynamically balanced workloads running on HMP systems. We propose simple analytical models for performance and power that are fitted offline using a few configuration space samples. These models only require execution time and power measurements, avoiding performance hardware counters, which may not be accessible on all architectures. The combination of the performance and power models' outcomes generates the energy model. Then, the performance and energy models are used to assess the whole configuration space, commonly in the order of thousands of points, and the outcomes are used to select the configurations that lay on the performance and energy Pareto frontier, whose number is often in the order of tens of points. Pareto frontier can be used to either look for the overall best energy configuration or the best energy configuration for a given performance. Hence, this methodology can estimate the best relationship between performance and energy consumption without requiring an expensive exploration of the full search space. Besides, this technique can be easily ported to other applications running on the same HMP system, since the power model of the HMP system can be reused. The performance model can be re-fitted based solely on the new application's execution time. In the Part II, we show the methodology employed in an ODROID-XU3 board on eight multi-thread applications. The results show that the proposed methodology and models can successfully estimate the Pareto-optimal configurations consistently with measured data. Also, it shows that the specific knowledge of the application performance and the system power embedded into analytical models can deliver significantly better results than conventional DVFS governors.

2. **An offline and online hybrid approach for selecting configurations dynamically regarding the whole device consumption at runtime**

This approach leverages the previous specific knowledge of the application performance and the system energy provided by our offline method to deliver better runtime choices. To select appropriate CPU configuration dynamically, we propose a hybrid approach between our offline method and a runtime support framework, called Nornir (De Sensi et al. 2018). Nornir performs a local search

for desirable configuration of cores and frequency using any predicted Pareto-optimal configuration as an appropriate start point. The configuration changes on the fly throughout the execution to adapt to workload fluctuations and external interference. Also, it is possible to express specific requirements that dictate the framework behaviour regarding performance or energy consumption. To improve our power models' accuracy, we consider the discrete values of the chip's voltage-frequency operation. Importantly, we intend to expose the Pareto-optimal configurations' impact, considering the energy consumption for a given application regarding a whole device. Predicting the whole device's energy consumption is challenging and very relevant for battery-based applications since the sum of all internal components' consumption effect the battery. Ultimately, the hybrid approach was employed in an ODROID-XU3 board on two multi-thread applications. The preliminary results indicate that Nornir's local search strategy algorithm potentially can lead to configurations with better performance-energy trade-offs using estimated Pareto-optimal configurations as an initial guess. Also, we observe that this strategy can achieve better results than conventional DVFS governors.

3. Software Tool

The methodology described in Part II (Finding Optimal Performance and Energy Trade-offs Methodology) has been implemented to be easily applied to any Linux operating system on the ODROID-XU3 platform. The code is available in the LAPPS/UFRN research group repository: <https://gitlab.com/lappsufrn/XU3EM>. For Part III (An Offline and Online Hybrid Approach for Selecting Optimal Configurations) which has the extension of the methodology to work symbiotically with Nornir is available at the address <https://gitlab.com/lappsufrn/iperfect>. IPerfEcT (Identifying heterogeneous multi-processing configurations for parallel Performance and whole Energy device consumption Trade-Offs) is an evolution of XU3EM, containing several scripts regarding performance and power measurement, modelling and Pareto frontier generation for ODROID-XU3 and NVidia Jetson TX2.

1.4 Publications

Some parts of this thesis have been published in the following publications:

1. **Coutinho, D. A. M.**, de Sensi, D., Xavier-De-Souza, S., "IPerfEcT - Identifying heterogeneous multi-processing configurations for Performance and Energy consumption Trade- Offs", HiPEAC – European Network on High Performance and Embedded Architecture and Compilation - virtual, 2021. <https://gitlab.com/lappsufrn/iperfect-tutorial>
2. **Coutinho, D. A. M.**, de Sensi, D., Lorenzon, A. F., Georgiou, K., Nunez-Yanez, J., Eder, K., and Xavier-De-Souza, S., "Performance and energy trade-offs for

parallel applications on heterogeneous multi-processing systems”, *Energies*, v. 13, p. 2409, 2020. <https://www.mdpi.com/1996-1073/13/9/2409>

3. **Coutinho, D. A. M.**, de Sensi, D., Georgiou, K., Eder, K., Xavier-De-Souza, S., “Predicting Energy-Performance Trade-offs in Single-ISA HMP”, *HiPEAC – European Network on High Performance and Embedded Architecture and Compilation*, Bologna, Italy, 2020. <https://gitlab.com/lappsufrn/hipeac>
4. **Coutinho, D. A. M.**, Georgiou, K., Eder, K., JNunez-Yanez, J. and Xavier-de-Souza, S., “Performance and Energy Efficiency Trade-Offs in Single-ISA Heterogeneous Multi-Processing for Parallel Applications”, 2019 IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC), Cuzco, Peru, 2019, pp. 232-233. <https://ieeexplore.ieee.org/document/8920384>
5. **Coutinho, D. A. M.**, Nikov, K., Georgiou, K., Nunez-Yanez, J., Eder, K., Xavier-De-Souza, S., “Analytic Models for Exposing Performance and Energy Efficiency Trade-Offs in Heterogeneous Multi-Processing”, *HiPEAC – European Network on High Performance and Embedded Architecture and Compilation*, Valencia, Spain, 2019.

1.5 Thesis Structure

This thesis is organized into four parts, described as follows:

- Part I** introduces the thesis, and Chapter 2 provides background on embedded multi-processor systems and heterogeneous architecture. Also, it shows related work and a review of software-energy management approaches and techniques. Eventually, it introduces an experimental setup and the applications used to perform our experiments.
- Part II** describes the offline methodology to find Pareto-optimal Performance and Energy Trade-offs contribution. Chapter 3 describes the general overview, the performance, power and energy modelling. We then evaluate this method in Chapter 4, over a comprehensive set of real applications, and compare the Pareto frontier with DVFS governors.
- Part III** describes the hybrid approach for selecting optimal configurations at runtime. Chapter 5 extends the offline method proposed in Chapter 3 to improve the models and adapt to the runtime framework. In chapter 6, we evaluate the extended models and validate the Pareto frontier. Also, we validate the hybrid approach starting with Pareto-optimal configurations and compare it against random configurations and Linux DVFS governors.
- Part IV** concludes this thesis, providing a summary of the key contributions and outlining possible future directions for this research.

Chapter 2

Background

This chapter provides some necessary background and review of existing approaches to software-energy management. First, Section 2.1 describes architectural innovations to address the performance and power demands of modern embedded multi-processor System-on-Chip (SoC) platforms. Next, Section 2.2 covers energy and power management approaches for heterogeneous single-ISA architectures, as it is the focus of this thesis. Eventually, Section 2.3 presents the experimental setup and benchmark applications used in this research.

2.1 Embedded Multi-Processor System

Multi-processing is often used in embedded computing systems since it enables us to meet the performance, cost, and energy/power consumption requirements usually needed for smaller in size and limited in power (Wolf 2014). Embedded multi-processors are used by platforms that contain multiple, typically heterogeneous processing elements (PE) with specific functionalities reflecting the need of the expected application domain and a memory hierarchy. These components are linked to each other by an on-chip Interconnection, such as buses and Networks-on-chip (NoCs) (Dave 2017). These designs meet the performance demands of telecommunication architectures, network security, multimedia applications and other fields while limiting the power consumption through specialised PE and architecture (Ben Abdallah 2017). Embedded SoCs typically target edge computing markets, the Internet of Things (IoT) and industrial IoT.

The options for designing an embedded multi-processor are more diverse and complicated than for the typical multi-processor. Servers typically use symmetric multi-processors manufactured of uniform memory and identical processing elements. However, the embedded system is frequently designed considering programming complexity and cost/performance/energy/power trade-offs by combining several PEs and increasing the system's diversity. For example, the processing elements can be Graphics Processing Units (GPUs), Central Processing Unit (CPU), Digital Signal Processing (DSP), Vision Processing Unit (VPU) or specific co-processors. Typically, co-processors help the CPU with particular tasks like, e.g., floating-point arithmetic, string processing, encryption and I/O interfacing.

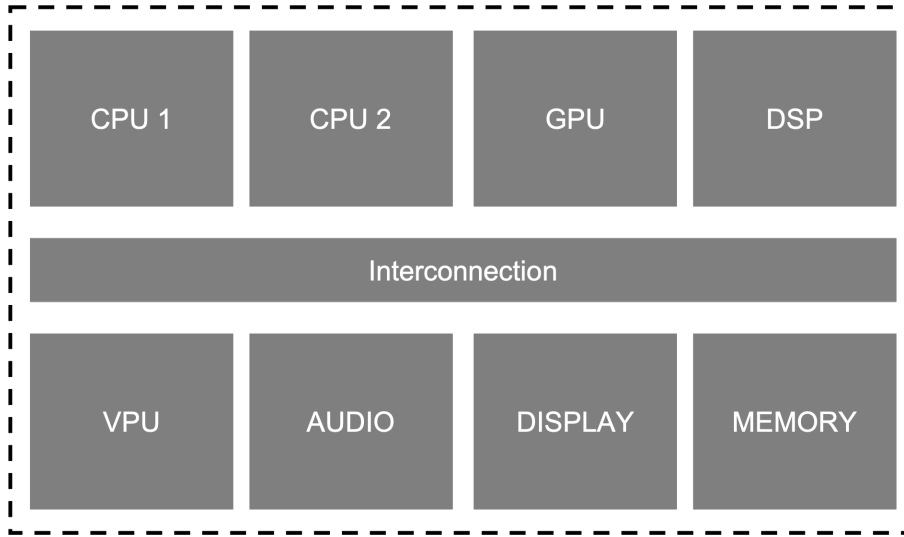


Figure 2.1: General Organization of an Embedded multi-processor with multiples processing elements.

Figure 2.1 shows an example of a modern embedded heterogeneous multi-processor system that includes several PE. Because many embedded systems are very cost-sensitive, all components, including the central processor, the various co-processors, the memory, peripherals, interconnect, usually are fabricated in a System-on-Chip (SoC). It saves area and cost, which is crucial for most embedded System. An SoC that is also a multi-processor is known as a Multi-Processor System-on-Chip (MPSoC).

2.1.1 Processing Cores Count

Processors have evolved over more than 40 years. Among the most significant changes, we highlight the increase in operation frequency to meet performance demands. For long decades, techniques such as superscalar, pipelining, and Out-of-Order (OoO) execution allowed the CPU to run at higher clock speeds to improve throughput (Tanenbaum & Austin 2005, Moore 1998). In the '80s and '90s, frequency scaling assisted in pushing the increasing performance demands. Consequently, processor manufacturers were able to double single-threaded performance approximately every 18 months, which has added an increased amount of complex logic to the processor core (Moore 1998).

The operating frequency scaling means an increase in the total power dissipated and, consequently, in the processing chip's heat generation. For a long time, new process technologies provided for smaller and less energy-consuming transistors. New processors were being built with an increasing number of transistors, as shown in the upper curve of Figure 2.2. However, the heat generated is further aggravated by the decrease in the chip's size, and the transistors count growth, making the power rise faster than the integration area (Sutter 2005). Therefore, causing an increase in power density. Figure 2.3 shows that the hot-spot's power density, which

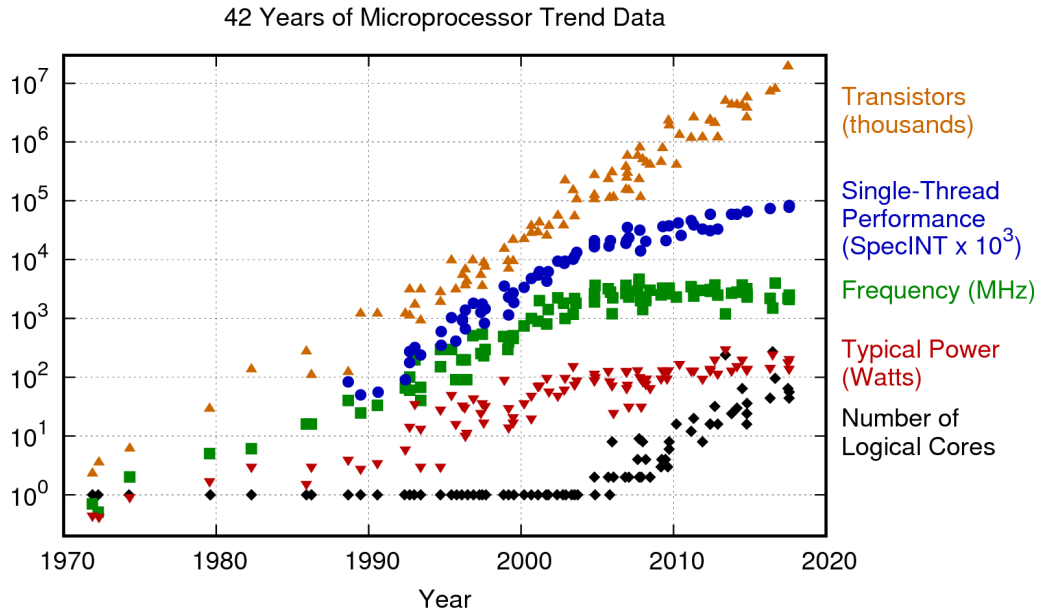


Figure 2.2: Historical trends in CPU performance (Rupp 2018).

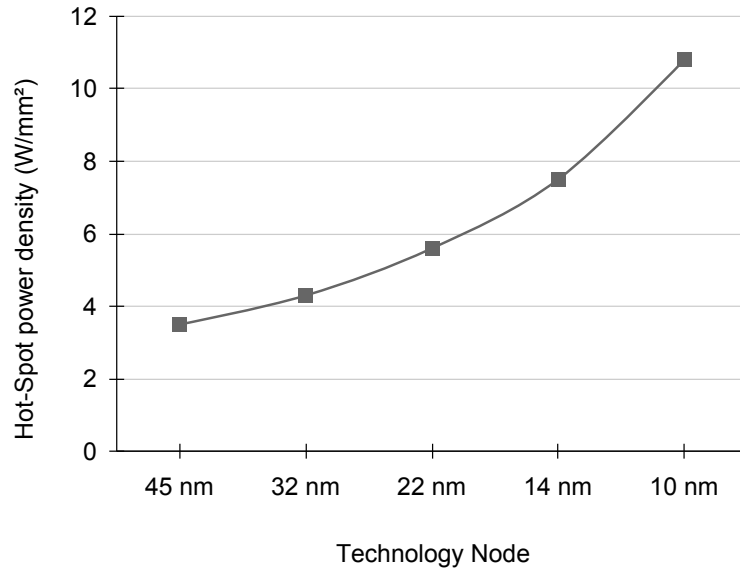


Figure 2.3: Hot-spot power density scaling with feature size. Reprinted from Huang et al. (Huang et al. 2011).

is heavily used parts of the cores, can rise by up to 3x from 45nm to 10nm.

In other words, there was an end to Dennard's Law (Dennard et al. 1974, Esmaeilzadeh et al. 2012), according to which the miniaturization of transistors should not affect the power density in integrated circuits, since the power would be proportional integration area. As more transistors are crammed together, the power density became a problem. In such a way, the conventional cooling solutions could not dis-

sipate the generated heat caused by increased power in a tiny area. These adverse effects made the increasing and continuous performance of single-core processors unfeasible, reaching a limit known as the Power Wall (Patterson & Hennessy 2014, section 1.7 The Power Wall). A type of barrier that hampered efforts at the time for better processor performance by power restrictions.

For performance growth to be maintained, processor manufacturers needed to modify their processor production strategy. A new paradigm was adopted in which the chips have two or more processing cores. Thus, it was possible to slow down the growth rate of the clock internal speed, reducing the heating and power dissipation problem. Since then, efforts have been made to double the number of cores each generation, calling the era multi-core (Borkar 2007). Figure 2.2 shows how the processors' characteristics can be associated with the periods before and after early 2000, with very different growth rates for the relative performance, the frequency, the power, and the number of cores per chip. Notably, Moore's law remains valid, but the number of transistors is proportionally associated with the rising number of cores.

While improving performance, processor manufacturers must not exceed the physical limits of power dissipation. Hence, energy efficiency is critical to obtaining additional performance gain. The need to scrutinizes parallel architecture technologies is vital, due to its inherent energy efficiency (Ramanathan 2006). For instance, embedded processors built-in battery-operated mobile devices have always been energy-constrained. In such circumstances, improved energy efficiency would allow for longer battery life.

2.1.2 Types of Multi-core Architectures

The implementation of processors using multi-core architecture proved to be adequate for most application domains. Moreover, multi-core architecture offers several advantages over single-core architecture (Singh 2013). For example, high transistor density allowed by future process technologies can have more cores while keeping core complexity the same. Moreover, computational performance can be improved with the number of cores, and some cores can be disabled/enabled depending on demands. Therefore, several multi-core chips have been developed, targeting different application domains (Oshana 2009).

Based on the differentiation in the complexity of the cores, a classification of the various types of multi-core processors can be made as follows.

Symmetric multi-core or homogeneous

In this type of processor, all cores are identical and use the same resources without distinction. At first, the cores operate at the same voltage and frequency. After that, however, there is the possibility of differentiating them and turning off one core while the others remain operating. In Figure 2.4, two images represent simplified homogeneous multi-core processors with each rectangle corresponding to a core.

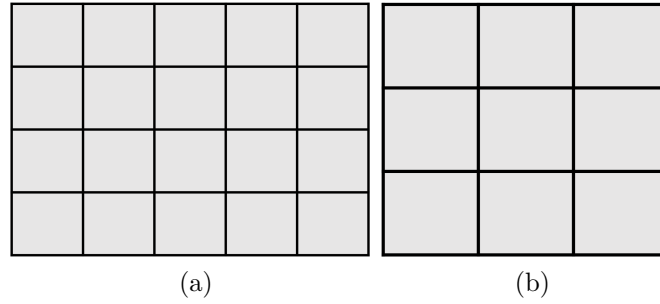


Figure 2.4: Simplified representation of symmetric multi-core processors.

Asymmetric multi-core or heterogeneous

In this processor type, the cores are not identical, and at least one is different from the others. For example, a single core is large and more powerful than the others, while the others are small and identical. In this case, the serial section of the code is executed by the powerful core. Moreover, the parallel part is divided among all the cores, including the participation of the big core. As in the previous item, Figure 2.5 depicts two simply heterogeneous multi-core processors. The first image (on the left) shows 15 cores, 14 small identical to each other and one large core with a capacity corresponding, by hypothesis, to six small cores. The second shows nine cores, eight small and identical to each other, and one large core with a capacity corresponding, by hypothesis, to 12 small cores.

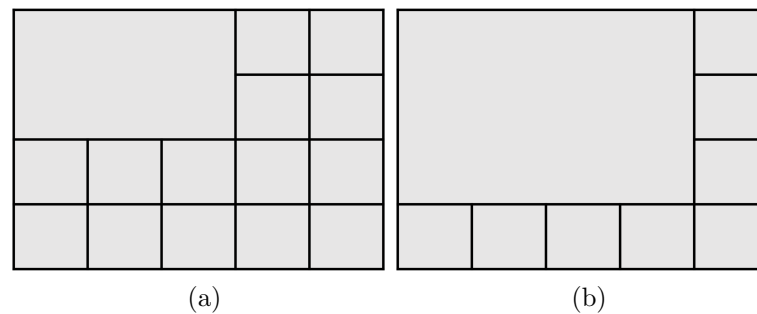


Figure 2.5: Simplified representation of asymmetric multi-core processors.

2.1.3 ARM big.LITTLE Architecture

ARM big.LITTLE SoC was first launched in October 2011, turning it the world's first heterogeneous processing technology to enter the mobile market (Govind Wathan 2017). The system consists essentially of a heterogeneous compute architecture that tightly combines two different but fully compatible CPU core types, a 'big' CPU cluster tuned for high-performance and a 'LITTLE' CPU cluster for low power consumption. The two processors use the same instruction set architecture to execute the same compiled code. Furthermore, all cores are connected to the same

memory subsystem so that tasks can be migrated from one type to another on the fly (Nguyen 2013).

The purpose is to produce a multi-core processor that is more suitable for dynamic computing needs and use less power than voltage-frequency scaling alone. Computation intensive threads that demand significant amounts of processing performance can be allocated to the cluster big. Memory intensive threads or that do not offer a time-critical to the user can be executed on cluster LITTLE (ARM 2012). For instance, while demanding applications like a web browser, multimedia and gaming run at cluster big, low energy demand applications, like phone calls, messaging, and email apps run at cluster LITTLE.

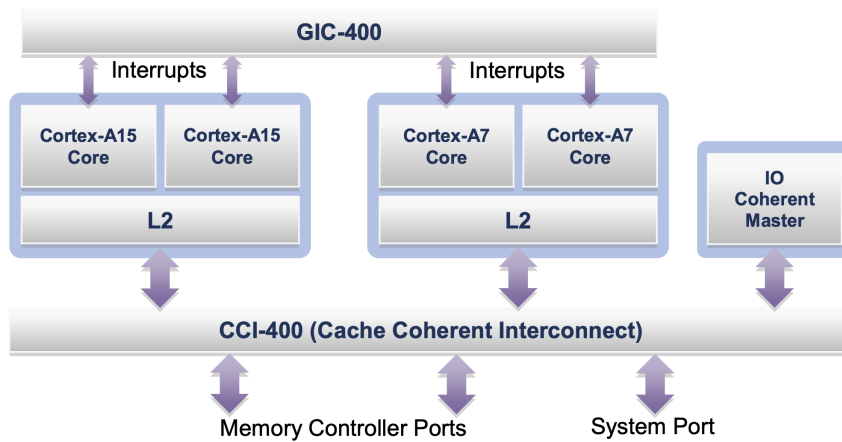


Figure 2.6: ARM big.LITTLE System (Greenhalgh 2011).

As shown at Figure 2.6, ARM big.LITTLE can be produced in association with the Cortex-A7 ¹ (LITTLE) intended to be architecturally compatible with the Cortex-A15 ² (big). Each processor is connected using a Cache Coherent Interconnect (CCI-400) and a generic interrupt controller (GIC-400) (ARM 2014). GIC allows interrupts to be migrated between any cores in the Cortex-A15 or Cortex-A7 clusters, and CCI-400 interconnect assists full coherency between Cortex-A15 and Cortex-A7. Besides, it has IO coherency for components such as a GPU. Both Cortex-A15 and Cortex-A7 implement the complete 32-bit ARMv7A architecture. Both CPUs can then execute all instructions in an architecturally consistent way, but with different performances. Both processors can be configured up to four cores, and integrate a level-2 cache inside the processing cluster.

It is in the microarchitectures that the differences between Cortex-A15 and Cortex-A7. While Cortex-A7 is an in-order, non-symmetric dual-issue processor with a pipeline length of between 8-stages and 10-stages (ARM 2013a, ARM 2013b), Cortex-A15 is an out-of-order sustained triple-issue processor with a pipeline length of between 15-stages and 24-stages (ARM 2011, Lanier 2010).

¹<https://developer.arm.com/ip-products/processors/cortex-a/cortex-a7>

²<https://developer.arm.com/ip-products/processors/cortex-a/cortex-a15>

The mobile market quickly adopted this technology, where power efficiency and battery life is crucial to user experience (Govind Wathan 2017). Commercial examples of this architecture include the entire Samsung Exynos line (Samsung 2019), some models from Qualcomm’s Snapdragon (Qualcomm 2019) and Apple processors, starting with the iPhone 8 with the A11 Bionic (Apple 2017). Such processors are used on mobile platforms or embedded devices.

This architecture paradigm was so well adopted that ARM announced DynamIQ as the successor to big.LITTLE in May 2017 (Wathan 2017). DynamIQ packages high-performance big CPUs and high-efficiency LITTLE CPUs into a single cluster with a shared coherent memory. All task migrations between big and LITTLE CPUs occur within a single CPU cluster by shared memory with the aid of an upgraded snoop management system, resulting in improved energy efficiency (Muralidhar et al. 2020).

ARM DynamIQ allows for mixing and matching Cortex-A75³ (big) and Cortex-A55⁴ (LITTLE) CPU cores, with up to eight cores total in a cluster (Triggs 2017). So rather than achieving a typical octa-core design using two clusters, DynamIQ can now achieve this with one. It produces many benefits, both in terms of performance and the cost-effectiveness of specific designs. Figure 2.7 shows that we could end up seeing 1+3, 1+4, 1+6 or 2+6 DynamIQ CPU designs that offer up better single-threaded performance than low and mid-tier SoCs (Triggs 2017). In addition, manufacturers can produce different frequencies and individually power each core.

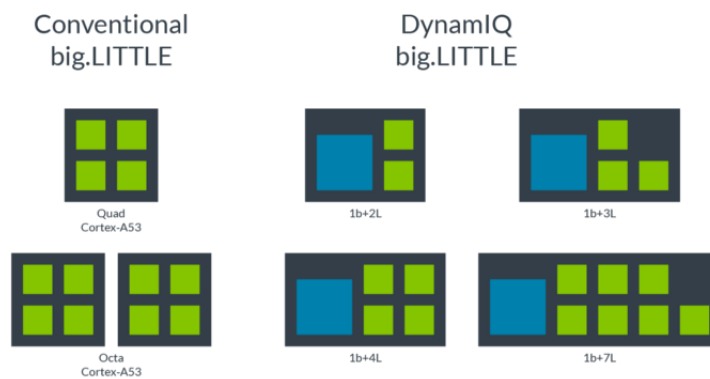


Figure 2.7: Conventional ARM big.LITTLE and DynamIQ System comparison (Shyamkumar 2019).

2.1.4 Parallel Processing at Multi-core Systems

In general terms, a parallel program running at a multi-core system can be divided into two portions. A parallel portion (f) represents a parallelizable percentage of the code, and a serial or sequential portion ($1-f$) represents how much of the code

³<https://developer.arm.com/ip-products/processors/cortex-a/cortex-a75>

⁴<https://developer.arm.com/ip-products/processors/cortex-a/cortex-a55>

cannot be parallel executed. These fractions' values are important factors since they affect some indicators related to the execution of the code in the multi-core processors, such as the PET set elements: Power, Energy consumption and Task execution time (Barros 2016). Additionally, the PET costs can also be affected by several other factors, such as processor architectures, the active number of cores and frequency operation.

Undoubtedly, multi-threaded/parallel processing of software delivers real gains in the many-core era. According to Amdahl (Amdahl 1967), the benefit of parallel application only occurs in the portion that can be parallelised, limiting the speedup to a value that depends on the time needed to execute the sequential section $(1-f)$ of the application. Consequently, the serial fraction limits parallel processing, even using an enormous number of processors/cores. Mathematically, according to the fraction f and the number of active processors/cores p , Amdahl's law says that Speedup is delimited at the value given by

$$S_p = \frac{1}{(1-f) + \frac{f}{p}}. \quad (2.1)$$

Generally, large speedup value brings the idea of great success at parallelising a sequential algorithm, reflecting its theoretical parallel potential. The speedup limitations can be represented by hardware or operating system restrictions and even the developer's code parallelising skill. It is such a important metric which can be used to associate with power models for conducting energy consumption assessments in multi-core processors (Barros et al. 2015).

Figure 2.8 shows the speedup curves generated using the different number of processors at various levels of parallel fraction f . For instance, if 95% of the program can be parallelised, the maximum speedup would be $20\times$ from 4096 processors. In other words, the 5% (serial section) cannot be improved further even by increasing the number of processors. However, the speedup can be increased by accelerating the sequential part's execution on a powerful processor specialised to higher performance than its counterparts. This is achievable using heterogeneous multi-core processors, where several portions of the application can exploit distinct features of different types of cores. Consequently, heterogeneous MPSoCs have displayed promising computing alternatives to achieve significant improvement at PET costs over their homogeneous counterparts (Hill & Marty 2008).

2.2 Software-Energy Management

This Section describes the Dynamic Voltage and Frequency Scaling (DVFS) technique and its relation to power modelling of an embedded system. Also, we present the existing Linux governors for managing power consumption, and then we show a review of existing energy management approaches in multi-core architecture.

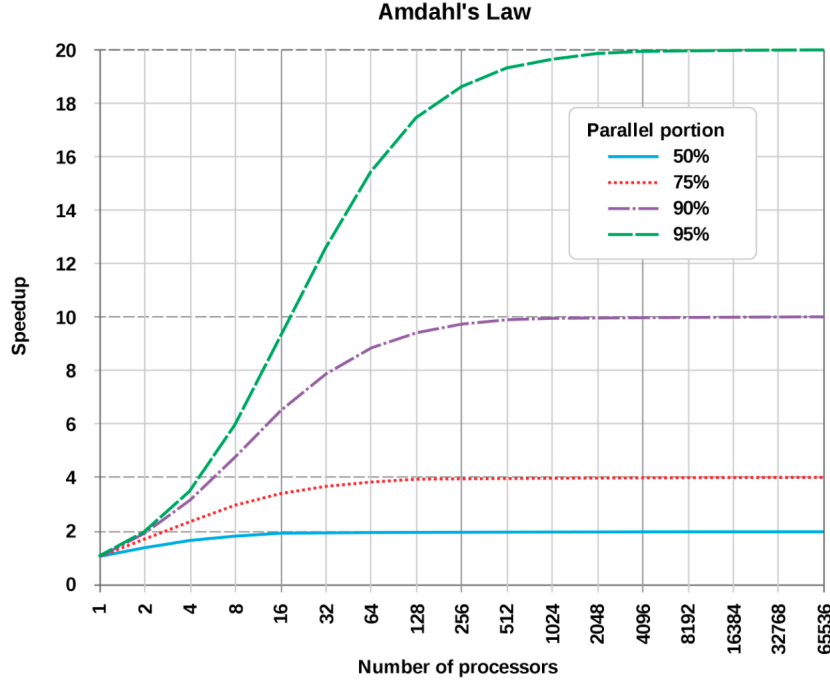


Figure 2.8: Speedup curves varying the number of processors/cores for different portions of parallel code in an application, using Amdahl's law ^a.

^aImage adopted from: https://en.wikipedia.org/wiki/Amdahl%27s_law

2.2.1 DVFS and Power Consumption

DVFS is a technique commonly used to save dynamic power by adjusting the operating voltage-frequency (V-f) of a processor regarding the workload and performance variations (Benini et al. 2000, Seo et al. 2012, Tang et al. 2017). It is a useful method for increasing an embedded system's energy efficiency by reducing the power dissipation (Usman et al. 2013). Power consumption in digital Complementary Metal Oxide Semi-conductor (CMOS) integrated circuit (IC) is the sum of the static power and the dynamic power represented by (Cardoso et al. 2017)

$$P = P_{\text{static}} + P_{\text{dynamic}}. \quad (2.2)$$

Short circuits and leakage currents are responsible for power consumption even when transistor devices are not switching (Cardoso et al. 2017). The static power consumption (P_{static}) can be calculated by (2.3), where V represents the supply voltage and I_{Leak} represents the overall current flowing through the device, which is given by the sum of the leakage currents (Chandrakasan & Brodersen 1995, Alonso et al. 2014, Kim et al. 2003). The static power depends mainly on the area of the IC and can be decreased by disconnecting some parts of the circuit from the supply voltage and/or by reducing the supply voltage.

$$P_{\text{static}} = V \times I_{\text{Leak}}. \quad (2.3)$$

Dynamic power (P_{dynamic}) consists of the charging and discharging the voltage saved in the circuit's node capacitances (Cardoso et al. 2017). It can be calculated by (2.4), where V represents the supply voltage, a represents the activity factor, C represents the load capacitance, and F denotes the clock frequency at which the device is operating (Chandrakasan & Brodersen 1995, Alonso et al. 2014, Kim et al. 2003).

$$P_{\text{dynamic}} = a \times C \times V^2 \times F \quad (2.4)$$

Device vendors provide a table of discrete values of supply voltages that the processing chip can operate beneath by maximum frequencies (Tang et al. 2017). For instance, we used the frequency-voltage information of ODROID-XU3 showed at Table 2.2 to obtain a refined power model discussed in Part III. Changing the CPU to a frequency-voltage pair is accomplished by sequentially stepping up or down through each adjacent pair (Cardoso et al. 2017). Thus, DVFS takes advantage of the quadratic relationship between the supply voltage and the dynamic power consumption (Burd & Brodersen 1995), which can result in significant energy savings. Typically, DVFS is controlled by software, e.g., Linux's power governors and ARM's Voltage and Frequency Scaling (VFS) firmware (Flynn 2012).

2.2.2 Linux Power Governors

`cpufreq` is the Linux kernel subsystem that allows the frequency to be explicitly set on processors and offers a modularized set of interfaces to manage the CPU frequency changes (Brodowski 2005). The subsystem provides frequency-changing policy governors (known as `cpufreq` governors), changing the CPU frequency based on different criteria, such as CPU usage (Pallipadi & Starikovskiy 2006). Moreover, most `cpufreq` governors only allow the CPU frequency to be set to predefined fixed values (Brodowski et al. 2010). Once a frequency is selected, its respective voltage operating value takes place. This mechanism is permitted, in the Linux kernel, through the Operating Performance Points or OPPs library (Menon 2009). OPPs allows the frequency-voltage pairs information being implemented by a set of discrete tuples.

The `cpufreq` governors can be divided into the trivial and the non-trivial groups. There are three trivial frequency governors (Brown 2016):

- **performance** governor sets the CPU statically to the highest frequency within a minimum and maximum frequencies borders.
- **powersave** governor sets the CPU statically to the lowest frequency within a minimum and maximum frequencies borders.
- **userspace** governor allows the user, or any userspace program running with UID 'root', to set the CPU to a specific frequency by making a `sysfs`⁵ file (`scaling_setspeed`) available in the CPU-device directory.

⁵The `sysfs` filesystem is a pseudo-filesystem which provides an interface to kernel data structures. (Kerrisk 2018)

The non-trivial governors change the CPU frequency based on the system load, which is defined by generic information known to the kernel rather than hardware-specific counters (Brown 2016). These governors use the CPU-usage statistics (visible in `/proc/stat`) to determine the average load, which is the ratio of CPU busy time to total time within a window of time. The non-trivial governors are (Brodowski & Golde 2010, Brodowski 2005):

- **ondemand** governor sets the CPU frequency depending on the current system load. It immediately jumps to the highest frequency and then possibly back off as the idle time increases, making it suitable for interactive tasks. The CPU must have the capability to switch the frequency very quickly.
- **conservative** governor scales up more gradually than the other non-trivial governors. It smoothly increases and decreases the CPU speed rather than jumping to max speed when there is any CPU load. This behaviour is more suitable in a battery-powered environment.
- **interactive** governor is designed for latency-sensitive, interactive workloads. In general, Its behaviours differs by raising CPU speeds to bring CPU load below a target.

2.2.3 Energy Management Approaches and Aspects

In this section, we describe some state-of-the-art solutions, which help outline the contributions of this thesis. We attempt to distinguish the main aspects that differentiate us from others concerning existing approaches for finding applications' configurations to improve energy efficiency. The most relevant works are characterised in Table 2.1.

Application Scenarios Goals

Research can be categorized concerning the application scenario focus, which can have the following characteristics: running concurrent applications, multi-threaded applications, imposing performance, power or energy requirements. Some works concentrate on concurrent execution of multiple applications (Aalsaud et al. 2016, Basireddy et al. 2020, Gupta et al. 2017, Kundan et al. 2021, Shamsa et al. 2021, Tzilis et al. 2019) and others exclusively on single-application scenarios (Al-hayanni et al. 2020, Endrei et al. 2018, Manumachu & Lastovetsky 2018, De Sensi et al. 2016). Tzilis et al. (Tzilis et al. 2019) and Kundan et al. (Kundan et al. 2021) propose energy management approaches based on performance and power estimations of applications choosing at runtime the most energy-efficient configuration. They do not consider multi-threaded versions, but instead, they allow multiple instances of the same application to run simultaneously. Indeed, energy-efficient management of concurrent applications is harder to accomplish due to dynamically changing situations. However, guarantee an energy consumption requirement with a minimum performance level of a single application affects the energy efficiency of a whole cluster-based

system. Thus, considering this aspect important, our methodology manages energy and performance requirements of individual applications as approached by other works (Endrei et al. 2018, Gupta et al. 2017, Manumachu & Lastovetsky 2018, Vasilakis et al. 2017). While some works are only concerned about the performance constraints (Basireddy et al. 2020, Loghin & Teo 2018, Shamsa et al. 2021, Tzilis et al. 2019). Furthermore, for parallel or multi-threaded applications, some works like ours extend Gustafson’s (Gustafson 1988) and Amdahl’s law (Amdahl 1967) in order to characterize the application (Al-hayanni et al. 2020, De Sensi 2016, Loghin & Teo 2018, Woo & Lee 2008). Others attempt to identify the phases of each application thread through monitoring performance counters (Basireddy et al. 2020, Gupta et al. 2017, Sawalha et al. 2011). As shown in Loghin et al. (Loghin & Teo 2018), the applications’ workloads with large sequential fractions present small energy savings regardless of the heterogeneous processing system. Therefore, it is vital to exploit the energy efficiency in HMP architectures and the parallelism of single multi-threaded applications.

Power-management Techniques

A notable number of power-management approaches target reducing power dissipation and a performance increase by combining three techniques. First, Dynamic Voltage and Frequency Scaling (DVFS) (Herbert & Marculescu 2007), which selects the optimal operating frequency. Second, Dynamic Power Management (DPM) (Benini et al. 2000), which turns off idle system components. Finally, application placement (thread allocation/mapping) (Singh et al. 2013), which determines the number of cores or the cluster type an application executes.

The process of mapping involves the binding of applications or threads to underlying hardware resources. For example, in a heterogeneous architecture with four high-performance larger and more power-hungry cores and four energy-efficient and slower cores (see Figure 2.9), allocating all threads to the high-performance cores will improve performance but consume more power. On the other hand, allocating all threads to low energy consumption cores can degrade application performance. It could also be possible to allocate a thread in one high-performance core to speed up a more complex task and the remaining threads pinned at slower cores (see Figure 2.9c). Also, using the DPM technique, unused cores may be turned off to optimise energy consumption. Finally, active cores can take advantage of DVFS to optimise even further performance and energy consumption. For example, a thread on the larger core could run at a specific frequency enough to speed up a sequential task without spending too much power, while the smaller cores run at the maximum frequencies.

The mapping problem is similar to the quadratic assignment problem, a well-known NP-hard problem (Garey & Johnson 1990). Besides, adding more two techniques as DPM and DVFS only makes finding an optimal solution that satisfies all given constraints extremely difficult and time-consuming. Thus, methods based on the application domain knowledge need to be employed to find a nearly optimal solution. The application mapping problem has been identified

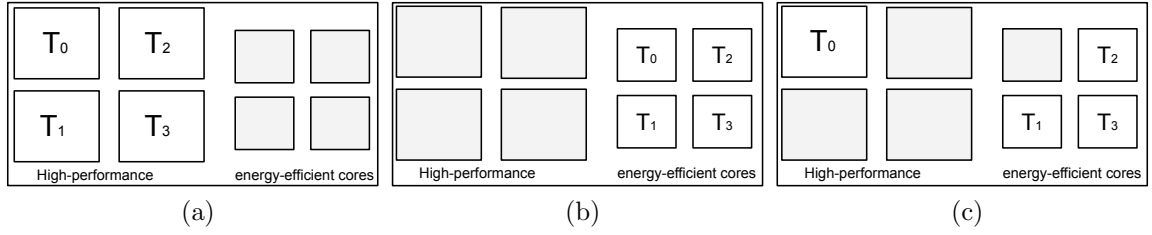


Figure 2.9: Thread mapping examples on a eight core heterogeneous architecture.

as one of the most urgent problem to be solved for implementing embedded systems (Marculescu et al. 2009, Marwedel et al. 2011). Some strategies decide the application placement without taking control of the DVFS aspect (Donyanavard et al. 2016, Sarma et al. 2015, Kim et al. 2014). While others only concern is finding the best cluster frequency performances within a power or energy buget (De Sensi 2016, Endrei et al. 2018, Manumachu & Lastovetsky 2018, Kundan et al. 2021). The works (Al-hayanni et al. 2020, Basireddy et al. 2020, Shamsa et al. 2021) focus on allocating and scaling system resources (DVFS) to minimize the energy consumption of the applications. As Tzilis et al. (Tzilis et al. 2019) and Aalsaud et al. (Aalsaud et al. 2016) approaches, we combine DVFS, DPM, and application placement by selecting the operating frequency-voltage pair, the type and number of active cores simultaneously. Besides, our target applications are multi-threaded, then we also do threads allocation by bidding threads accordingly to cores during our runtime approach.

Decision Guidance

Next, we describe the points that are essential to guide the configuration choice. Generally, we distinguish: (i) offline application profiling; (ii) runtime performance or power monitoring; (iii) predictions performed using a model.

Shamsa et al. (Shamsa et al. 2021) make offline profiling of the application’s power consumption and performance to generate a bias model. Then, a monitor checks for new applications to create a list of running applications with their information collected offline. Kundan et al. (Kundan et al. 2021) use a selection of performance monitoring counters (PMC) which are most correlated to IPS (Instruction Per Second) and power consumption to train a regression-based Multi-Layer Perceptron (MLP) artificial neural networks (NN) at the different supported frequencies of the cluster. At runtime, the selected PMC events are sampled and read by the set of the pre-trained MLP NNs to predict which frequency best satisfies the user-defined performance and power consumption constraints. Al-hayanni et al. (Al-hayanni et al. 2020) present a Parallelization-Aware Run-time Management (PARMA) approach which reads hardware performance counters to calculate the instantaneous degree of parallelism (DoP) value for an application running at a given homogeneous multi-core platform. Then, DoP is used to access an offline pre-calculated discrete table that provides optimal operating point (number of cores and

operating frequencies) for various values of DoP. Gupta et al. (Gupta et al. 2017) characterize the application by collecting power consumption, processing time and six performance counters. They use the data to train classifiers that map performance counters to optimal configurations. Then, at runtime, these classifiers and performance counters are used to select the optimal configuration concerning a specific metric, e.g. energy consumption. The major problem of this approach is the growth of the number of cores and cluster types, leading to this strategy losing reliability. Tzilis et al. (Tzilis et al. 2019) use a strategy that requires a total number of runs that is linear concerning the number of cores. They use a similar number of performance counters as Gupta et al. to profile the application. When the profiled application spawns, they match the online measurement to the closest profiled value and use it as a starting point to predict its performance in the current situation. De Sensi et al. (De Sensi et al. 2016) focus on a single application and monitor data to refine power consumption and throughput prediction models. They compared the predicted outcomes from their models against the data monitored in the current configuration. In case the prediction error is lower than a defined threshold, the calibration phase completes. The problem of online monitoring is the overhead incurred in refining models or making decisions. For this reason, as the works (Al-hayanni et al. 2020, Gupta et al. 2017, Kundan et al. 2021, Shamsa et al. 2021, Tzilis et al. 2019), we focus on an online optimization facilitated by offline profiling. Such approaches lead to better performance results than only online optimizations as they take advantage of both offline and online computations (Reddy, Basireddy Karunakar 2019).

A sort of performance and power modelling is commonly used to guide the configuration selection. Many works (Aalsaud et al. 2016, Al-hayanni et al. 2020, Basireddy et al. 2020, Kundan et al. 2021, Shamsa et al. 2021, De Sensi et al. 2016, Tzilis et al. 2019, Vasilakis et al. 2017) are based on models that rely upon runtime measurements retrieved from traces or instrumentation to collect performance counters, which is often not a trivial process. Usually, performance and energy events should be recorded in different runs to prevent counter multiplexing. Otherwise, it may cause application disturbance and decrease measurement accuracy (Endrei et al. 2017). As in other works (De Sensi 2016, Endrei et al. 2018, Manumachu & Lastovetsky 2018, Shamsa et al. 2021), our approach only relies on a minimum set of parameters, such as the number of cores of each cluster and the cluster frequency. Thus, our approach is simple to automate because it does not require any external instrumentation tools, and its utilisation across different architectures is less restricted. In contrast to De Sensi et al. (De Sensi et al. 2016) and Kundan et al. (Kundan et al. 2021) which use non-structured models to make predictions, analytical models often require only a few executions of the application to provide sufficiently good predictions, as shown in Furtunato et al. (Furtunato et al. 2020). As heterogeneous multi-processing systems have a vast design space, we consider this aspect of analytical models fundamental to estimate unseen configurations of a given architecture with reduced time and energy overheads.

Al-hayanni et al. (Al-hayanni et al. 2020) devised parallelization-aware power

models which allow the optimization of any weighted-product energy/performance trade-off metrics, such as Energy per Instruction (EPI) and Energy-Delay Product (EDP). Their works show the importance of considering the DoP to find optimal operation points for a given multi-core architecture. The only caveat is that it does not apply to heterogeneous architecture with multiple types of cores. Loghin and Teo (Loghin & Teo 2018) derived equations to evaluate the speedup and energy savings of modern shared-memory homogeneous and heterogeneous computer systems. They introduced two parameters: the inter-core speedup (ICS), describing the execution time relation among diverse types of cores in a heterogeneous system for a given workload; and the active power fraction (APF), representing the ratio between the average active power of one core and the idle power of the whole system. They validated their work against measurements on different types of heterogeneous and homogeneous modern multi-core systems. Their results show that energy savings are limited by the system’s APF, particularly on performance cores and by the workload’s sequential fraction when running on more efficient cores. We used a similar approach when devising our performance and power equations. Our performance model has a parameter (**perf**) that represents the median of the larger (big) cores speedup compared with the smaller (LITTLE) cores parameter — see Chapter 3. The value of this parameter can be determined independently of any target application, i.e., application-agnostic. For the offline-online approach presented in Chapter 5, we determine this parameter during the performance model fitting, i.e., application-dependent. Our power model describes the dynamic and the static power consumption of the active cores of each cluster type. The aim is to capture the properties of the hardware architecture so that we do not need to fit the power model for each application. For the offline methodology approach presented in Chapter 3, we use a few strategically sampled power measurements of each cluster. For the offline-online approach, we measure all components of a device, not only the CPU power. We consider this approach very important since it can generate more impact for battery-based devices. Ultimately, our energy model is a combination of a performance model and an application-agnostic power model. The models are used for predicting the optimal performance and energy consumption trade-off configurations of a parallel application for a given HMP system.

The main concern with using heuristics, models refinement or any other strategy to make decisions at runtime is the added overhead. Thus, the most significant contribution of our work is to avoid a full exploration of the search space to find the best configurations to execute an application at runtime. A similar goal, but with a different approach, has been pursued by the works (De Sensi 2016, Kundan et al. 2021, Shamsa et al. 2021). De Sensi (De Sensi 2016) uses a hardware architecture that is not HMP, and it offers only 312 possible configurations. Kundan et al. (Kundan et al. 2021) tuned the layers of the MLP to decrease execution time without inducing significant accuracy loss at a given heterogeneous architecture. However, only the operating frequency is considered, which considerably decreases the total number of configurations to explore. Shamsa et al. (Shamsa et al. 2021) considered DoP, DVFS and the core selection as knobs, creating vast search space of possible

configurations. Nevertheless, they demonstrated that a hybrid approach, relying on offline profiling and an online selection, can offer a negligible runtime overhead. Moreover, Endrei et. al (Endrei et al. 2018) and Manumachu et al. (Manumachu & Lastovetsky 2018) use Pareto Frontier as a strategy to minimize energy consumption without degrading application performance. Still, they do not work on ISA heterogeneous architecture as Gupta et al. (Gupta et al. 2017) and our approach.

Decision Strategy - Pareto Frontier

Using the Pareto frontier method, we can find optimal configurations since it represents the optimal trade-off between energy consumption and performance for the target system (Balaprakash et al. 2014). The Pareto configurations are the set of all optimal solutions from which it is not possible to improve a criterion (performance or energy) without degrading another. Its performance and energy consumption can characterize every configuration of the system. The Pareto frontier represents configurations in the current system that use the lowest energy and hence are the most efficient among all configurations that can serve a given load (Sen & Wood 2017). These configurations are Pareto optimal because, among these configurations, one cannot reduce energy without also reducing load or increase load without also increasing energy.

Users can take advantage of configurations at the Pareto frontier to guide development toward energy-optimal computing (Sen & Wood 2017). Also, these configurations can provide a power-constrained performance optimization to identify the optimized performance under a power budget (Bailey et al. 2015). Moreover, the Pareto-optimal fronts offer a trade-off zone that can be used to produce the optimal performance and energy efficiency prediction of a model (Endrei et al. 2018, Coutinho et al. 2019). We use Pareto-optimal front predictions to support runtime management that selects frequencies and cores on the fly according to the application's energy and performance requirements. The runtime selection algorithm is a simple local search (Nelder-Mead method) in which a Pareto configuration is provided as a starting point. As the method requires typically only one or two function evaluations per iteration (Singer & Nelder 2009), the runtime overhead for finding configurations is not very expensive or time-consuming.

In summary, we propose a method to find the optimal energy-performance trade-offs on a two-cluster heterogeneous architecture of a single multi-threaded application. The energy estimation is obtained by combining the offline performance and power models. The offline performance application characterization is straightforward; it does not use performance counters. The power model involves only architecture-specific parameters, so it is sufficient to fit only once for a specific platform. The individual applications' performance and power requirements can be achieved by the energy-performance trade-off zone provided by the Pareto frontier. Ultimately, the Pareto-optimal configurations are used as a starting point to a runtime local search algorithm.

Table 2.1: Related work categorized by various aspects.

	Scenarios					Techniques			Decision guided by			Decision Strategy	Arch.
Reference	Multi-threaded	Multi-apps	Power	Energy	Performance	DVFS	DPM	Placement	Monitoring	Profiling	Prediction	Pareto	Heterogeneous
(Kundan et al. 2021)	-	✓	✓	-	✓	✓	-	-	✓	✓	MLP NN	-	✓
(Shamsa et al. 2021)	✓	✓	-	-	✓	✓	-	✓	✓	✓	Bias Model	-	✓
(Al-hayanni et al. 2020)	✓	-	-	-	-	✓	-	✓	✓	✓	Analytical	-	-
(Basireddy et al. 2020)	✓	✓	-	-	✓	✓	-	✓	✓	-	Analytical	-	✓
(Endrei et al. 2018)	✓	-	-	✓	✓	✓	-	-	-	-	Analytical	✓	-
(De Sensi et al. 2016)	✓	-	✓	-	✓	✓	-	✓	✓	-	ML	-	-
(Gupta et al. 2017)	✓	✓	-	✓	✓	✓	✓	-	✓	✓	-	✓	✓
(Manumachu & Lastovetsky 2018)	✓	-	-	✓	✓	✓	-	-	-	-	Analytical	✓	-
(Tzilis et al. 2019)	-	✓	-	-	✓	✓	✓	✓	✓	✓	Analytical	-	✓
(Loghin & Teo 2018)	✓	-	-	-	✓	-	-	-	-	-	Analytical	-	✓
(Vasilakis et al. 2017)	✓	-	-	✓	✓	✓	-	✓	-	-	Analytical	-	✓
(De Sensi 2016)	✓	-	✓	-	✓	✓	-	-	-	-	Analytical	-	-
(Aalsaud et al. 2016)	-	✓	✓	-	-	✓	✓	✓	-	✓	-	-	✓
Proposed Thesis	✓	-	-	✓	✓	✓	✓	✓	✓	✓	Analytical	✓	✓

2.3 Experimental Setup and Applications

This section presents the details of hardware platforms and applications used to conduct experiments in the following chapters.

2.3.1 Odroid-XU3 Platform

The main experimental platform considered in this work is the ODROID-XU3 board developed by Hardkernel co., Ltd (Hardkernel 2014). Figure 2.10 shows the ODROID-XU3 board which uses a Samsung Exynos5422 System on a Chip (SoC) (Samsung 2014). It utilises ARM big.LITTLE technology with two ARM 32-bit instruction clusters—Cortex-A15 (big) quad-core cluster and a Cortex-A7 (LITTLE) quad-core cluster, united with a CCI. In addition, the chip contains a Mali-T628 GPU and 2 GB DRAM LPDDR3. Each cluster offers different trade-offs between power and performance. The ARM Cortex-A15 is a high-performance processor with 2 MiB L2-Cache whereas ARM Cortex-A7 CPU is an extremely energy-efficient processor with 512 kiB L2-cache.

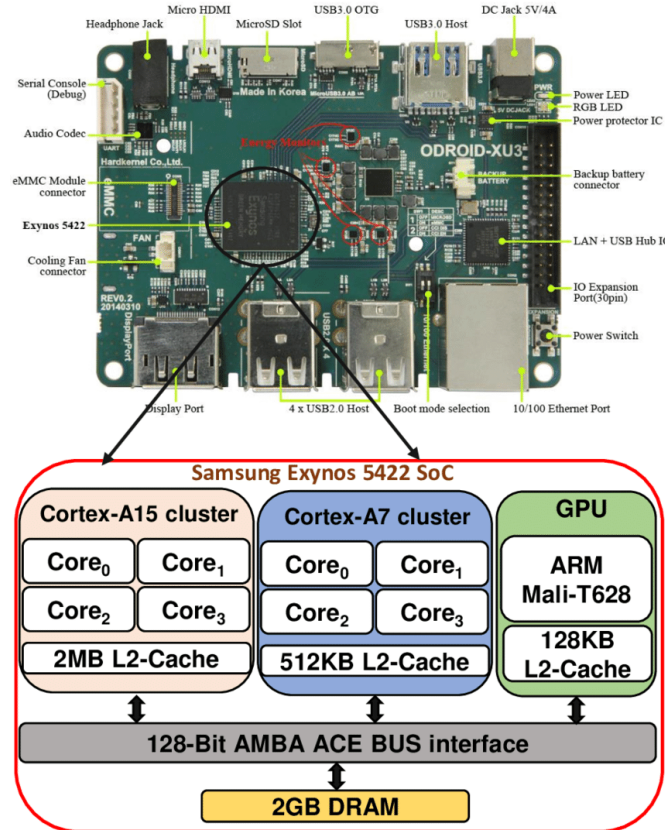


Figure 2.10: ODROID-XU3 board highlighting Samsung Exynos5422 chip internal organization (Reddy, Basireddy Karunakar 2019).

The board has four current and voltage sensors to measure the power dissipation

of the cortex-A15 cores, cortex-A7 cores, GPUs and DRAMs individually. Hardkernel provides different Linux flavours and kernel versions to run on the board. Despite a limitation with Kernel version above 4.0 with the experimental board (see Section 3.3), any kernel version should support cores disabling. The platform also has DVFS to optimise system operation in terms of performance and energy consumption. The voltage and frequency are adjusted on the fly depending on the actual CPU's needs to conserve power and reduce the chip's heat. In the Samsung Exynos, the supply voltage and clock frequency (v-f) are tailored for each cluster within its respective power domain. For the Part II, we devise a power model approximating the v-f relationship linearly (see section). For the Part III, we use the Table 2.2 provided by the Hardkernel Linux Kernel version to devise a new power modelling considering the v-f pairs of the Exynos5422 (see Section 5.2).

Table 2.2: Voltage and Frequency relation for Exynos5422

Cortex-A15 Cluster		Cortex-A7 Cluster	
Frequency (MHz)	Voltage (V)	Frequency (MHz)	Voltage (V)
200	0.9	200	0.9
300	0.9	300	0.9
400	0.9	400	1
500	0.9	500	1
600	0.9	600	1
700	0.9	700	1
800	0.925	800	1.1
900	0.95	900	1.1
1000	0.975	1000	1.1
1100	1	1100	1.25
1200	1.05	1200	1.25
1300	1.05	1300	1.25
1400	1.0625	1400	1.25
1500	1.0875	1500	1.25
1600	1.125		
1700	1.1625		
1800	1.2		
1900	1.25		
2000	1.3125		

It has 19 available frequency levels ranging from 200 MHz up to 2 GHz on Cortex-A15 and 14 available frequency levels from 200 MHz to 1.5 GHz on Cortex-A7. Therefore, considering that each configuration is a combination of the number of cores in each cluster and a frequency pair that is available as a resource for an application to be executed, there are 6384 possible configurations to explore. To further clarify, consider that each cluster can select none to four cores, then $(\# \text{ of big cores}) \times (5 \times 19) \times (\# \text{ of LITTLE cores}) \times (5 \times 14) = 6650$. For each

combination of active numbers of LITTLE and big cores, there are 19×14 arrangements of frequencies to choose. Since the combination of 0 big cores and 0 LITTLE cores is not possible, the number of available configurations is $6650 - (19 \times 14) = 6384$. The device firmware automatically regulates the voltage for a selected frequency. As this platform counts on multiple heterogeneous processing elements and accurate CPU power measurement sensors, it makes an excellent device to validate our methodology for the Part II. In the Part III, we are looking for performance and the whole board's energy consumption, so we use the ODROID-XU3 with a power meter – see next section.

2.3.2 Odroid SmartPower2

Odroid SmartPower2 is a power-supply capable of retrieving power data (current, voltage, power, watt-hour) via telnet or USB cable. Figure 2.11 shows the components of the electronic board. Also, Figure 2.12 shows the experimental setup for Part III. ODROID-XU3 is connected to the SmartPower2 electronic board which can provide up to 5.3V/5A. For the work developed to Part II, we use the USB connection to measure the power consumed of the whole ODROID-XU3. We modified the SmartPower2 firmware to be able to sample reading every 250 milliseconds.

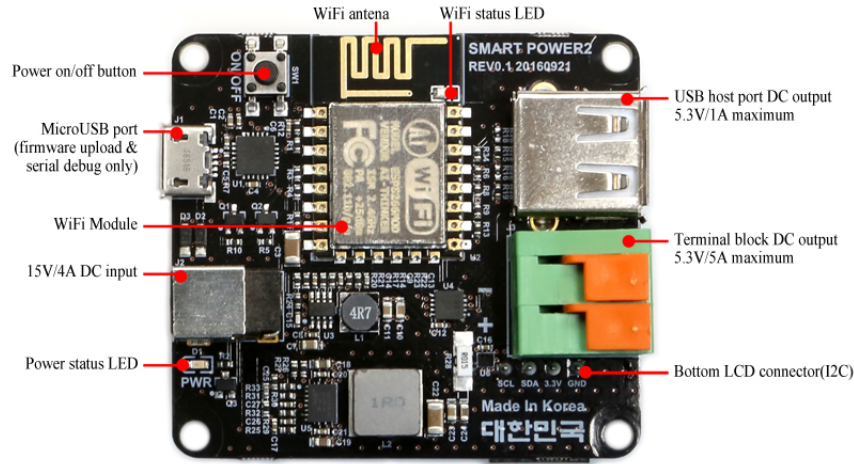


Figure 2.11: Odroid SmartPower2 board detail (Hardkernel 2017).

2.3.3 Benchmark Applications

We used eight applications to run experiments in this thesis—all eight used in Part II and two of them in Part III. The Black-Scholes, Bodytrack, Freemine applications provided by PARSEC, Smallpt and x264 by Phoronix Test Suite and kmeans, Particle Filter and LavaMD from Rodinia benchmark. PARSEC (Bienia 2011) is a well-known benchmark suite of parallel applications, which focuses on emerging workloads and is designed to be representative of next-generation shared-memory programs for chip multi-processors. The Phoronix Test Suite (Suite

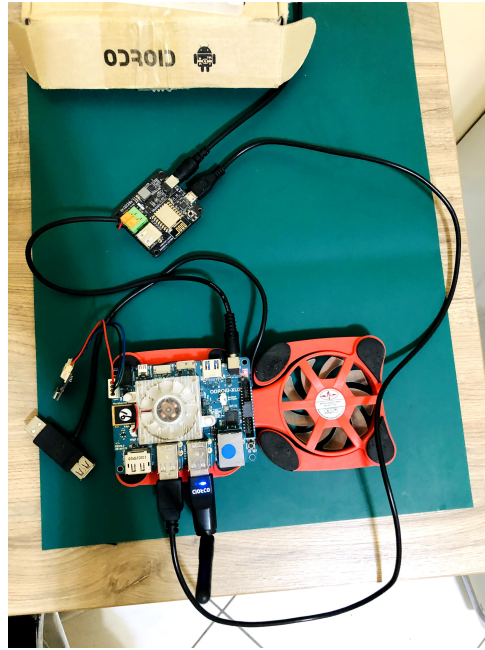


Figure 2.12: ODROID-XU3 connected to Odroid SmartPower2.

2021) is a benchmarking platform that provides an extensible framework for carrying out tests in a fully automated manner from test installation to execution and reporting. Rodinia benchmark (Che et al. 2009) is a set of applications designed for heterogeneous computing infrastructures with OpenMP (OpenMP 2021), OpenCL (Team 2021), and CUDA (NVIDIA 2016) implementations. We have chosen the applications because they were implemented using OpenMP. It allowed us to modify their workloads to be dynamically balanced using dynamic OpenMP schedule clause (Kale 2018). Thus, enabling the applications to comply with our statistical models (See Section 3.2).

The notable exception for the chosen criteria is the `x264` encoder provided by the Phoronix test suite since it has only POSIX threads for parallelism. As higher video quality significantly depends on the encoding process, i.e., longer execution time and higher energy consumption, it is essential to improve the energy efficiency of video encoding on embedded devices. We therefore evaluate the behaviour of our methodology on this video encoder.

The characteristics of the applications which are used in this paper, according to (Özisyilmaz et al. 2006, Che et al. 2009, Che et al. 2010, Bienia 2011, Goodrum et al. 2012), can be seen in Table 2.3. Each application is classified as CPU intensive and memory intensive. The former means that the application possesses tasks that demand many CPU cycles, so the speed is often limited by how fast the processor can compute the data. The latter means that the application is speed limited by how fast the memory can feed data to the processor. Most applications are CPU and memory intensive, so we expect a considerable use of the HMP chip and the memory system as well as the communication between them.

Our power and performance model does not explicitly include implicit hardware or software features, such as communication-to-computation ratio, cache sharing and off-chip traffic. However, as those features should limit the parallel speedup, it is expected that the sequential fraction of the application may be more significant for some applications. Ultimately, we aim at demonstrating that a low-overhead, straightforward model is adequate to estimate a Pareto frontier.

Table 2.3: Characteristic details of each application based on

Benchmark	App.	Domain	Type	Problem Size	Description
PARSEC	Black-Scholes	Financial Analysis	CPU intensive	10,000,000 options	Portfolio price calculation using Black-Scholes PDE
	Bodytrack	Computer Vision	CPU and memory intensive	4 cameras, 261 frames, 4000 particles, 5 annealing layers	Computer vision, tracks 3D pose of human body
	Freqmine	Data Mining	CPU and memory intensive	Database composed of spidered collection of 250,000 web HTML documents	Array-based version of the FP-growth method
Rodinia	Kmeans	Data Mining	CPU and memory intensive	1000,000 points, 34 dimensions, 5 clusters	Mean-based data partitioning method
	Particle Filter	Medical Imaging	CPU and memory intensive	Video resolution 128×128 , 10 frames, 10,000 particles	Tracks cells by statisticaly estimating the path in a Bayesian framework.
	LavaMD	Molecular Dynamics	CPU and Memory intensive	100 boxes	Computes the interactions between particles in a three-dimensional cubic space
Phoronix	Smallppt	Image Renderer	CPU intensive	1024×768 , 128 samples/ pixel	Generates a single image of a modified Cornell box rendered with full global illumination
	x264	Media Processing	CPU and memory intensive	1920×1080 pixels (HDTV resolution), 600 frames	H.264 video encoder

Part II

Finding Optimal Performance and Energy Trade-offs Methodology

In this part of the thesis, we describe our first and most important contribution, the methodology for finding performance and energy trade-offs for single parallel applications running on single-ISA HMP systems. In Chapter 3, we introduce the workflow of the methodology describing the performance, power and energy modelling to estimate an application's performance and energy consumption according to the whole configuration space of a given HMP architecture. This methodology is evaluated in Chapter 4, by employing it over eight different applications on real hardware. The results of each modelling parameter and validation of the Pareto-optimal configurations are shown for each application. Also, to show the effectiveness of our technique, we compare the application's performance and energy consumption under three default DVFS governors and the Pareto configurations.

Chapter 3

Finding Optimal Performance and Energy Trade-off

In this chapter, we describe the proposed methodology, which estimates the Pareto-optimal configurations for HMP systems. A configuration is defined as a point in the vast parametric space that defines parallel software operation, including the number of processing cores used in each cluster and the operating frequency of those cores. Also, we assume that the clusters have their own voltage and frequency domain, which is shared among the cores of the cluster. Thus, design attributes such as the number of clusters and the number of cores and frequency levels of each cluster are known parameters. As a result, the model matches a large subset of existing HMP systems, to which the proposed methodology can be applied.

Next section describes each step of the methodology. We devise the performance, power and energy consumption models for a given application and HMP architecture, in the Sections 3.2, 3.3, 3.4, respectively.

3.1 General Overview

There are five steps that compose this method: sampling of configurations, runtime measurements, fitting of models, model validation, and the Pareto frontier selection. Figure 3.1 presents an overview of these steps.

In the first step—sampling of configurations, we need to carefully choose the hardware configurations that is used to obtain the required data to fit the parameters of the models. To obtain parameters that generate higher accuracy, data measurements should be as diverse as possible, i.e., two or more measurement points should not have a similar number of cores and operating frequency, since those points most likely not add significant information to the modeling. For this reason, we used a Halton low discrepancy number generator to choose configurations that are not too similar to each other. Morokoff et al. (Morokoff & Caflisch 1994) describe how the low discrepancy number generator ensures that it covers the domain more evenly compared to pseudo-random number generators. Also, De Sensi (De Sensi 2016) shows that by picking equidistributed points, it is possible to achieve higher accuracy compared to pseudo-random selection of the points.

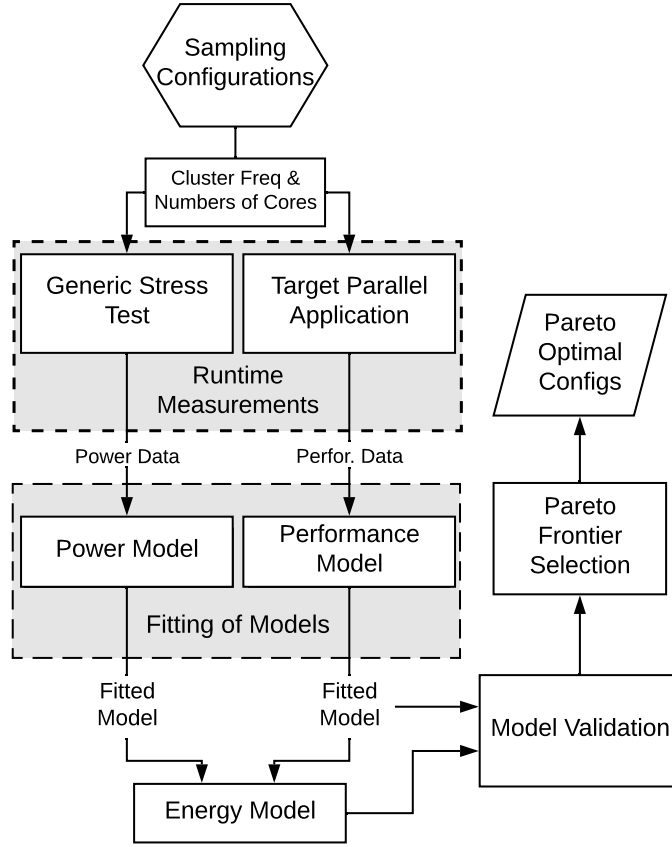


Figure 3.1: Flowchart of the methodology applied for predicting the optimal performance and energy efficiency trade-offs of a parallel application.

In the second step—runtime measurements, we evaluate the sampled configurations at runtime, measuring performance and power data to fit the models. First, for each configuration, we execute a generic stress-test to stress the processor and capture power consumption. Second, we execute the target parallel application to measure execution time for another subset of sampled configurations using the Halton number generator.

In the third step—fitting of models, we fit the measured data to the proposed architecture- and application-specific performance model and the architecture-specific, application-agnostic power model. The performance model intends to estimate the execution time of a parallel application for any point in the configuration space. We adjust the parameters of the power and performance models separately. The goal of the proposed power model is not to be accurate per se, but to capture the trend changes in power when a change of the operating frequency and number of active cores occurs, independent of the application running. The power consumption prediction accuracy is not strictly crucial if there are no power restrictions to apply on system budget. More importantly, we aim for decreasing energy consumption in HMP since it has more impact on batteries, which is essential on embedded devices. Moreover, being application-agnostic avoids the need for power measurements for

every application while preserving the architecture’s signature. Thus, we do not need to fit the power model for each application. Besides, the process of acquiring these measurements also increases the difficulty of automating these models as practical tools. For each model, we use a non-linear regression¹ minimizing the Root Mean Square Error, given by

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^n (A_i - E_i)^2}{n}}, \quad (3.1)$$

where A_i is the i th measured data of E_i , E_i is the i th estimated value given by the model, and n is the total number of configurations. Sections 3.2 and 3.3 detail the proposed performance and power consumption models, respectively. The proposed energy model is a combination of the power consumption model—previously fitted to a specific HMP system, and the performance model—fitted to a given parallel application and the same HMP system. Section 3.4 presents the proposed energy model.

In the fourth step—model validation, we evaluate the performance and energy consumption models for every point in the vast configuration space that defines parallel software operation.

Finally, in the fifth step—Pareto frontier selection, we select all modeled configurations whose models’ outcomes lay on the Pareto frontier, i.e., those configurations that yield the optimal performance and energy consumption trade-offs.

3.2 Application Performance Modelling

In this section, we devise a performance model for an HMP platform with two processing clusters. We assume that a parallel application coherently runs in b big cores and L LITTLE cores. Moreover, we do not consider or require different frequencies for each core in a cluster, i.e., all the cores in a cluster run at the same frequency. Inspired by other works (Hill & Marty 2008, Barros et al. 2015, Morad et al. 2006), we devised the following model that can be used to estimate the performance of a given parallel application running on a two-cluster HMP.

$$T_{\text{HMP}}(F, F_b, F_L, b, L) = T_L(F) \left(\frac{(1-f)F}{\text{perf} \cdot F_b} + \frac{fF}{b \cdot \text{perf} \cdot F_b + LF_L} \right), \text{ if } b > 0, \quad (3.2)$$

where T_{HMP} is the performance target, that is the total execution time goal for a given application. $T_L(F)$ is the application’s execution time running on a single LITTLE core with operating frequency F , which is not necessarily the same frequency of F_L . The number of active big and LITTLE cores in the processor is denoted by b and L , respectively. F_b and F_L are the operating frequencies of the big and LITTLE cores,

¹We used Differential Evolution, and Particle Swarm Optimization (DEPS) found at the solver <https://wiki.openoffice.org/wiki/NLPSolver>. Both are especially suited for numerical problems, such as non-linear optimization, and complement each other in that they even out their weaknesses.

respectively. **perf** is the performance improvement when moving computation from a LITTLE to a big core, independent of an application. Note that, **perf** depends on the hardware design, so we assume it has a fixed value representing how fast a big core is when compared to any LITTLE core. Thus, we do not need to fit this performance speedup parameter for each application.

The value of f represents the parallel fraction of the application. It is the only parameter that characterizes the application. Indeed, by modelling complex parallel applications using their parallel fraction only, we are neglecting the effects that a frequency change has on the performance of the memory hierarchy, on the parallel overhead, and the distribution of load across the heterogeneous cores (Furtunato et al. 2020). As those features should limit the parallel speedup, it is expected that the sequential fraction includes them. Most importantly, as shown in Section 2.1.4, f plays a vital role at the PET elements, increasing the chance of tangible gains for higher f in the multi-core architectures. Besides, It is our priority to keep our model straightforward to achieve low runtime overheads. We observed that, despite the many simplifications, the model still provides reasonably accurate energy consumption estimations.

We consider that the sequential fraction of the application is accelerated by using one of the big cores when such core is available. In this approach, the sequential portion expects to include the parallel overhead, i.e., communication or synchronization among the threads, which does not accelerate by making the big core run faster—as it would do for actual sequential code. Otherwise, we can extend Equation (3.2) as follows:

$$T_{\text{HMP}}(F, F_b, F_L, b, L) = T_L(F) \left(\frac{(1-f)F}{F_L} + \frac{fF}{LF_L} \right), \text{ if } b = 0. \quad (3.3)$$

In Equation (3.3), when there is no big core available or active, the application would not take advantage of the performance improvements from the big cores, therefore the parameter **perf** is removed. Also, the big core operating frequency is not used by the parallel application.

This performance model assumes that the modelled parallel workload is dynamically distributed and balanced over the running threads. If that is not the case for the application, cores may run out of work and become idle, while others might become overloaded and delay the end of the program’s execution. Thus, the performance model may then make incorrect estimations as idle cores still consume power. This can lead to improper power predictions, resulting in inadequate energy predictions. Although this may be seen as a limitation of the proposed approach, a well-balanced workload is necessary to increase the performance and scalability of parallel applications. Eventually, the advantages of performance gain and energy reduction presented in this work argue for pursuing this type of workload balancing scheme.

3.3 Power Consumption Modelling

In this section, we devise a power model for an HMP platform with two clusters. Assuming that the transistors used on both types of cores have the same power consumption behaviour with respect to their operating frequency and voltages, we can model the power consumption of the big and LITTLE cores, as follows:

$$P_{\text{HMP}}(F_b, F_L, b, L) = P_b(F_b, b) + P_L(F_L, L), \quad (3.4)$$

where $P_b(F_b, b)$ is the power consumption of b big cores running at frequency F_b , $P_L(F_L, L)$ is the power consumption of the L LITTLE cores running at frequency F_L .

Consider the equation that defines the maximum operating frequency for a transistor (Suleiman et al. 2005, Venkatachalam & Franz 2005):

$$F_{\text{Max}} = \kappa_1 \frac{(V - V_{\text{Th}})^h}{V}, \quad (3.5)$$

where V_{Th} denotes the threshold voltage, κ_1 is a constant, and h is a technology-dependent value often assumed to be within the range $[2, 3]$. Since performance is proportional to the operating frequency, digital circuits often operate with voltage and frequency pairs that push frequency closer to the maximum. If maximum performance is not required, the voltage is scaled down with frequency to maintain this policy (Cardoso et al. 2017). Generally, the supply voltage V can be perceived as a linear function of the frequency depicted such as follows:

$$V = \kappa_2 F, \quad (3.6)$$

where κ_2 is any constant. Some authors make a similar approach (Cardoso et al. 2017, Usman et al. 2013) showing that it is reasonable and common in the literature to make such approximation. Furthermore, we reiterate that we intend to capture the trend of how a change in the operating frequency-voltage would affect the power consumption of the system. For this work, our target is to save energy consumption and not accuracy in power consumption prediction.

Considering the Equations (2.2), (2.3) and (2.4) which represent the power consumption of a CMOS IC, we can rewrite as follows:

$$P(F) = aCV^2F + VI_{\text{Leak}}, \quad (3.7)$$

where a is the average activity factor, C is the load capacitance, V is the supply voltage, F is the operating frequency, and I_{Leak} is the average leakage current. Also, Using Equation (3.6), we can rewrite Equation (3.7) as follows:

$$\begin{aligned} P(F) &= aC(\kappa_2 F)^2 F + \kappa_2 F I_{\text{Leak}}, \\ &= aC\kappa_2^2 F^3 + \kappa_2 F I_{\text{Leak}}. \end{aligned} \quad (3.8)$$

Considering I_{Leak} and a to be approximately constant, we can approximate Equation (3.8) by

$$P(F) = \alpha F^3 + \beta F, \quad (3.9)$$

where $\alpha = aC\kappa_2^2$ and $\beta = \kappa_2 I_{\text{Leak}}$ are considered constants that mainly abstract semiconductor technology attributes. These attributes should be application-independent and fixed since they rely on hardware design. Then, the power of each cluster type is modelled as follows:

$$P_b(F_b, b) = b\alpha_b F_b^3 + C_b\beta_b F_b, \quad (3.10)$$

$$P_L(F_L, L) = L\alpha_L F_L^3 + C_L\beta_L F_L, \quad (3.11)$$

where b and L are the numbers of active big and LITTLE cores, respectively, accounted for the dynamic power modelling component. The technical setup of our experimental platform did not permit disabling individual core for this work. Therefore, we consider a single term of leak current for each cluster. Then, C_b and C_L represent the total number of cores of each cluster type and account for the static power modelling component.

Note that this power model does not distinguish one application from another since the activity factor a is assumed to be constant. This is a very generic assumption, which may cause an inaccurate power consumption estimation. Nevertheless, for this approach, more critical than accurate estimations of power is to capture how a change of the operating frequency-voltage would affect power consumption. It is expected that the way in which this change affects the power is the same, regardless of the activity factor. In other words, as long as the activity factor of the target application does not change too much from one frequency configuration to another, this assumption does not adversely impact on the proposed power model.

By combining Equations (3.4), (3.10) and (3.11), we come to the power model for the whole HMP chip parameterized by the two frequencies of each cluster as follows:

$$P_{\text{HMP}}(F_b, F_L, b, L) = P_{\text{HMP}_P} = b\alpha_b F_b^3 + C_b\beta_b F_b + L\alpha_L F_L^3 + C_L\beta_L F_L. \quad (3.12)$$

Equation (3.12) represents the power consumption of a parallel fraction (P_{HMP_P}) of a parallel application. Also, we need to predict the power of the sequential part of a code that runs in one core.

$$P_{\text{HMP}_S}(F_b, F_L) = \begin{cases} \alpha_b F_b^3 + C_b\beta_b F_b + C_L\beta_L F_L, & \text{if } b > 0, \\ C_b\beta_b F_b + \alpha_L F_L^3 + C_L\beta_L F_L, & \text{otherwise.} \end{cases} \quad (3.13)$$

When there is, at least, one big core available, we consider the dynamic power of one big core and the static power of all cores. In the case that just LITTLE cores are available, then the sequential part runs on one LITTLE core. In this case, the power consumption of the sequential portion contains the static power of all cores and the dynamic component of one LITTLE core.

3.4 Energy Modelling

In this section, we devise an energy model for a parallel application running on a two-cluster HMP platform by aggregating the sequential and parallel energy consumption as follows:

$$E_{\text{HMP}}(F, F_b, F_L, b, L) = E_{\text{HMP}_S} + E_{\text{HMP}_P}. \quad (3.14)$$

By combining the parallel part of Equation (3.2) and the power model of the whole chip in Equation (3.12), we have the parallel portion energy model component:

$$E_{\text{HMP}_P} = \frac{T_L(F) f F (L\alpha_L F_L^3 + C_L\beta_L F_L + b\alpha_b F_b^3 + C_b\beta_b F_b)}{b \cdot \text{perf} \cdot F_b + L F_L}. \quad (3.15)$$

Using Equation (3.13) and the sequential part of the Equations (3.2) and (3.3), we have the sequential portion energy model component, which relies on the number of active big cores available:

$$E_{\text{HMP}_S} = \begin{cases} \frac{T_L(F)(1-f)F(C_L\beta_L F_L + \alpha_b F_b^3 + C_b\beta_b F_b)}{\text{perf} \cdot F_b}, & \text{if } b > 0, \\ \frac{T_L(F)(1-f)F(C_b\beta_b F_b + \alpha_L F_L^3 + C_L\beta_L F_L)}{F_L}, & \text{otherwise.} \end{cases} \quad (3.16)$$

Finally, combining Equations (3.15) and (3.16), we come to the consolidated energy model for the whole HMP chip parameterized by the frequencies and the number of active cores in each cluster:

$$E_{\text{HMP}}(F, F_b, F_L, b, L) = T_L(F)F \times \begin{cases} \frac{(1-f)(C_L\beta_L F_L + \alpha_b F_b^3 + C_b\beta_b F_b)}{\text{perf} \cdot F_b} + \frac{f(L\alpha_L F_L^3 + C_L\beta_L F_L + b\alpha_b F_b^3 + C_b\beta_b F_b)}{b \cdot \text{perf} \cdot F_b + L F_L}, & \text{if } b > 0, \\ \frac{(1-f)(C_b\beta_b F_b + \alpha_L F_L^3 + C_L\beta_L F_L)}{F_L} + \frac{f(L\alpha_L F_L^3 + C_L\beta_L F_L + C_b\beta_b F_b)}{L F_L}, & \text{otherwise.} \end{cases} \quad (3.17)$$

We can now estimate energy consumption and the performance of any configuration for a given parallel application using our energy model in Equation (3.17) and performance models in Equations (3.2) and (3.3), respectively. Considering fitted models, we can estimate all possible performance and energy consumptions of an application. Then, these estimations are used in the Pareto method to obtain the most optimal performance-energy trade-offs. In the next section, we show the values obtained after the fitting process and the Pareto approach validation on a two-cluster heterogeneous board.

Chapter 4

Evaluation and Results

This Chapter evaluates the models and validate the method to find the Pareto Frontier. In Section 4.1, we show the hardware design parameters and the values of the parallel fractions obtained through the non-linear regression in Section 4.2. We validate our proposed methodology concerning the estimated Pareto frontier against direct measurements in Section 4.3. Moreover, we compute the error of the estimations by using the Mean Absolute Percentage Error (MAPE), defined as:

$$\text{MAPE} = \frac{1}{n} \sum_{t=1}^n \left| \frac{A_t - E_t}{A_t} \right|, \quad (4.1)$$

where n is the number of configurations, A_t is the actual measurement (execution time or energy consumption) and E_t is the estimated execution time or energy consumption.

We used the ODROID-XU3 (See Section 2.3.1) to run our experiments. The experimental platform was set up with a Ubuntu Minimal 18.04 LTS running Linux Kernel LTS 4.14.43, and it did not permit disabling individual cores. Also, we sampled the clusters' power consumption readings every 0.5 seconds with timestamps. Then, by integrating the power samples, we computed the energy used by a given application. We consider the average power calculated by dividing total energy by execution time. Moreover, we used *cpuset* (Derr 2004) subsystem to confine all the data collection instrumentation and operating system's processes/threads to the not active cores, when available, and assign the workload application appropriately to the cores. Thus, this mitigates the interference of all non-target application in the experiment.

In addition, we compare the Pareto configurations that give the least energy consumption and the highest performance against **performance**, **ondemand**, and **powersave** Linux governors for all applications (Section 4.4).

4.1 Hardware Design Parameters

This section shows the values obtained for the parameters that represent the hardware design characteristics. As these parameters are related to the CPU design

and they must be application-agnostic, we used a CPU stress tool, called **stress-ng** (King 2014). This tool can load and stress a computer system in various selectable ways and is available in any Linux distribution. This tool has an extensive range of specific stress tests that exercise floating-point, integer, bit manipulation, and control flow operations on the CPU.

We tested each method to find the one that maximizes the stress on the ODROID-XU3, i.e., the method that led to the highest power draw as cores executed their instructions (see Figure 4.1). Hence, after executing every method for 15 s on two big cores (1.9 GHz), and four LITTLE cores (1.5 GHz), we chose the **callfunc** stress method. Using all four big cores at the highest frequency would have overheated the board. High CPU frequencies, especially from the big cores, produce a large amount of heat, making the system protection mechanism against high temperatures halt the device to prevent damage. Moreover, running 15 seconds provides 30 power reading samples from the sensors. The average standard deviations regarding all methods are 0.0792 and 0.2395 for LITTLE and big power consumption, respectively.

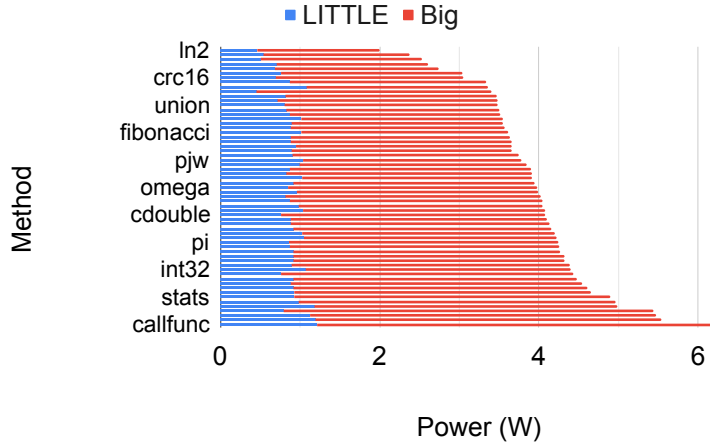


Figure 4.1: Power consumption from LITTLE and big clusters for every **stress-ng** method.

4.1.1 The Perf Parameter

The **perf** performance parameter represents the big core speedup in comparison with the LITTLE core. It is application-agnostic, relying only on the chip design. We executed the **stress-ng** with **callfunc** method on one core of each cluster type, fixing the number of operations at 87,555. This number of operations provides approximately one second to conclude in one big core with the highest frequency and up to 15 s for one LITTLE core with the lowest frequency.

Figure 4.2 shows the result of **perf** fitting. The left-hand side Y-axis shows the median execution time of five runs for one core of each cluster type and different frequencies. The frequency range used is 200 MHz–1.5 GHz. Higher values are not possible due to the maximum LITTLE core frequency. The right-hand side Y-axis

represents the speedup of one big core when comparing to one LITTLE core for each frequency. The average standard deviations of the runs regarding all frequencies are 0.0252 and 0.0404 for LITTLE and big core, respectively. The median of every speedup is 1.897 which is the obtained **perf** value.

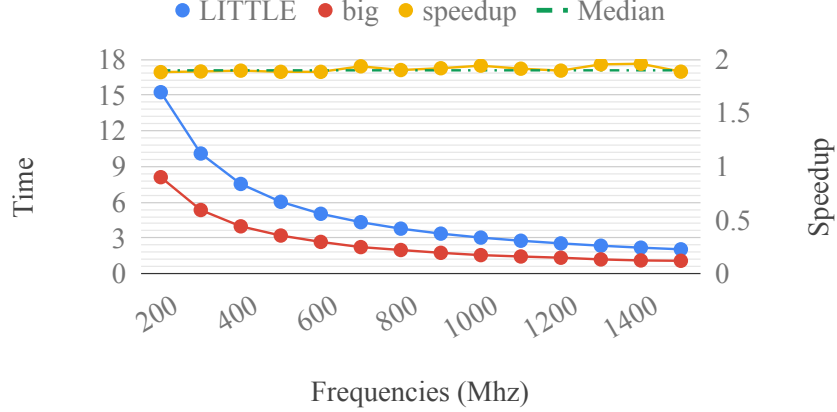


Figure 4.2: Performance parameter **perf** fitted for the architecture using a fixed number of operations of the **stress-ng** tool for one core of each cluster.

4.1.2 Power Consumption Parameters

We aim to obtain a power model that independent of the application, represents the power consumption of the two-clusters. As we want to model the total power consumption of the entire chip, we measured the whole chip to obtain P_{HMP} . We defined the number of processes and which cores would be active to execute the **stress-ng** by generating 95 evenly distributed Halton configurations. Each configuration is composed of the number of active big cores b , active LITTLE cores L , big's frequency cluster F_b and LITTLE's frequency cluster F_L .

We used the **stress-ng** with **callfunc** stress method and a fixed execution time of 75 seconds, with the command line **stress-ng -cpu N -cpu-method callfunc -t 75 -taskset t**. The **-cpu** parameter determines the number of processes on which the same method is executed, and the parameter **-taskset** can be used to set a process's CPU affinity.

Running 75 seconds provides 150 power samples. The average standard deviations regarding all 95 configurations are 0.0164 and 0.1478 for LITTLE and big core power consumption, respectively. As the configuration parameters and their total power consumption measurements are known, we fit the parameters $\alpha_L, \beta_L, \alpha_b, \beta_b$ from the Equation (3.12). The fitted values are: $\alpha_L = 5.953 \times 10^{-29}$, $\alpha_b = 2.914 \times 10^{-28}$, $\beta_L = 1.033 \times 10^{-10}$, $\beta_b = 9.342 \times 10^{-11}$.

4.2 Applications

We used all eight applications described in Section 2.3.3. We strive to capture the nuances of each configuration that affects the way the application works on the hardware, producing different execution time. Therefore, we fit the performance model to find the specific parallel fraction f of each application by executing 50 Halton configurations (F_b, F_L, b, L) . As Halton generates configurations that cover the domain more evenly than pseudo-random algorithms, 50 configurations provide sufficient data for us to build our performance model. One more configuration is needed to find $T_L(F)$, it is $b = 0$, $L = 1$, $F_b = 2$ GHz, $F = F_L = 800$ MHz. We calculated the median of the execution time of five runs for each configuration.

Considering the fitted **perf**, and the Equations (3.2) and (3.3), we use non-linear regression to find the parallel fraction f for every application, as explained in the Section 3. Table 4.1 shows the value of f for each application. Note that **Black-Scholes** and **kmeans** have a parallel fraction of 0.7743 and 0.6381, respectively, indicating a larger sequential portion of these applications when compared to other applications. The following section shows the validation of the Pareto frontier. It provides the evidence that our low-overhead models are sufficient for achieving our goals.

Table 4.1: The fitted parallel fraction values of each application.

Application	f
LavaMD	0.9961
Smallpt	0.9898
Bodytrack	0.9384
Freqmine	0.9343
Particle Filter	0.9251
x264	0.888
Black-scholes	0.7743
Kmeans	0.6381

4.3 Pareto Frontier Validation

In this Section, we validate our proposed methodology. Using the energy model in Equation (3.17) and the fitted performance model in Equations (3.2) and (3.3), we generated a pair of estimated performance and energy consumption values of all possible configurations for each application in Table 2.3. Then, the Pareto frontier method selects the performance and energy pairs that offer the optimal trade-off.

For all applications, the Pareto frontier selection step resulted in configurations that use all cores of the two clusters. The frequencies of the cluster big and LITTLE were the only parameters that have changed. This probably occurred because it was not possible to disable the unused cores during the fitting. As a result, the energy

savings from configurations, which are not using all available cores, are not enough to justify the performance mitigation. Therefore, the optimal configurations become those that use all cores.

Figure 4.3 shows the estimated and measured Pareto Frontier for all applications w.r.t. all possible configurations. Each point describes a pair of performance and energy consumption of a configuration. The grey points are the estimated outcomes of our energy and performance models for all possible configurations. In total, there are 6384 available configurations w.r.t. the combinations of the number of cores of each cluster and pair frequency.

The black circles depict the selected Pareto configurations from the estimated energy and performance pairs. Notice that the estimated Pareto have the least energy consumption concerning different performances. Each red point represents the actual values measured from the sensors using each selected Pareto configuration. Observe that the measured and estimated values are similar to each other but, as the execution time increases, the difference between the measured and estimated energy consumption also rises. The yellow triangle, the blue plus sign and the green hexagon are the **performance**, **ondemand** and **powersave** Linux governors, respectively. We compare our methodology with them in the next section.

Table 4.2 shows the total number of configurations selected by Pareto Frontier and the measured Pareto Frontier variations of energy and performance for each application. Our approach decreased the search space of 6384 available configurations for the given platform by approximately 99%. Considering the highest and least energy consumption and performance among the measured Pareto configurations, we observe energy savings of up to 68.23% and performance gains of up to 75.62% for kmeans application. The **Freqmine** application presented the least variation of energy, up to 54.35%, however, the performance gains were up to 85.41%. This demonstrates that the selected configurations by Pareto frontier can provide a good range of performance improvement and energy saving options for operating systems.

Table 4.2: Pareto Frontier energy and performance variations concerning the measured values and the total number of Pareto configurations.

	Application							
	Black-scholes	Bodytrack	Freqmine	Smallpt	x264	Kmeans	Particle Filter	LavaMD
Number of Pareto configurations	93	71	72	66	78	108	74	57
Variation of Energy (saving)	60.20%	64.71%	54.35%	59.43%	57.74%	68.23%	58.28%	54.48%
Variation of Performance (gain)	80.47%	84.84%	85.41%	89.53%	85.55%	75.62%	83.97%	88.58%

Figure 4.4 gives a zoom-in in Figure 4.3. The grey points that represent all possible estimated configurations' outcomes are not displayed. Moreover, the measured and the related estimated Pareto configurations are linked using a dotted line, thus, showing the configurations' results from the models against the measured data.

Depending on the application, we could not execute some configurations to avoid overheating the board. As anticipated in Section 4.1, high CPU frequencies produce high temperatures that make the system halt. For instance, the **Freqmine** results presented in Figure 4.4c show four estimated Pareto points with the frequencies (big,LITTLE) being (1500,2000), (1500,1900), (1400,1900) and (1500,1800) without their respective measured points. Note that two of them are overlapping. The same occurred with the **Smallpt**, **LavaMD**, **x264** and **particle Filter** applications (see Figures 4.4d,e,g,h).

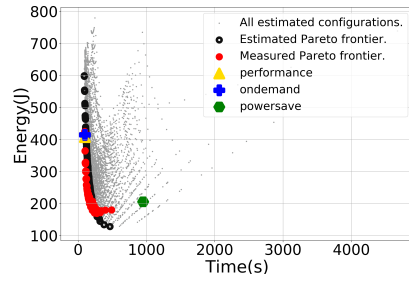
Notice that all applications except the **Particle Filter** and **kmeans** (see Figures 4.4g,f), have a point when the measured energy consumed is higher than the estimated. A vertical line represents at which time and big cluster frequency this occurs (see Figure 4.4). As the energy is also influenced by the performance modeling, the measured execution time in some cases takes longer when compared with the expected performance consuming more energy.

Notice the highlighted configurations (see Figure 4.4) that have similar measured energy. Those points have the same big frequency, and, as the LITTLE frequency scales down, the execution time increases, and so does energy consumption. Further, when the big cluster frequency decreases, there is an abrupt drop in energy consumption. For instance, this can be seen in Figure 4.4b of the **Bodytrack** application in points above 200 seconds. This shows that the big cores play an important role in energy consumption.

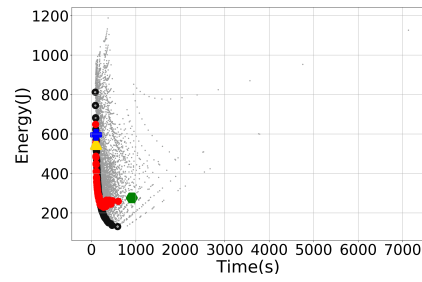
Some applications do not have a well-balanced workload as we expected. This results in the predicted performance not being well correlated with the measured data. Even though **Freqmine** (see Figure 4.4c) and **kmeans** (see Figure 4.4f) have the dynamic clause included in their parallel loops, they do not present consistent performance. Additional work is required to understand why these applications are not workload balanced. In particular, the **x264** application (see Figure 4.4e) requires more refined modelling since video processing is much more complex, and the application phases tends to vary according to the frames from the input video. Our approach shows reasonable energy savings, even though fixing a Pareto-optimal configuration to the whole application execution.

Table 4.3 shows the mean absolute percentage error (MAPE) between the measured and estimated performance and energy consumption of the Pareto Frontier, including the average standard deviation of the measured data. The highest energy and performance errors are from **Black-Scholes** (38.92%) and **x264** (11.10%) applications, respectively. The lowest errors are observed in **kmeans** and **LavaMD** with 10.48% and 1.75% of energy and performance errors, respectively. It is important to notice that the **x264** had a reasonable accuracy, considering its complexity. On average, we achieved performance and energy errors of 5.53% and 22.30%, respectively.

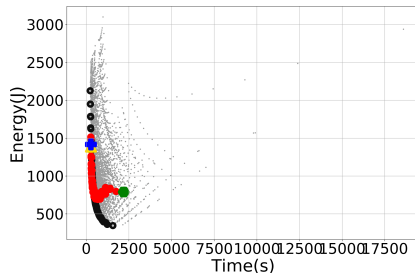
It is important to recall that our concern is the trend of the measured Pareto frontier. Therefore, even without an exact match between measured and estimated values, the Pareto configurations are reasonable alternatives that give each application suitable trade-off choices. In the next section, we compare our approach against



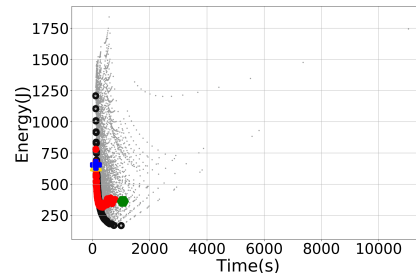
(a) Black-Scholes Parsec Application.



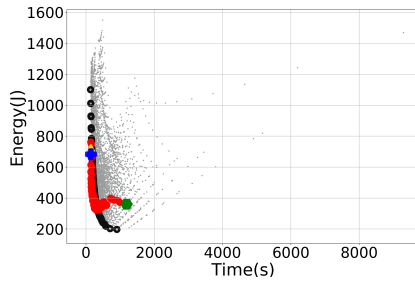
(b) bodytrack Parsec Application.



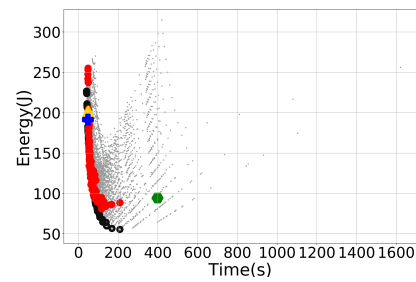
(c) Freqmine Parsec Application.



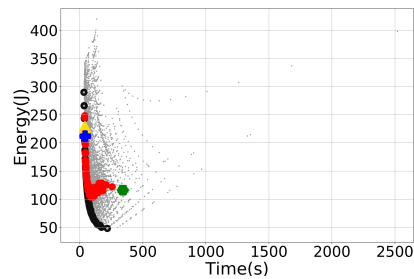
(d) Smallpt Phoronix Application.



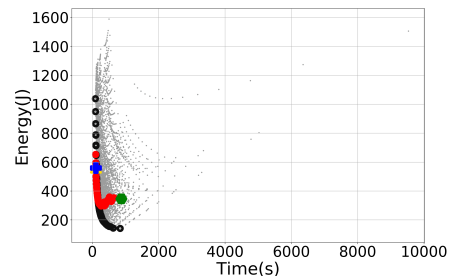
(e) x264 Phoronix Application.



(f) kmeans Rodinia Application.



(g) Particle Filter Rodinia Application.



(h) LavaMD Rodinia Application.

Figure 4.3: Estimated and measured Pareto frontier compared with the default governors, and all possible modeled configurations.

the Linux governors.

Table 4.3: The average standard deviation and mean absolute percentage error of performance and energy consumption of the estimated Pareto frontier against measured data.

		Application							
		Black-scholes	Bodytrack	Freqmine	Smallpt	x264	kmeans	Particle Filter	LavaMD
Performance	MAPE	4.88%	2.55%	6.8%	2.16%	11.10%	6.96%	8%	1.75%
	Ave. Std. Deviat.	2.4536	0.8468	17.534	0.6111	6.8414	2.4148	0.2099	0.34
Energy	MAPE	38.92%	23.97%	17.49%	27.42%	19.64%	10.48%	17.96%	22.52%
	Ave. Std. Deviat.	1.2244	1.7626	11.7941	4.9319	1.9157	7.2793	0.6401	1.1927

4.4 Comparison against DVFS Governors

All applications were executed under the **performance**, **ondemand**, and **power-save** Linux governors (see Figure 4.4). We explain how they work in Section 2.2.2. We compared our approach to these three governors as they provide a consistent variety of optimization options and are implemented on numerous mobile devices, making them competitive baselines.

The green line in Figure 4.4 shows the measured points that have higher performance and less energy consumption when compared with the **powersave** governor to all applications. Notice that there are many configurations pointed from the Pareto frontier better than the **powersave** governor. The blue and yellow lines show the measured points that have higher or similar performance when compared with the **ondemand** and **performance** governor, respectively. Notice that there are few configurations at the Pareto frontier better or similar to those governors.

4.4.1 Normalized Comparison

Figures 4.5 and 4.6 show, respectively, the normalized energy consumption and the normalized performance of all the benchmarks compared with the **performance**, **ondemand** and **powersave** governors. We chose the measured Pareto configurations that give the least energy consumption (MPLE) and the highest performance (MPHP) to normalize the energy and the performance, respectively, for each application.

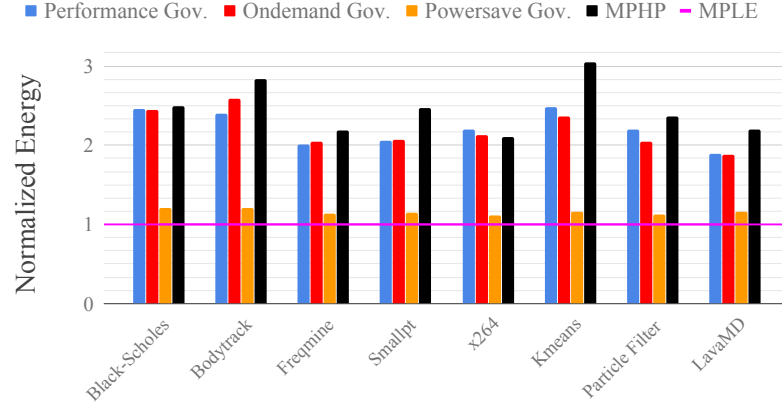


Figure 4.5: Normalized energy consumption by the measured Pareto configuration that provides the least energy consumption (MPLE) for each application.

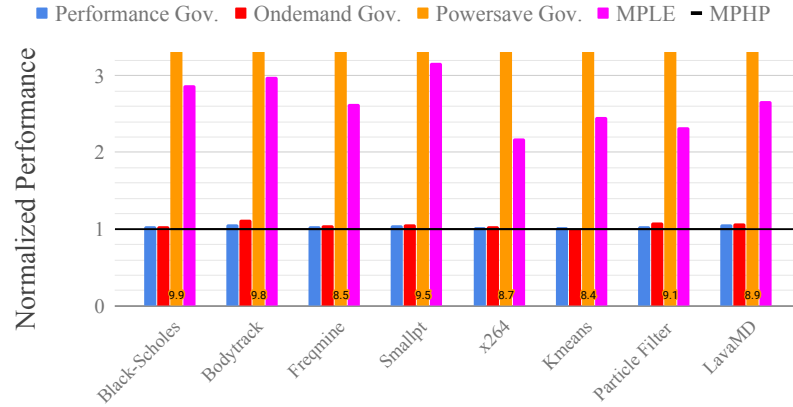


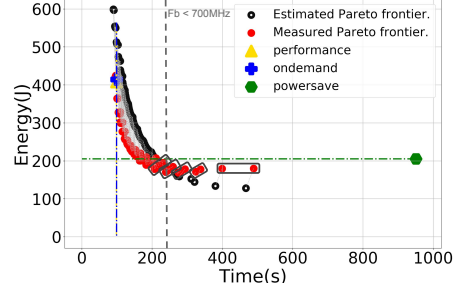
Figure 4.6: Normalized performance by the measured Pareto configuration that provides the highest performance (MPHP) for each application.

Figure 4.5 shows that the MPLE saved more energy when compared to all governors for every application. Also, we can observe that MPHP saved energy when compared to the **performance** and **ondemand** governors for the **x264** application. Figure 4.6 shows that MPHP has lower performance when compared to **performance** and **ondemand** Linux governors. Notice that MPLE has higher performance when compared with **powersave** governor for all applications.

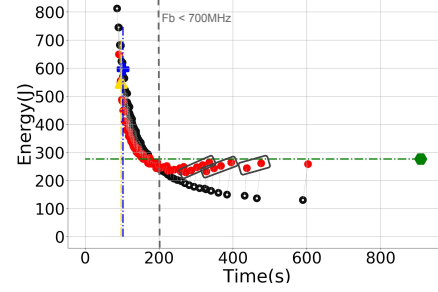
Table 4.4 shows the percentage of performance gains and energy savings when compared to the MPHP and MPLE, respectively, for all applications. Our methodology saved, on average, 54.38%, 53.99% and 13.67% of energy w.r.t. the **performance**, **ondemand**, and **powersave** Linux governors, respectively. Also, we observed 4.23%, 5.52% and 89.03% of speedup w.r.t. the **performance**, **ondemand**, and **powersave** Linux governor, respectively.

Table 4.4: Percentage of performance gains and energy savings when compared to the measured Pareto configurations that give the highest performance (MPHP) and the least energy consumption (MPLE), respectively, for all applications.

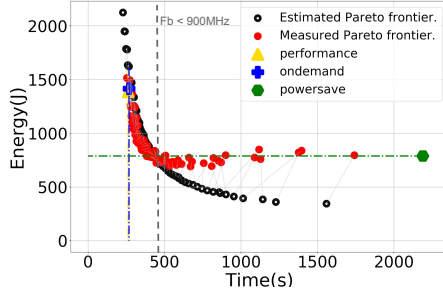
Linux Governors				
		Performance	Ondemand	Powersave
Black-Scholes	MPHP	3.69%	3.37%	89.96%
	MPLE	59.33%	59.19%	17.70%
Bodytrack	MPHP	6.42%	10.63%	89.94%
	MPLE	58.29%	61.38%	17.15%
Frequemine	MPHP	3.17%	5.24%	88.39%
	MPLE	50.11%	51.15%	12.40%
Smallpt	MPHP	4.53%	5.93%	89.54%
	MPLE	51.30%	51.68%	12.75%
x264	MPHP	2.67%	4.07%	88.48%
	MPLE	54.62%	52.88%	10.63%
kmeans	MPHP	2.86%	−0.01%	88.10%
	MPLE	59.77%	57.67%	14.07%
Particle Filter	MPHP	4.18%	8.05%	89.09%
	MPLE	54.54%	51.09%	10.67%
LavaMD	MPHP	6.29%	6.86%	88.78%
	MPLE	47.05%	46.86%	14.02%
Average	MPHP	4.23%	5.52%	89.03%
	MPLE	54.38%	53.99%	13.67%



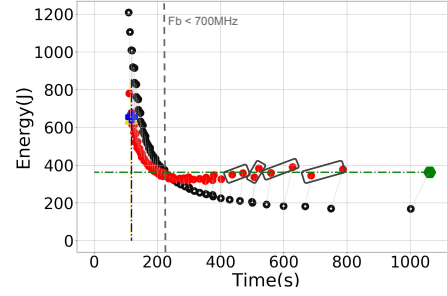
(a) Black-Scholes Parsec Application.



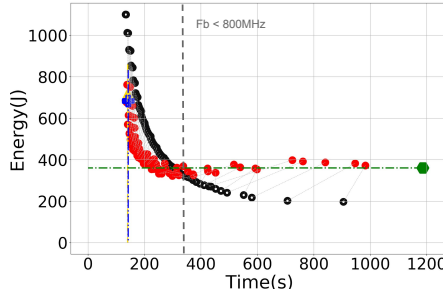
(b) bodytrack Parsec Application.



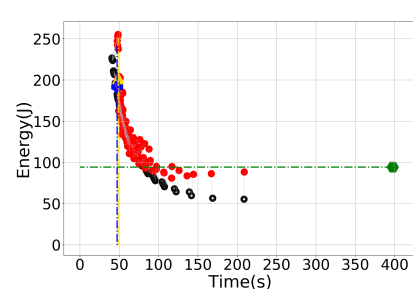
(c) Freqmine Parsec Application.



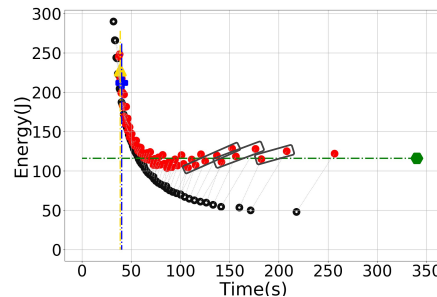
(d) Smallpt Phoronix Application.



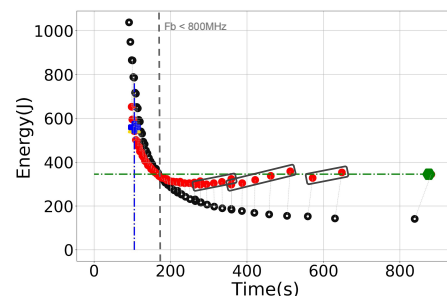
(e) x264 Phoronix Application.



(f) kmeans Rodinia Application.



(g) Particle Filter Rodinia Application.



(h) LavaMD Rodinia Application.

Figure 4.4: Estimated and measured Pareto frontier compared with the default governors removing all possible modeled configurations.

Part III

An Offline and Online Hybrid Approach for Selecting Optimal Configurations

In Part II, we showed how we could find optimal performance and energy trade-offs tailored to a single parallel application with dynamically balanced workloads for a given two-cluster device using an offline approach. This part of the thesis explores the estimated Pareto configurations coupled with Nornir’s local search strategy to dynamically find optimal performance and energy trade-offs during the application’s execution, considering the entire embedded system’s energy consumption. In Chapter 5, we describe the changes and improvements from the models presented in Chapter 3 to obtain more accurate predictions and how to capture the whole board’s power consumption. In Chapter 6, we show the results and evaluation of the combination of our improved Pareto-optimal configurations and Nornir.

Chapter 5

Selecting Optimal Configurations

In chapter 3, we explain the central methodology for finding configurations whose energy and performance trade-offs are optimal, using analytical models and Pareto frontier. This chapter describes how we include all components consumption and the frequency-voltage table to the power and energy modelling and extend our off-line methodology to suit Nornir workflow. Figure 5.1 illustrates the steps presented in Section 3.1, highlighting runtime measurement and fitting of models steps that were modified. The sampling configurations, model validation and Pareto frontier selection steps remain unchanged. An extra step is added—runtime selection—responsible for dynamically selecting desirable configurations during the application’s execution using a Pareto configuration as a start point. For this step, we use the Nornir framework, which is responsible for selecting a configuration dynamically at runtime. The following sections detail what distinguishes this part from the strategy presented in Section 3.1.

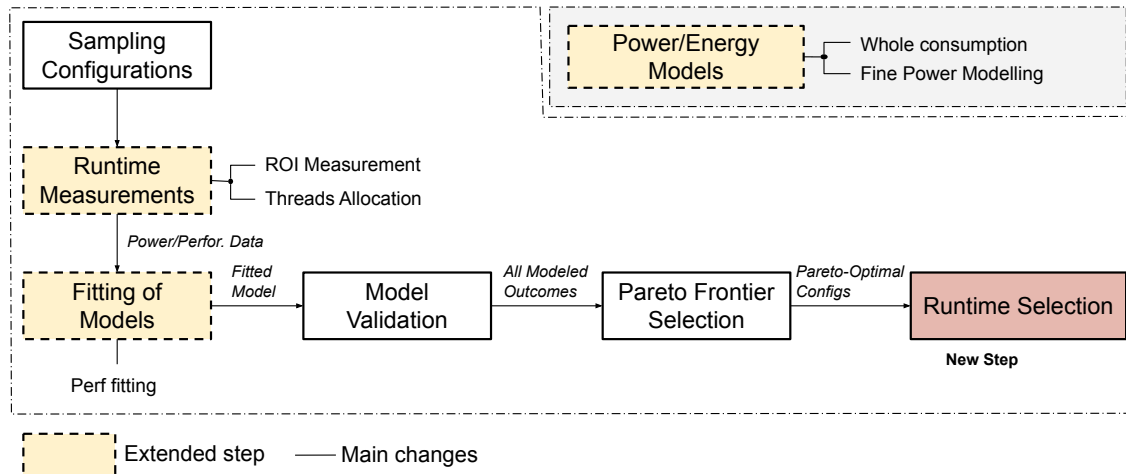


Figure 5.1: Flowchart highlights the extended steps to suit our approach of selection configurations using Pareto configurations at runtime. Also, it shows the main changed features from the power and energy modelling.

5.1 Runtime Measurements and the perf Parameter

The runtime measurement step has changed to suit Nornir runtime support. Section 5.4 explains that Nornir uses instrumentation calls placed in the main loop of the code to collect the application’s performance data. So, we collect the power consumption and performance in the region of interest (ROI) of the application instead of the entire execution as performed in the experiments of Chapter 4. This approach becomes attractive since we can diminish the interference of the multiple phases of an application. For this reason, we created a small library containing three simple functions: `iperfect_roi_init()`, which initializes the instrumentation; `iperfect_roi_begin()`, which begins collecting power data; and finally, `iperfect_roi_end()`, which terminates the power collection and saves the data. Code 5.1 shows how the functions must be placed to collect the necessary data to fit the models. As the collection is done immediately before and after the main loop, no overhead is added. Observe that begin and end functions must be placed at the same loop containing the Nornir library instrumentation calls.

```
#include <iperfect_instr.h>
...
iperfect_roi_init();
...
iperfect_roi_begin();
while(...) { // This is the main loop
... //Nornir instrumentation
... // Loop core
}
iperfect_roi_end();
```

Listing 5.1: An example of code with power collection instrumentation calls.

The application determines the number of threads, and Nornir does not change it. Moreover, Nornir’s manager may select fewer cores than the number of threads created by the application. Therefore, to obtain accurate performance/power predictions, we model the same Nornir’s thread mapping. Section 5.4) has a further explanation about the threads allocation strategy applied by Nornir. Indeed, we are extrapolating the performance model by considering more threads running per core. As explained in Section 3.2, the performance model expects that the parallel workload is dynamically distributed over the running threads. However, our performance model still applies to this scenario, as shown in the results in Chapter 4, which unbalanced applications such as x264 and freqmine obtained reasonable performance and energy prediction regarding Pareto validation. Ultimately, Nornir can balance the workload dynamically by selecting the best configuration to comply with the performance and energy requirements. During the fitting of models step, the `perf` parameter of the performance model is not agnostic in this part of the thesis. It is fitted together with the parallel fraction of the code f , i.e., they characterize the application. The parameter `perf` has the same definition as in the

previous part — capturing the performance gain in moving the computation from a LITTLE to a big core, but now, it is application-dependent. Thus, `perf` can incorporate how effectively the big cores can improve the code’s sequential performance, including several aspects of an application, such as workload and type of instructions. With these methodology adjustments, we obtain Pareto Front that provides optimal P-E trade-offs from the fitted models regarding ROI power and performance measurement. At last, Pareto frontier configurations are used as the start point of Nornir’s reconfiguration algorithm. Sections 5.2 and 5.3 describe the power and energy models to accommodate the voltage-frequency discrete table and the whole device’s consumption.

5.2 Fine Power Modelling and Whole-System Consumption

In Part II, we devised a power modelling in Section 3.3 assuming an a linear relation between frequency and voltage in a processing chip. In this part of the thesis, we devised a similar power model considering that the discrete voltage-frequency table is available. Thus, the estimated power consumption tend to be more accurate. Then, we can rewrite the Equation (3.7) regarding the power consumption of a processor with a operating frequency F as follows.

$$\begin{aligned} P(F) &= aCV^2F + VI_{\text{Leak}} \\ &= \alpha V^2F + VI_{\text{Leak}}, \end{aligned} \quad (5.1)$$

where $\alpha = aC$ and I_{Leak} are considered constants that mainly abstract semiconductor technology attributes. Then, the power of each cluster type is modelled as follows:

$$P_b(F_b, V_b, b) = b\alpha_b V_b^2 F_b + bF_b I_{\text{Leak}_b}, \quad (5.2)$$

$$P_L(F_L, V_L, L) = L\alpha_L V_L^2 F_L + LF_L I_{\text{Leak}_L}, \quad (5.3)$$

where b and L are the numbers of active big and LITTLE cores, respectively, accounted for the dynamic power modelling component. V_b, V_L are the voltage supply that the processing chip can operate beneath by operating frequency F_b and F_L for cluster big and LITTLE, respectively.

The power model aggregates the power consumption of both cluster (see Equation (3.4)), so we can combine Equations (5.2) and (5.3), to come to the model for the whole board consumption parameterized by two frequencies and voltages of each cluster as follows:

$$\begin{aligned} P_{\text{HMP}}(F_b, F_L, V_b, V_L, b, L) &= P_{\text{HMP}_P} = P_b(F_b, V_b, b) + P_L(F_L, V_L, L) \\ &= b\alpha_b V_b^2 F_b + bF_b I_{\text{Leak}_b} + L\alpha_L V_L^2 F_L + LF_L I_{\text{Leak}_L} + \theta. \end{aligned} \quad (5.4)$$

Equation (5.4) represents the power consumption of a parallel fraction (P_{HMP_P})

considering all components of a device. θ parameter is a constant power that does not vary with the voltage or frequency. Also, we define the sequential part of the model that represents the consumption for an application running in one core, as follows:

$$P_{\text{HMP}_S}(F_b, F_L, V_b, V_L) = \begin{cases} \alpha_b V_b^2 F_b + F_b I_{\text{Leak}_b} + F_L I_{\text{Leak}_L} + \theta, & \text{if } b > 0, \\ \alpha_L V_L^2 F_L + F_L I_{\text{Leak}_L} + \theta, & \text{otherwise.} \end{cases} \quad (5.5)$$

When there is, at least, one big core available, we consider that the sequential part runs in one core big—i.e. $b = 1$ and $L = 0$, resulting in dynamic and static power consumption of one core big. As essential kernel processes are bound to core 0 (LITTLE core) and cannot be migrated to another core, we never disable core 0. As a result, one static power component of the core LITTLE is present for the $b > 0$ case. Otherwise, the sequential part runs on one LITTLE core, computing only the dynamic and static power of one core LITTLE.

5.3 Energy Modelling

In Part II, we describe the energy modelling presented in Section 3.4, as a combination of the sequential and parallel energy consumption for a given application running on a specific HMP. The same principle takes place to model the energy consumption from the whole device as follows:

$$E_{\text{HMP}}(F, F_b, F_L, b, L) = E_{\text{HMP}_S} + E_{\text{HMP}_P}. \quad (5.6)$$

Using the power model in Section 5.2 and the same performance model from Section 3.2, we can devise the parallel and sequential part of the energy model. The Equation (5.7) represents the parallel portion energy model component by combining the parallel part of the performance Equation (3.2) and the power model of the whole device in Equation (5.4).

$$E_{\text{HMP}_P} = \frac{T_L(F) f F (b \alpha_b V_b^2 F_b + b F_b I_{\text{Leak}_b} + L \alpha_L V_L^2 F_L + L F_L I_{\text{Leak}_L} + \theta)}{b \cdot \text{perf} \cdot F_b + L F_L}. \quad (5.7)$$

Using the power Equation (5.5) and the sequential part of the performance Equations (3.2) and (3.2), we have the sequential portion energy model component, which relies on the number of active big cores available:

$$E_{\text{HMP}_S} = \begin{cases} \frac{T_L(F)(1-f)F(\alpha_b V_b^2 F_b + F_b I_{\text{Leak}_b} + F_L I_{\text{Leak}_L} + \theta)}{\text{perf} \cdot F_b}, & \text{if } b > 0, \\ \frac{T_L(F)(1-f)F(\alpha_L V_L^2 F_L + F_L I_{\text{Leak}_L} + \theta)}{F_L}, & \text{otherwise.} \end{cases} \quad (5.8)$$

Finally, combining Equations (5.7) and (5.8), we come to the consolidated energy model for the whole device parameterized by the frequencies, voltage and the number

of active cores in each cluster:

$$E_{\text{HMP}}(F, F_b, F_L, b, L) = T_L(F)F \times \begin{cases} \frac{(1-f)(b\alpha_b V_b^2 F_b + bF_b I_{\text{Leak}_b} + L\alpha_L V_L^2 F_L + LF_L I_{\text{Leak}_L} + \theta)}{\text{perf} \cdot F_b} + \frac{f(b\alpha_b V_b^2 F_b + bF_b I_{\text{Leak}_b} + L\alpha_L V_L^2 F_L + LF_L I_{\text{Leak}_L} + \theta)}{b \cdot \text{perf} \cdot F_b + LF_L}, & \text{if } b > 0, \\ \frac{(1-f)(\alpha_L V_L^2 F_L + F_L I_{\text{Leak}_L} + \theta)}{F_L} + \frac{f(L\alpha_L V_L^2 F_L + LF_L I_{\text{Leak}_L} + \theta)}{LF_L}, & \text{otherwise.} \end{cases} \quad (5.9)$$

We can now estimate energy consumption and the performance of any configuration for a given parallel application using our energy model in Equation (5.9) and performance models in Equations (3.2) and (3.3).

5.4 Runtime Selection

We use Nornir to enable a dynamic allocation of resources to parallel applications according to performance and energy consumption requirements. The framework uses some self-adaptive strategies already provided by the framework or customized by adding new self-adaptive algorithms. A person in charge of creating new self-adaptive strategies can focus on the algorithm by exploiting the framework's infrastructure. Figure 5.2 shows the general architecture of the Nornir framework¹.

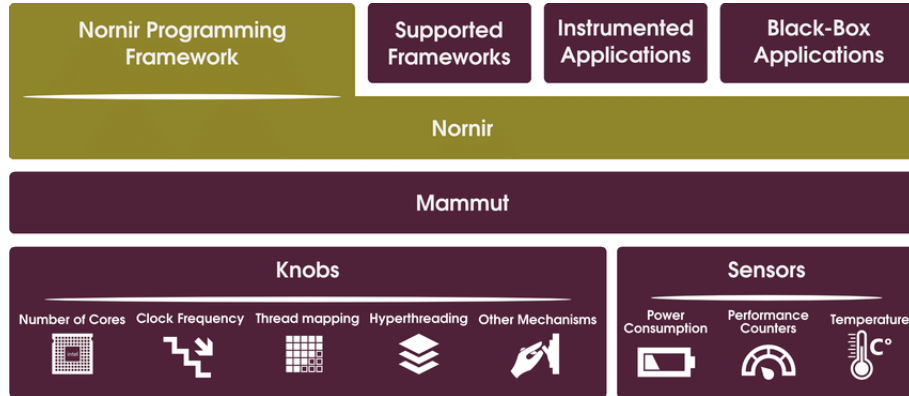


Figure 5.2: General Architecture of Nornir Framework (De Sensi et al. 2018).

Nornir can control different applications by coupling them with a manager process. At each sampling interval, the manager acts by executing an iteration of the so-called Monitor-Analyze-Plan-Execute loop (MAPEL) (Kephart & Chess 2003). MAPEL loop is composed of a Monitor phase, which collects data about the application; by Analyze phase, which check whether the monitored data is compliant with

¹Available at <http://danieledesensi.github.io/nornir/>

the user’s requirement or not; by the Plan phase, which is triggered to decide the actions to be executed to achieve the goals specified by the user; and, by Execute phase which applies the actions (e.g. scale up/down the clock frequency of the CPU) decided by the Plan strategy. To implement the monitor and execute phases, Nornir interacts with the system knobs and sensors by using the Mammut library (De Sensi et al. 2017).

The user can enforce performance and performance/energy consumption requirements throughout applications execution by specifying them in an XML file. The configuration file at Listing 5.2 shows an example of specifications that can be used to request Nornir to find a configuration described by a power consumption lower than 100 Watts and a by the highest throughput. We list in Table 5.1 the metrics used to implement different scenarios in this work.

```
<?xml version="1.0" encoding="UTF-8"?>
<nornirParameters>
<requirements>
<throughput>MAX</throughput>
<powerConsumption>100</powerConsumption>
<requirements>
</nornirParameters>
```

Listing 5.2: An example of user requirements.

Table 5.1: Main requirements specified in this thesis.

Metric	Description
Throughput	Number of iterations executed per second.
Power Consumption	Instantaneous power consumption. It corresponds to the system power consumption.
Energy	Power integrated over time.
Expected Tasks Number	The expected number of iterations to be executed.

Monitoring some metrics (e.g., power consumption, temperature or current clock frequency) do not require a direct interaction between the manager and the application. However, the collection of other metrics, like performance and application-specific data, requires an explicit interface between Nornir Manager and the target application. One possibility is to insert instrumentation calls provided by the Nornir framework. These calls are invoked at each iteration to collect all needed performance data to take reconfiguration decisions. It is essential to underline that Nornir uses an adaptive sampling mechanism, which automatically selects the appropriate sampling length to control overhead below 1% (De Sensi et al. 2018). We use existing applications previously instrumented for our experiments in Section 6.1.2.

5.4.1 Analyze-Plan Algorithm

Nornir provides various online algorithms for deciding how many resources to use according to the application requirements. Nevertheless, only one runs for HMP

systems, a Nelder-Mead algorithm starting from an initial guess. The Nelder-Mead algorithm, originally published in 1965 by John Nelder and Roger Mead (Nelder & Mead 1965), is an optimization algorithm for non-linear multidimensional unconstrained optimization without derivatives. As it does not require any derivative information, it is suitable for optimization problems where the function's gradient is unknown or cannot be reasonably computed (Singer & Nelder 2009).

The algorithm uses a simplex concept, a shape structure composed of $n + 1$ vertices in n dimensions, such as a triangle on a plane, a tetrahedron in three-dimensional space. In general, at each iteration, it seeks to exclude the vertex with the worst function value and replace it with another point with a better value (Nocedal & Wright 2006). The new point is achieved by reflecting, expanding, or contracting the simplex along the line that joins the worst vertex with the centroid of the remaining vertices. If a better point is not found in this manner, the vertex with the best function value remains, and the simplex shrinks by moving all other vertices toward this value. The search stops when the points converge on an optimum, when a minimum difference between evaluations is observed, or when a maximum number of function evaluations are performed. An example of a visual Nelder-Mead search for the Himmelblau's function can be seen in Figure 5.3.

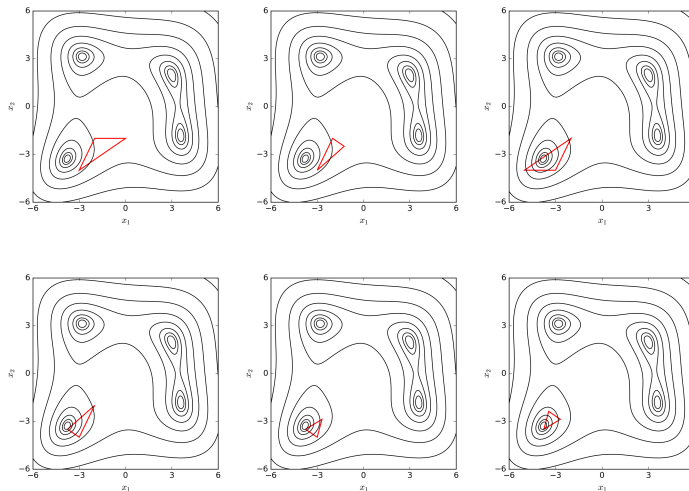


Figure 5.3: An example of Nelder–Mead simplex search over the Himmelblau's function ^a.

^aAdapted from https://en.wikipedia.org/wiki/File:Nelder-Mead_Himmelblau.gif

The method can take many iterations with a slight improvement in function value, despite being far from a minimum, causing not locating the global optimum (Singer & Nelder 2009). Therefore, an approach to deal with such cases is to restart the algorithm several times with different starting points (Nocedal & Wright 2006). Furthermore, the Nelder-Mead may benefit from an endpoint of another global optimization algorithm, such as an optimal Pareto Front.

Listing 1 shows an XML example that provides the parameters and requirements for executing the Nelder-Mead algorithm. The `nelderMeadRange` parameter determines how large is the search area. In the example, the algorithm checks the configurations with ± 4 cores, ± 4 frequency levels for each cluster. Also, the first-Configuration parameter describes the initial configuration, indicating the number of cores (virtualCores) and the frequencies (kHz) (frequency) of the LITTLE and big cluster.

```
<?xml version="1.0" encoding="UTF-8"?>
  <nornirParameters>
    <strategySelection>HMP_NELDERMEAD</strategySelection>
    <nelderMeadRange>4</nelderMeadRange>
    <requirements>
      <throughput>MAX</throughput>
      <powerConsumption>100</powerConsumption>
    </requirements>
    <firstConfiguration>
      <virtualCores>4:4</virtualCores>
      <frequency>1400000:1100000</frequency>
    </firstConfiguration>
  </nornirParameters>
```

Listing 5.3: An example of XML parameters using Nelder-Mead.

Nornir adopts a round-robin policy for mapping threads for each core during a parallel application's execution. The threads are firstly fixed to the cores big and after to the core LITTLE. For example, considering an application has eight working threads, and the planning algorithm selects three LITTLE (L) and two big (b) cores, then each thread is assigned to an active core, and the extra three threads are mapped to 2 big cores and then to one LITTLE core. Figure Figure 5.4 depicts this example.

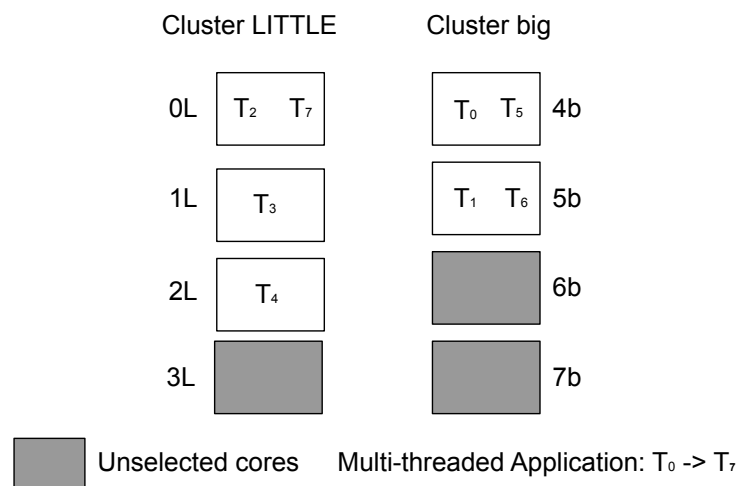


Figure 5.4: An example of Nornir's threads mapping policy.

Chapter 6

Evaluation and Results

In this chapter, we validate the Pareto configurations and the online search configurations using an ODROID-XU3 board. Section 2.3.1 explains our experimental setup, including the specifications of the platform. For this work, the experimental platform was set up with Ubuntu 16.04, running kernel 3.10 since it permits disabling individual cores. In this kernel version, the maximum frequency of cluster LITTLE is 1.4 GHz — not 1.5 GHz as the newest version. Thus there are 13 available frequency levels from 200 MHz to 1.4 GHz. Also, we are not considering the big cluster frequencies higher than 1.8 GHz to maintain the temperature below the threshold of the CPU throttling. Consequently, 17 available frequency levels are ranging from 200 MHz up to 1.8 GHz on cluster big. Considering there are five options of how many cores can be selected for each cluster (none to four cores), then there exist 5525 combinations of cores and frequency $((5 \times 13) \times (5 \times 17) = 5525)$. However, the combination of none cores big and none LITTLE cores are not possible. At least one core needs to be active to an application be able to run. Finally, the final total number of available configurations is $5525 - (13 \times 17) = 5304$.

As explained in Section 2.3.2, we use the ODROID power-supply Smartpower2 to measure the whole board with a sample reading every 250 milliseconds with timestamps. Then, by integrating the power samples, we computed the energy used by a given application. We consider the average power calculated by dividing total energy by execution time as presented in Chapter 4. In Section 6.1, we show the validation of our models and Pareto-optimal frontier. Eventually, in Section 6.2 we compare the hybrid approach for finding optimal configurations at runtime starting with Pareto configurations against non-Pareto and Linux Governors.

6.1 Modelling Fitting Results

This section presents the values obtained for the parameters in the power and performance models. Also, we show a comparison between the predicted and the actual values for each model and Pareto configurations.

6.1.1 Power Consumption Parameters

Aiming to represent the power consumption of a whole HMP board, we stress the CPU using `stress-ng` with `callfunc` method (See Section 4.1). As we want to model the total power consumption considering all components, we measured the whole board to obtain P_{HMP} . We pinned each process for each core to execute the `stress-ng` by generating 100 evenly distributed Halton configurations running for 100 seconds. As we have five more configurations when comparing Section 4.1.2, the Halton generator can provide different configurations to keep the configuration samples evenly distributed among the configuration space. Each configuration is composed of the number of active big cores b , active LITTLE cores L , big's frequency cluster F_b and LITTLE's frequency cluster F_L . This strategy tries to incorporate the hardware design characteristics, not taking into account the target applications.

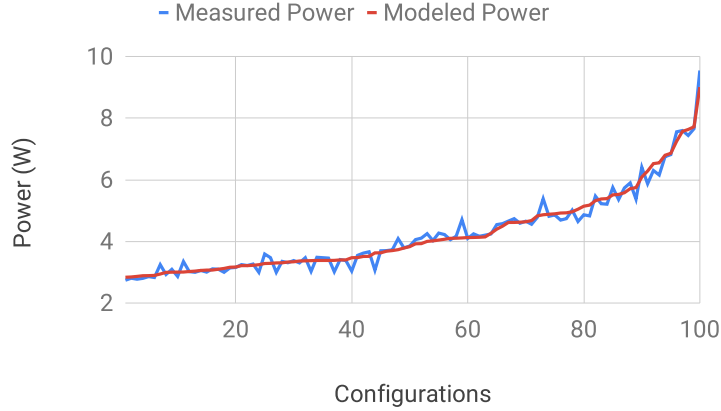
The power is the average of the samplings readings for each configuration. As the voltage-frequency table (See Section 2.3.1), the configuration parameters and the total power consumption measurements are known, we fit the parameters $\gamma_b, \gamma_L, I_{\text{leak}_b}, I_{\text{leak}_L}, \theta$ from the Equation (5.4). The values found are: $\gamma_b = 5.876 \times 10^{-10}$, $\gamma_L = 1.073 \times 10^{-10}$, $I_{\text{leak}_b} = 5.656 \times 10^{-24}$, $I_{\text{leak}_L} = 2.290 \times 10^{-18}$, $\theta = 2.810$.

We collected the power consumption of 100 random configurations with uniform distribution to validate our fitting process. Figure 6.1 shows the modelled and measured value for Halton and random configurations. Observe that both Halton configurations used for power fitting and random configurations have an excellent approximation, following the measured power consumption trend. The mean absolute error between estimated and measured is 4.4% and 3.5% for Halton and random configurations. Moreover, the average standard deviations are 1.31 and 1.60 for Halton and random configurations, respectively.

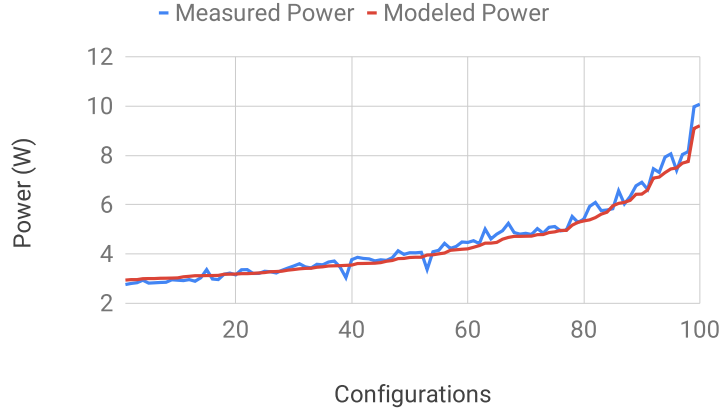
6.1.2 Performance Parameters

We used `Black-Scholes` and `Bodytrack` P³ARSEC applications, which contains parallel patterns implementations of some applications included in the PARSEC benchmark (De Sensi et al. 2017). In addition, P³ARSEC applications have Nornir instrumentation calls to provide dynamic reconfiguration of the applications by relying on the Nornir framework. As we are using the OpenMP version, these applications do not differ in implementation presented in Section 2.3.3. Also, we selected these two since they relatively demand less time to run among the three parsec applications executed in the experiment of Part II. Ultimately, the results here are preliminary and suggest that we are on the right track.

As explained in Chapter 5, we aim to find the optimal configuration that provides the best performance and energy trade-offs for the application's ROI section at runtime. Then, besides Nornir instrumentation, we also added our extra instrumentation calls to measure the performance and power for fitting purpose. Further technical explanation about the software instrumentation used to model the applications can be found in Section 5.1. We try to capture the distinctions of execution time produced by the different arrangements of cores and frequencies. Therefore,



(a) Measured and modelled power values of Halton configurations for power model fitting.



(b) Measured and modelled power values of random configurations for power model validation.

Figure 6.1: Measured and Modelled power consumption of 100 Halton and random configurations ordered by modelled power.

we fit the performance model to find the specific parallel fraction f and **perf** parameter of each application by executing 50 Halton configurations (F_b, F_L, b, L) . Also, the configuration $b = 0$, $L = 1$, $F_b = 2$ GHz, $F = F_L = 800$ MHz is necessary to measure the value of $T_L(F)$. We calculated the median execution time of five runs for each configuration.

Considering the Equations (3.2) and (3.3), we use non-linear regression to find the parallel fraction f and the **perf** parameter for the two applications, as explained in Section 3.1. Then, the parallel fraction of f and the **perf** for Black-Scholes are 0.98 and 1.99, respectively. For Bodytrack, f and the **perf** are 0.84 and 1.42, respectively. Note that Black-Scholes has a higher parallel fraction than Bodytrack, which did not occur considering the entire application presented in Section 4.1.2. This indicates that the Black-Scholes' ROI section is more parallel when com-

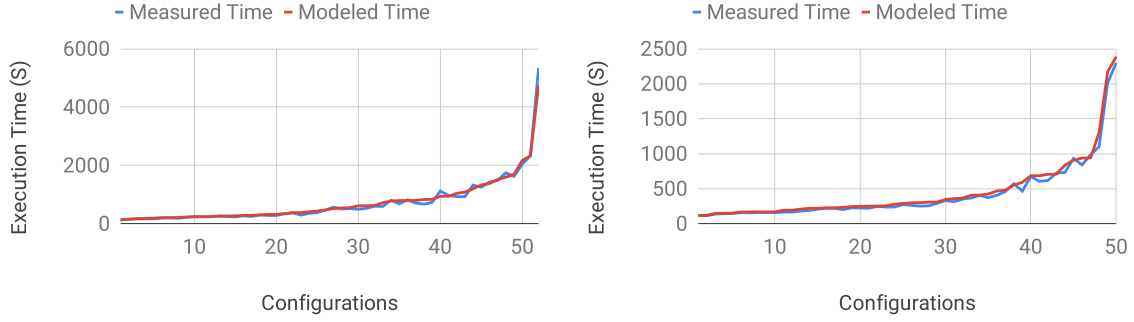
pared to **Bodytrack**. Considering the **perf**, the cores of cluster big provide almost twice more performance compared to cluster LITTLE for **Black-Scholes**—similarly, $1.42\times$ more performance from LITTLE to big cores for **Bodytrack**. In the next section, we show the errors and standard deviation from our models.

6.1.3 Modelling Validation

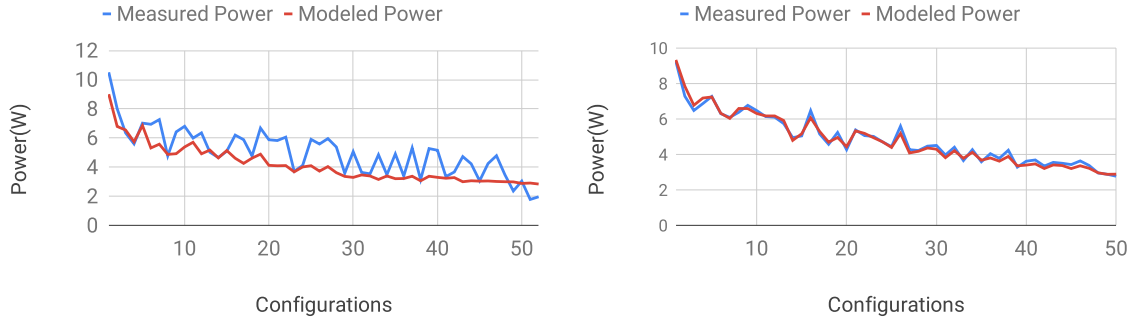
The Figure 6.2 shows a comparison of measured and modelled values for Halton and random configurations with uniform distribution for **Black-Scholes**. There are 50 Halton configurations used for fitting our models, and the same number for random configurations is used to show each model’s goodness. Execution time, power, and energy charts are ordered in ascending order by the modelled time. So, it means that fewer cores and lower frequencies are found at the most right. Consequently, Figure 6.2a shows the execution time on an ascending curve with an excellent match between measured and modelled. Figure 6.2c shows a power consumption descent curve as fewer resources are being used. Although the difference between modelled and measured power values can reach up to 2W for few configurations, it still follows the measured power’s curve trend. In the Figure 6.2e, the modelled energy follows the trend of the measured values, with some accumulated errors due to power and execution time errors. Note how the execution time has a significant impact on energy consumption for a low number of resources.

Looking at Figures 6.2b, 6.2d and 6.2f for the random configurations, they have a great match for the modelled and measured values. Thus, it may indicate a good approximation of the models and the application execution for points not included in the fitting process. Ideally, we should have executed for all possible configurations to verify the goodness of the model. However, it would be unfeasible to execute thousands of configurations for all applications. As the number of cores and frequencies are decreased, the application can take several hours to end. It would take an excessive number of days to validate the models for one single application. Besides, our purpose is to find the optimal configuration without exploring the whole design space domain. Ultimately, we would be contradicting in doing such experiments for model validation purpose.

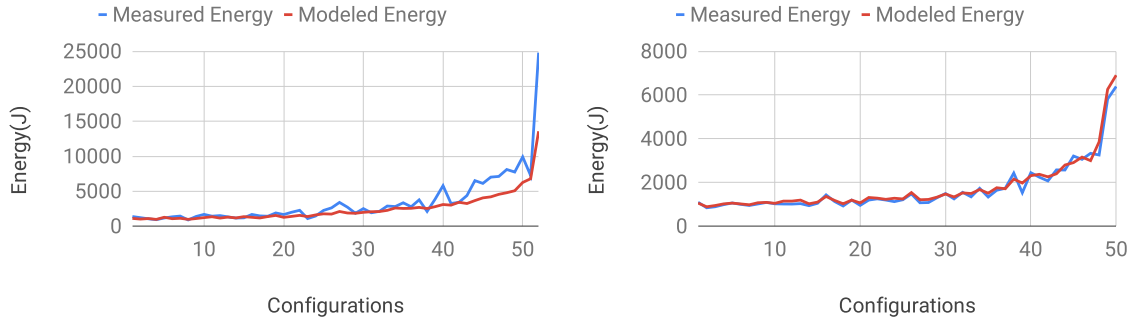
The Figure 6.3 shows a comparison of measured and modelled values for Halton and pseudo-random configurations with uniform distribution for **Bodytrack**. In Figures 6.3a and 6.3b, there are two configurations whose measured execution time is very higher when compared to the expected. However, the model seems to follow the curve trend. In Figures 6.3c and 6.3d, the modelled power has a great approximation from the 25th to 50th configuration, meaning good results for low resources. Moreover, In Figures 6.3e and 6.3f, the modelled energy has accumulated the performance errors, yet following the curve trend. Table 6.1 shows the average standard deviation for each metric and the MAPE between measured and modelled values. Observe that MAPE decrease from Halton to random configurations regarding **Black-Scholes**. It means that the actual values are closer to the model’s prediction for random configurations. For **Bodytrack**, time and Energy



(a) Model fitting with Halton configurations. (b) Model validation with random configurations.



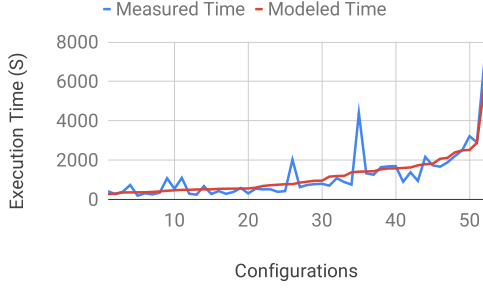
(c) Model fitting with Halton configurations. (d) Model validation with random configurations.



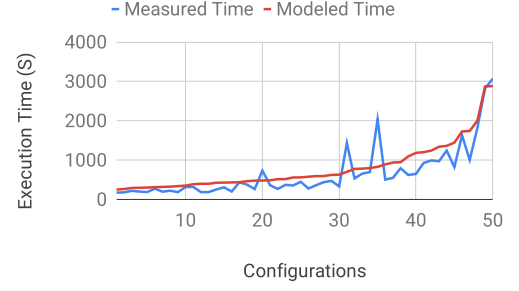
(e) Model fitting with Halton configurations. (f) Model validation with random configurations.

Figure 6.2: Measured and modelled metric values comparison for 50 Halton and random configurations for **Black-Scholes** application ordered by modelled time.

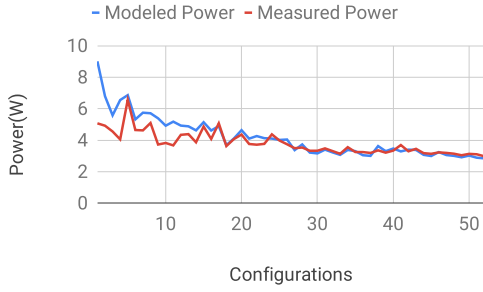
MAPE increased 10% from Halton to random configurations. According to the errors and figures presented, the power model's prediction is reasonable even though it is application-agnostic for the given applications.



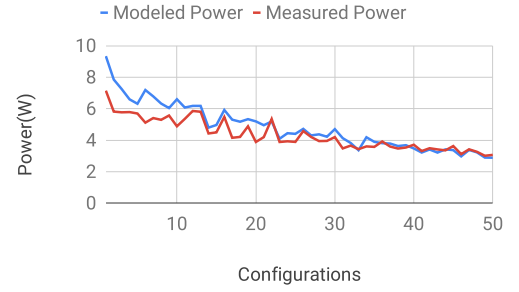
(a) Model fitting with Halton configurations.



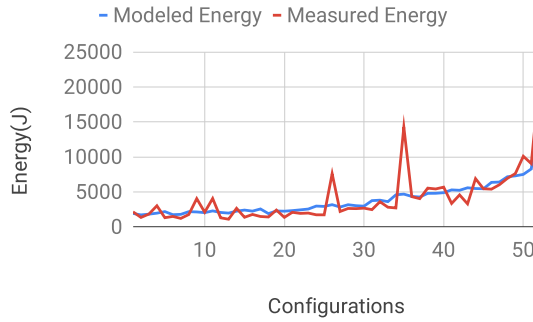
(b) Model validation with random configurations.



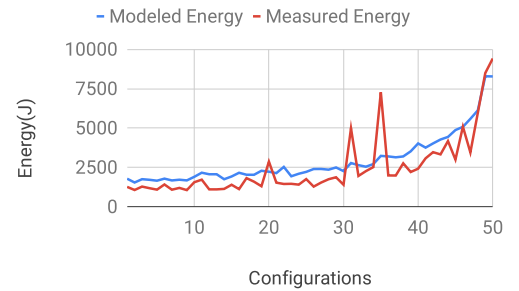
(c) Model fitting with Halton configurations.



(d) Model validation with random configurations.



(e) Model fitting with Halton configurations.



(f) Model validation with random configurations.

Figure 6.3: Measured and modeled metric values comparison for 50 halton and random configurations for `Bodytrack` application ordered by modeled time.

6.1.4 Pareto Frontier Validation

This section shows our proposed offline methodology's validation considering the ROI part of the code and the whole energy consumption. Using the fitted performance and energy model (5.9) for each application, we generated a pair of estimated performance end energy consumption for all possible configurations. Then, the Pareto frontier method chooses the pairs that offer the optimal trade-off.

Table 6.1: Mean absolute percentage error (MAPE) between the measured and modelled time, power and energy between values, including the average standard deviation of the measured data for **Black-Scholes** and **Bodytrack**.

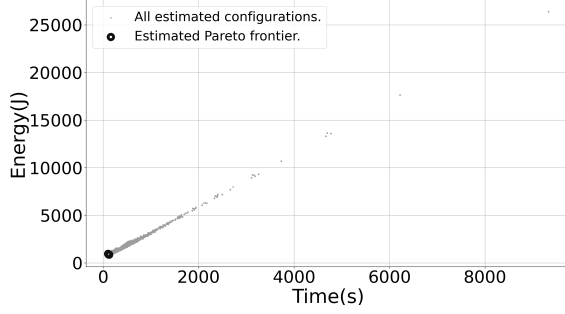
Applications		Halton			Random		
		Ave.	Std. Dev	MAPE	Ave.	Std. Dev	MAPE
Black-Scholes	Time	2.28		8.80%	1.28		9.32%
	Power	0.04		19.29%	0.006		3.41%
	Energy	40.22		19.77%	7.31		7.28%
Bodytrack	Time	5.20		38.33%	2.64		48.02%
	Power	0.008		11.63%	0.009		11.78%
	Energy	19.47		31.91%	11.61		41.35%

Figure 6.4 has six charts that show different perspectives of the Pareto frontier configurations generated by the models for all applications. Figures 6.4a and 6.4b show the estimated outcomes of our energy and performance models for all possible configurations regarding **Black-Scholes** and **Bodytrack**, respectively. There are 5304 available configurations concerning the combinations of the number of cores of each cluster and pair frequency. In Figures 6.4c and 6.4d, the x-axis is limited by the energy and time values of the **powersave** Linux governor¹. It is possible to show where measured and estimated Pareto and the **powersave** are placed among several estimation points. Figures 6.4e and 6.4f shows the estimated and measured Pareto configurations and the default governors removing all possible modeled configurations.

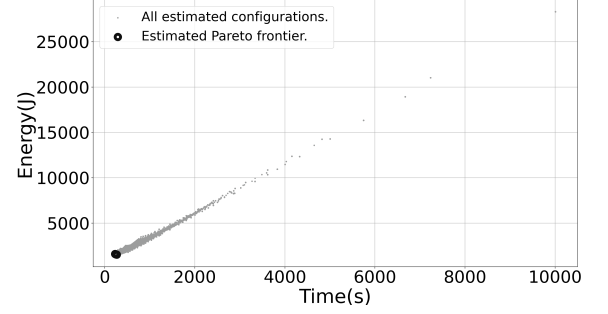
Comparing with Table 4.2 the number of Pareto points decreased from tens to some units. Moreover, the Pareto frontier selection step resulted in configurations that use all cores of the two clusters as it occurred in Section 4.3. We believe it occurred caused by the small variation in energy consumption and execution time values that justifies a more significant amount of Pareto-optimal points. Another important point that we demonstrate in Section 6.2.2 is that the parallel applications in this hardware run optimally with all eight cores. Table 6.2 shows that we obtained small performance and energy variations gains of Pareto measured values, and performance error is higher than presented in Table 4.3. Moreover, **Black-Scholes** and **Bodytrack**, respectively, have lower and higher energy error compared to Table 4.3 values. It is important to recall that our concern is the trend of the measured Pareto frontier.

We chose the measured Pareto configurations that give the least energy consumption (MPLE) and the highest performance (MPHP) to compare the energy and the performance against the Linux governors, respectively, for each application. Table 6.3 shows a comparison between MPHP and MPLE against the governors. The negative percentage means that a given governor has higher performance or

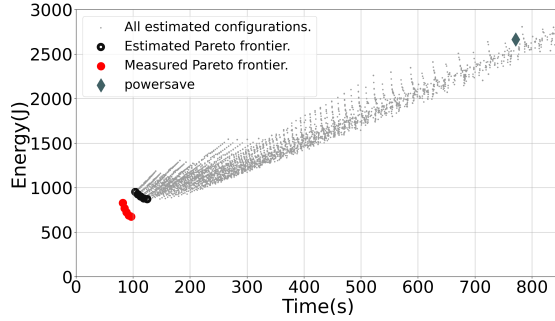
¹We explain how the governors work in Section 2.2.2.



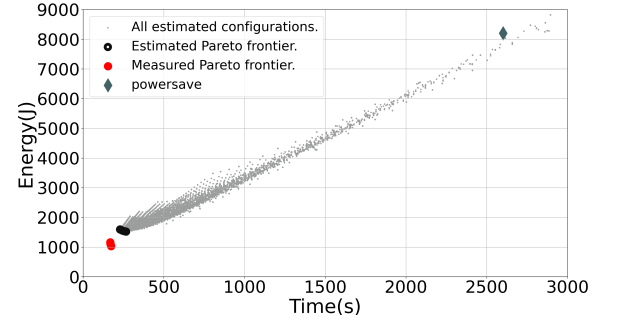
(a) All estimated configurations for Black-Scholes.



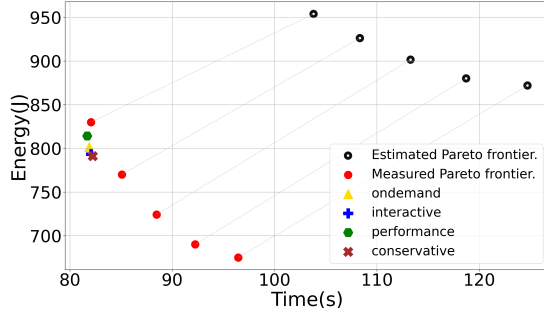
(b) All estimated configurations for Bodytrack.



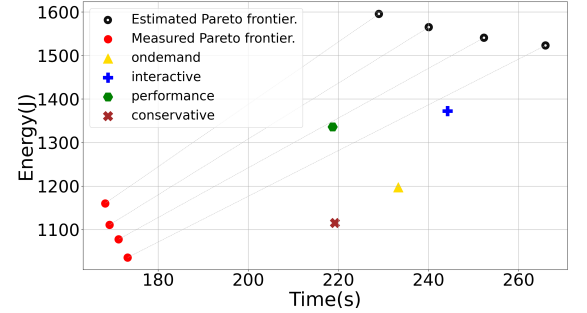
(c) Measured and estimated Pareto frontier for Black-Scholes.



(d) Measured and estimated Pareto frontier for Bodytrack.



(e) Measured Pareto frontier and Governors Comparison for Black-Scholes.



(f) Measured Pareto frontier and Governors Comparison for Bodytrack.

Figure 6.4: Estimated and measured Pareto frontier compared with the default governors and all possible modelled configurations.

higher energy savings than the respective Pareto configuration. Thus, MPHP wins over **powersave** and **conservative**, and MPLE is up to 17.09% of energy savings with a performance degradation of up to -18.07% for Black-Scholes PARSEC application. Considering Bodytrack application, only **conservative** governor has higher energy savings when compared to MPHP.

Table 6.2: Pareto Frontier energy and performance variations concerning the measured values, mean absolute percentage error (MAPE) between the measured and modelled performance and energy consumption of the Pareto Frontier and average standard deviation of the measured data for each application.

		Application	
		Black-Scholes	Bodytrack
Performance	MAPE	27.98%	44.44%
	Variaton	14.91%	2.87%
	Ave. Std. Deviat.	0.08	0.46
Energy	MAPE	23.28%	40.48%
	Variaton	18.67%	12.19%
	Ave. Std. Deviat.	1.22	5.31

Table 6.3: The percentage of performance gains and energy savings when compared to the MPHP and MPLE, respectively, for each application.

Black-Scholes				
MPHP			MPLE	
Governor	Performance	Energy	Performance	Energy
ondemand	-0.22%	-3.52%	-17.77%	15.81%
interactive	-0.02%	-4.62%	-17.54%	14.91%
powersave	89.36%	68.85%	87.50%	74.67%
performance	-0.47%	-1.94%	-18.07%	17.09%
conservative	0.20%	-4.90%	-17.29%	14.69%
Bodytrack				
MPHP			MPLE	
Governor	Performance	Energy	Performance	Energy
ondemand	27.77%	0.65%	25.63%	12.76%
interactive	31.00%	13.33%	28.96%	23.90%
powersave	93.52%	85.50%	93.33%	87.27%
performance	22.96%	10.97%	20.69%	21.82%
conservative	23.14%	-6.66%	20.87%	6.35%

6.2 Hybrid Approach Validation

We detail the Nornir framework in Section 5.4 which has a Nelder-Mead algorithm implementation to search for an HMP configuration (b, l, F_b, F_L) accordingly with specified requirements. Nornir performs a processing cycle to define or not a new configuration every 2.5 seconds. Also, this Section presents the results of using Pareto and non-Pareto configuration as a start point of the Nelder-Mead search algorithm implemented by Nornir framework (See Section 5.4.1). Moreover, we compare Nornir with a Pareto configuration and the default governors.

6.2.1 Pareto X Non-Pareto

We explore the Pareto configurations selected by our methodology, creating two scenarios: Nornir configured to provide maximum throughput for an application, called Nornir HP (Highest Performance); and Nornir configured to deliver the least energy consumption for an application, called Nornir LE (Least Energy). For each scenario, we provide a Pareto configuration provided by the methodology. For Nornir HP, we use a Pareto configuration that yields the estimated Highest Performance (HP Pareto) and another Pareto which deliver the estimated Least Energy consumption (LE Pareto) for Nornir LE.

Figure 6.5 shows the execution time and energy consumption for HP and LE Pareto, and Nornir HP and LE with Pareto and random configurations for **Black-Scholes** and **Bodytrack**. For the non-Pareto scenarios, the values presented are a median of 25 random configurations. For the Pareto Scenarios, the values are a median of five runs with an average standard deviation of 7.37 and 32.21 for time and energy, respectively. Observe that HP Pareto has a faster execution time when compared to Nornir HP for both applications. Also, LE Pareto has lower energy consumption when compared to Nornir LE for both applications. We believe this occurred since the processing overhead makes Nornir slower and consumes more energy than their respective fixed Pareto configuration. However, note that Nornir with a Pareto configuration yields better results than the median random configuration considering both applications. On average, Nornir HP and LE - Pareto runs 15% and 44% faster and saves 1% and 32% of energy, respectively, compared to random configuration. It indicates that the Pareto frontier selected by our models give a significant advantage to a dynamic selection of resources allocation as the Nelder-Mead algorithm of Nornir.

We showed that the Nornir with a Pareto configuration has better results than a median of 25 random configurations. Now, we want to investigate further that nornir benefits from Pareto points by looking into the distribution of random configurations. Figure 6.6 shows a boxplot graph of execution time and energy consumption for Nornir HP and Nornir LE with 25 random configurations considering **Black-Scholes**. Figure 6.6a shows that the lowest execution time is 93.43 seconds which is higher than Nornir HP with Pareto (=90.7 sec) for **Black-Scholes**. Considering Nornir LE with random configurations, the lowest execution time is 101.83, which is

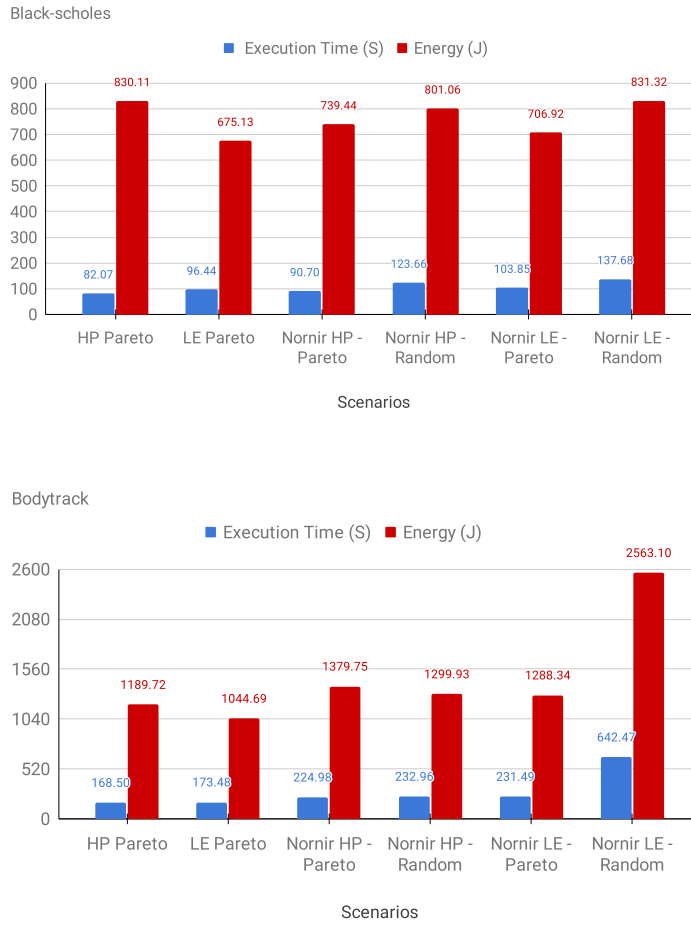
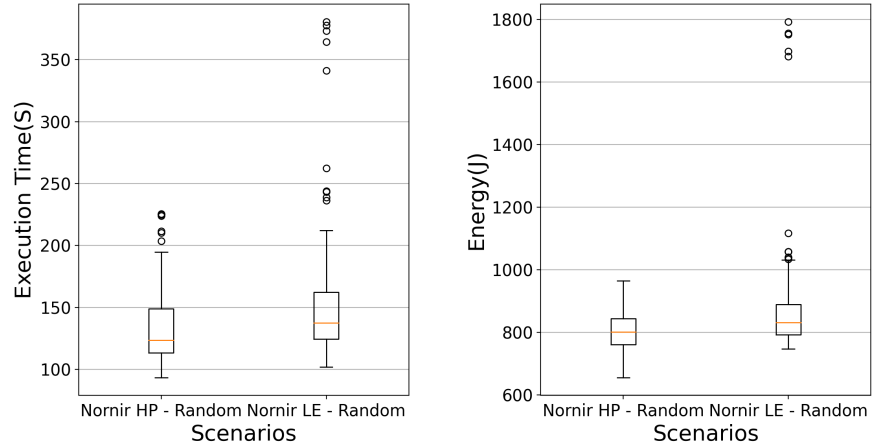


Figure 6.5: Measured execution time and energy consumption for HP and LE Pareto, Nornir HP and LE with Pareto and random configurations.

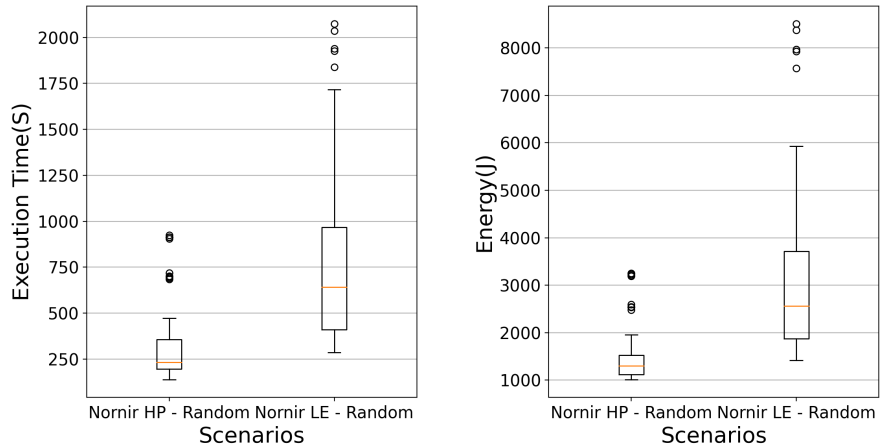
lower than Nornir LE Pareto (=103.85 sec). However, it is a great result - as Nornir LE Pareto intends to produce the least energy consumption. Looking at the energy, Nornir HP - random has 25% of the configurations within 655.6J to 761.29J (lower whisker). Nornir HP Pareto is not the lowest energy consumption (=739.44J), but it is within the values of the lower whisker. Moreover, Nornir LE - random has the least energetic value of 747.30J, which is higher than Nornir LE Pareto (=706.92J).

Figure 6.6b shows a boxplot graph for execution time and energy consumption of Nornir HP and Nornir LE with the 25 random configurations for *Bodytrack*. If the longer part of the box is above the median, the data is skewed right. Considering the execution time for Nornir HP - Random, the box below the median is shorter than the part above the median. That means the configurations whose execution time is lower than the median (=232.96 sec) are more condensed than above the median. Nornir HP Pareto (=224.98 sec) is below Nornir HP - Random median, but not higher than the inferior part of the box(=196.46 sec). Considering Nornir LE, random configurations do not present lower execution time than Nornir LE starting

with Pareto (=231.49 sec). In terms of energy, half of the random configurations used as the start point in Nornir HP have a 407.29J of variability from the median (=1299.93J). Nornir HP Pareto has a higher energy consumption (=1379.75J) compared to the Nornir HP median, but it is lower than the upper part of the box (=1787.04). Concerning Nornir LE - Random, none of the random configurations yielded better results than Nornir LE with Pareto (=1288.34J).



(a) Black-Scholes



(b) Bodytrack.

Figure 6.6: Boxplot graph for 25 random configurations of Nornir HP and LE for Black-Scholes and Bodytrack applications.

6.2.2 Linux Governors Comparison

In this section, we compare Nornir runtime support against the available Linux governors. Since `powersave` governor has the policy of keeping the frequencies at the minimum, it always produces the least performance and the highest energy consumption. Therefore, we decided to discard it in this discussion. We also want to show the relationship between the measured execution time and energy against all possible combinations of cores of each cluster for the applications.

Figure 6.7 and Figure 6.8 show the available Linux governors execution time and energy consumption considering all cores big and LITTLE combinations for **Black-Scholes** and **Bodytrack**, respectively. Notice that as the number of cores of cluster LITTLE increase, the execution time and energy decreases. Nevertheless, it is the core big that has a higher impact on decreasing the overall energy. It is important to remark that an optimal number of cores provide the most energy efficiency for these applications. For **Black-Scholes**, we see that the combination 4+4 (b+L) is the most appropriate number of cores, whereas 4+0 and 4+4 are the two best options for **Bodytrack**. Ultimately, using all eight cores is the optimal number to execute these applications under any available governors.

Nornir tries to find the optimal configuration every 2.5 seconds by performing the MAPEL (See Section 5.4). It can result in unwanted overhead. Thus, the governors have the advantage of running at the optimal number of cores and only regulating the frequency. Therefore, to give a fair comparison with Nornir, we consider the median of each governor execution time and energy for all 24 cores combinations.

In the Figure 6.9, we normalized the execution time and energy consumption by the best-measured value regarding all governors, Nornir HP and LE with Pareto for **Black-Scholes** and **Bodytrack**. Considering **Black-Scholes**, Nornir HP is the least execution time, whereas the governors take up to $1.79\times$ more time than Nornir HP. Moreover, Nornir LE is the least energy consumption, whereas the governors consumed up to $1.62\times$ more than Nornir LE. Considering **Bodytrack**, Nornir HP and LE have the least execution time and energy, respectively. The governors are up to $1.82\times$ more time than Nornir HP and up to $1.45\times$ more energy consumption than Nornir LE. Ultimately, considering all combinations of cores, Nornir HP and LE with Pareto has higher performance and lower energy consumption than Linux governors.

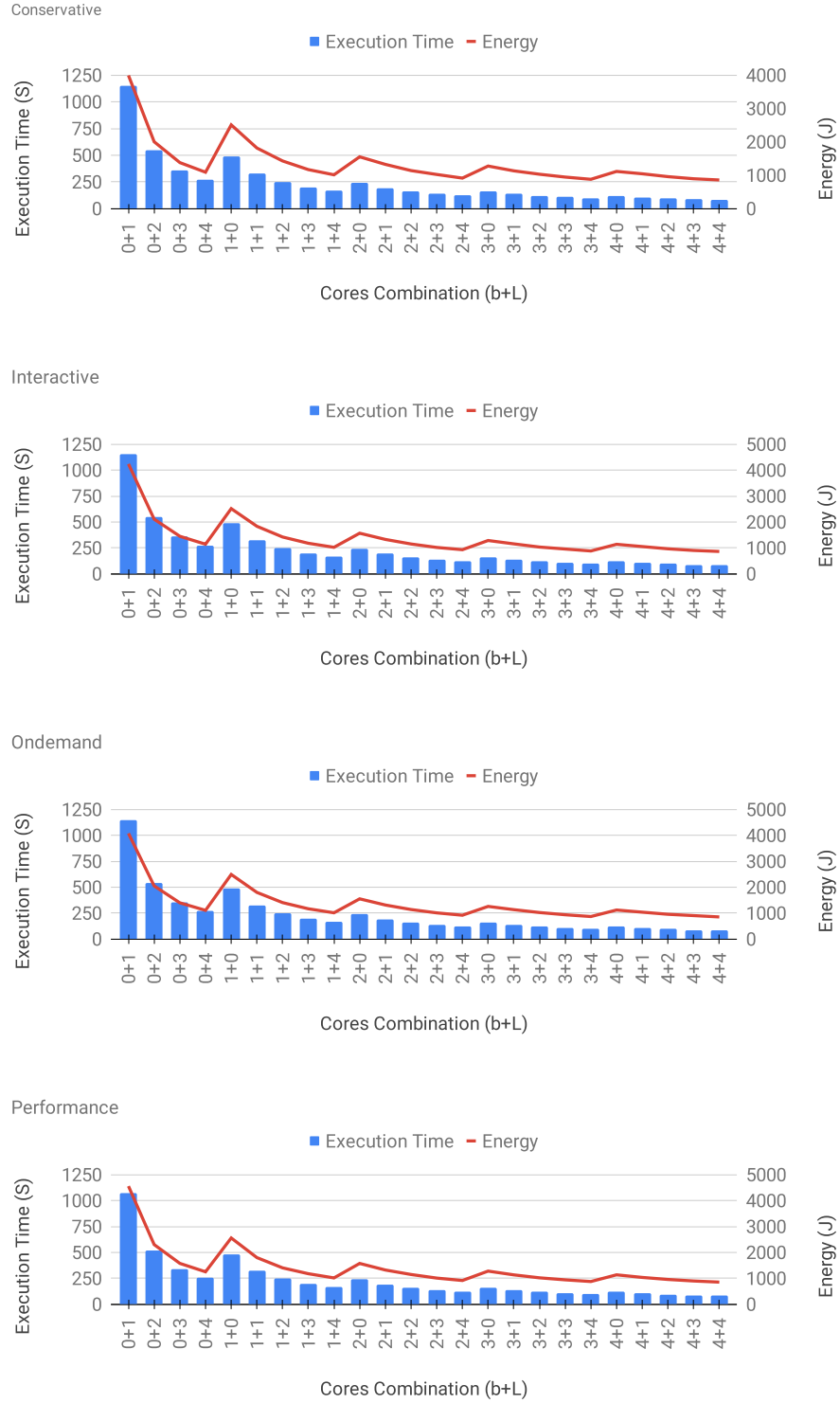


Figure 6.7: The available Linux governors' execution time and energy consumption considering all cores big and LITTLE combination for Black-Scholes.

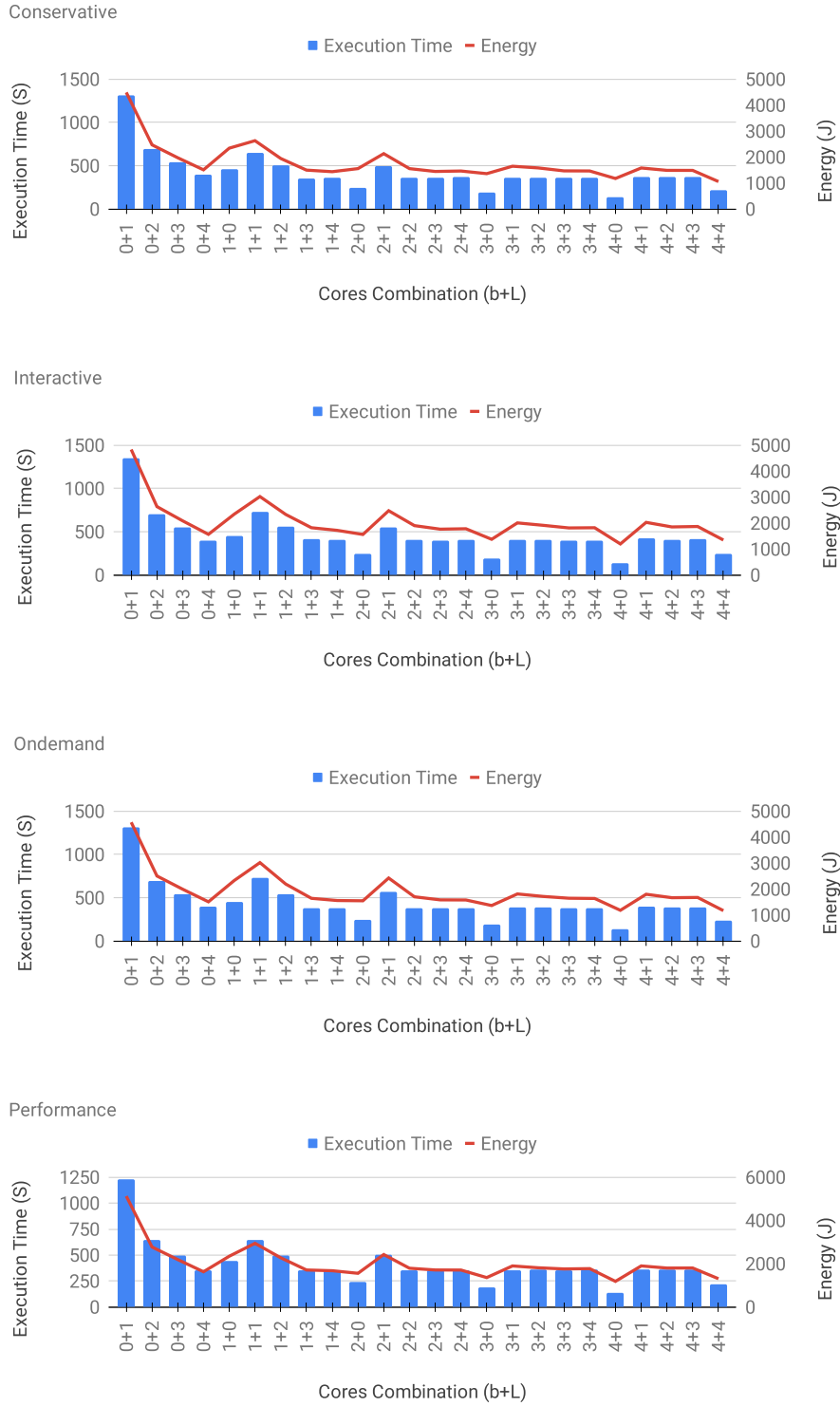


Figure 6.8: The available Linux governors' execution time and energy consumption considering all cores big and LITTLE combination for Bodytrack.

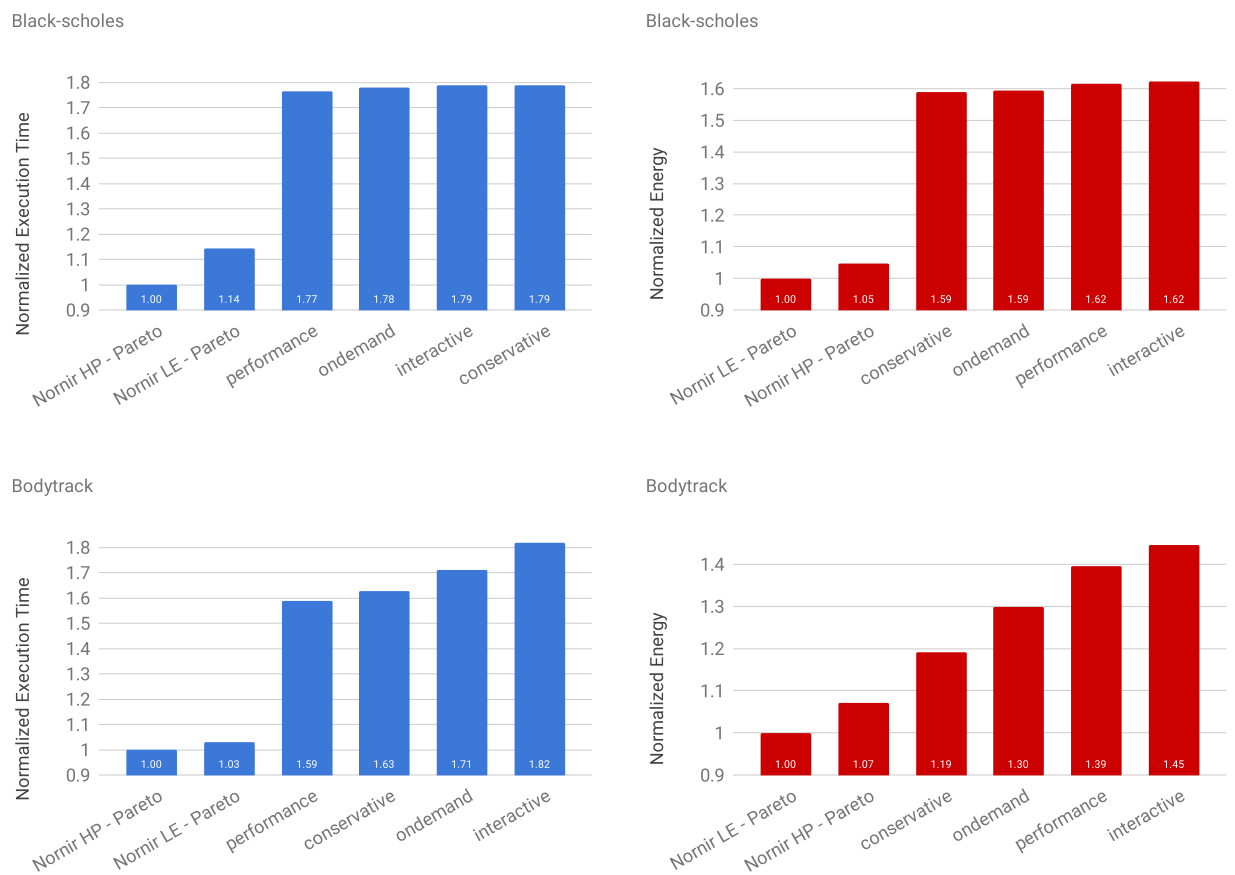


Figure 6.9: Normalized execution time and energy consumption by the respective least measured value.

Part IV

Conclusions and Future Research Directions

Chapter 7

Conclusion

This thesis approached the importance and impact of adequately choosing a configuration within the vast performance-energy trade-offs diversity of operating points for given parallel software with dynamically balanced workload. More specifically, we addressed the problem of finding HMP configurations whose performance and energy are optimal for a given parallel software running on a single-ISA clustered processor. We divided the research into two parts: developing an offline methodology capable of estimating the Pareto-optimal configurations; another, coupling this methodology with an online platform for dynamically searching for optimal configuration throughout an application's execution.

In Chapter 3, we detailed the proposed methodology, devised an analytical low-overhead straightforward performance model for a given multi-thread application, and an analytical application-agnostic power model for a specific two-cluster HMP system. We only need execution time and power measurements from a few configuration space samples to fit these models. The energy model combines these models to assess all available configurations to predict an application's energy consumption. The Pareto frontier uses the provided offline performance and energy models to select the optimal performance-energy trade-off from all available options. We validated our methodology on an ODROID-XU3 board to fit our models and validate the Pareto frontier configurations. Moreover, we compared the measured Pareto frontier with the performance, powersave and ondemand Linux governors. Our approach achieved a reduction of the configuration search space of approximately 99%, significantly decreasing the number of options available to identify an optimal performance-energy trade-off configuration. The average variation of performance and energy concerning all applications are 84.25% and 59.68%, respectively. On average, the performance and energy absolute percentage errors between the measured and the estimated Pareto frontier for all applications are 5.53% and 22.30%, respectively. Also, obtained 13.67% of energy savings regarding the powersave governor with higher or similar performance. Therefore, these results encourage future research in performance- and energy-aware schedulers using our methodology. Furthermore, we can apply our models to predict optimal energy and performance trade-offs.

Considering the previous parallel application characterization acquired from our methodology, we investigated the use of the estimated Pareto frontier to improve

the search for optimal performance-energy trade-offs for a given parallel application at runtime. For that, we developed a hybrid approach between our offline methodology and Nornir runtime support. Nornir is a framework capable of selecting suitable configurations at runtime, imposing power and performance constraints for parallel applications. In Chapter 5, we explained the extensions implemented to the offline methodology to improve accuracy and adapt to specific Nornir’s characteristics. The power model has been improved, taking into account the non-linearity of the voltage and frequency relationship, unused cores disabling, and an extra parameter to absorb the power consumption of the entire device regardless of the selected configurations. During the performance and power collection phase, we measured the application’s ROI instead of the entire execution of the application. The intention is to minimize the influence of the multiple phases and adapt to the Nornir’s operations, which must be within the main application loop. After data collection, the remaining methodology for selecting the Pareto frontier among the configuration space is maintained. Then, the Pareto frontier indicates the performance and the whole energy consumption of the embedded system within the application’s ROI.

These Pareto-optimal configurations are used as starting points for the Nornir platform, which implements the Nelder-Mead search algorithm for HMP. The algorithm searches for the best configuration that meets the performance and energy prerequisites at the application runtime. We validated the Pareto-optimal configurations and the online search algorithm in real hardware with PARSEC **BlackScholes** and **Bodytrack** applications. We found that Nornir execution overhead causes a slight disadvantage compared to the fixed Pareto points during the applications’ ROI. The results also showed that Nornir using the Pareto configurations as starting guess achieved better results than random guesses. As we have shown in Chapter 6, it is inappropriate to compare Nornir runtime support against the DVFS governors since they do not change the number of cores. Therefore, we calculated the median of both performance and energy consumption of all possible cores combinations for each Linux governor. In this context, Nornir starting with Pareto configurations has higher performance and least energy consumption overall DVFS governors. Furthermore, we consider promising results to investigate this hybrid strategy further to find the optimal performance and energy trade-offs at runtime.

We believe that providing some open source tools will enable further research in the future. For this reason, as a contribution to this thesis, we developed the IPerfEcT tool that can collect the power and performance of the entire application or a region of interest chosen by the researcher. Using the model computation scripts, the researcher can choose from the different power models presented in this thesis, fit the perf along with the performance model or not, and finally generate the Pareto curve. The power model can vary among the options: use the table of v-f information; switch off unused cores, or include the theta parameter. The tool also allows automatic data collection with Nornir, governors, and fixed configurations. It is compatible with ODROID-XU3 and Nvidia Jetson TX2 and can be easily extended to other Linux-based platforms.

Chapter 8

Future Work

The following list contains some potential future research directions to extend or some improvements:

- Strengthen the hybrid strategy experiments.

The experiments presented in Chapter 6 involving Nornir are preliminary. Further applications are needed to have more solid results. We want to add at least the same applications that were used in Chapter 4. Moreover, we want to add a deep learning application, given the importance in the mobile world and image classification applications. Another interesting experiment to be yet done is adding some spurious processes that would randomly add resources consumptions throughout the target application execution. Thus, we could take advantage of Nornir’s auto-adaptation feature to adjust resource usage to accommodate performance, energy consumption, and workload variations within desired specifications.

- Implement a multi-phase strategy.

The parallel fraction can vary dynamically in workloads, and it is generally unknown at runtime without application-level instrumentation. The parallel workload can be a significant indicator for the runtime allocation of parallel resources. For instance, if the parallel fraction is below a certain level, there is insufficient motivation to use too many cores. Therefore, a runtime algorithm using heuristics or machine learning can dynamically choose the best configuration according to the running application phase change. We could extend our offline methodology to generate the Pareto-optimal configurations for each application’s phase providing the most performance-energy efficiency configurations. Then, a runtime algorithm can choose the best configuration among the Pareto Frontier adapted to a desired performance and energy consumption for a given application. Furthermore, we need to tailor the runtime algorithm using the minimum number of architecture counters to minimize runtime overhead and hardware complexity, mainly on embedded systems where these counters are limited.

- Extend the Methodology and Devise models towards CPU-GPU.

We consider it extremely important to explore the heterogeneity of the MPSoC, which integrates many processing elements. GPU is an essential component as several mobile applications, such as games, critically depend on the GPU for rendering; or any other image, video processing applications used to classify objects. We can investigate how to devise analytical models for predicting the performance and energy consumption of CPU-GPU relation, considering the frequency of the GPU cores and the CPU configurations (number of cores and frequency of each cluster) as input parameters. As we explore the heterogeneity by adding more knobs as GPU frequency, we exponentially increase the search space. It makes the problem more complex, adding more overhead for an offline methodology but mainly for online algorithms. However, increasing the number of knobs usually means greater control over performance and power consumption, leading to better performance and energy trade-offs.

- Address multiple application scenario.

We can tackle multiple-application scenario, which is more challenging but closer to real-life use cases such as mobile devices and some Drones use-cases. We can extend the methodology to find configurations that provide the optimal performance-energy trade-offs for multiple applications running simultaneously on single-ISA HMP. The model validation step can be adjusted to ensure that each application has an exclusive set of resources. Thus, avoiding performance interferences between applications sharing system resources. This way, we would have a larger configuration space considering all possible combinations for two applications running simultaneously compared to a single application. In the end, the Pareto frontier step selects all modelled configurations whose models' outcomes yield the optimal performance and energy consumption trade-offs regarding multiple applications.

Bibliography

Aalsaud, Ali, Rishad Shafik, Ashur Rafiev, Fie Xia, Sheng Yang & Alex Yakovlev (2016), Power-aware performance adaptation of concurrent applications in heterogeneous many-core systems, *in* ‘Proceedings of the 2016 International Symposium on Low Power Electronics and Design’, ISLPED ’16, ACM, New York, NY, USA, pp. 368–373.

URL: <http://doi.acm.org/10.1145/2934583.2934612>

Al-hayanni, Mohammed A Noaman, Ashur Rafiev, Fei Xia, Rishad Shafik, Alexander Romanovsky & Alex Yakovlev (2020), ‘PARMA: Parallelization-Aware Run-Time Management for Energy-Efficient Many-Core Systems’, *IEEE Transactions on Computers* **69**(10), 1507–1518.

URL: <https://ieeexplore.ieee.org/document/9007357/>

Alonso, Pedro, Manuel F. Dolz, Rafael Mayo & Enrique S. Quintana-Ortí (2014), ‘Modeling power and energy of the task-parallel cholesky factorization on multicore processors’, *Computer Science - R&D* **29**(2), 105–112.

URL: <https://doi.org/10.1007/s00450-012-0227-z>

Amdahl, G. M. (1967), ‘Validity of the single processor approach to achieving large scale computing capabilities’, *Proc. AFIPS 1967 Spring Joint Computer Conf. 30 (April)*, Atlantic City, N. J. pp. 483–485.

Andrae, Anders S. G. & Tomas Edler (2015), ‘On global electricity usage of communication technology: Trends to 2030’, *Challenges* **6**(1), 117–157.

URL: <https://www.mdpi.com/2078-1547/6/1/117>

Apple (2017), ‘iPhone 8 and iPhone 8 Plus: A new generation of iPhone’. Accessed: 2019-02-28.

URL: <https://www.apple.com/newsroom/2017/09/iphone-8-and-iphone-8-plus-a-new-generation-of-iphone/>

ARM (2011), ‘Cortex-A15 Technical Reference Manual’. Accessed: 2021-02-10.

URL: <https://developer.arm.com/documentation/ddi0438/c/>

ARM (2012), ‘Arm big.LITTLE’. Accessed: 2021-02-10.

URL: <https://www.arm.com/why-arm/technologies/big-little>

ARM (2013a), ‘Cortex-A7 MPCore - Technical Reference Manual’. Accessed: 2021-02-10.

URL: <http://infocenter.arm.com>

- ARM (2013b), ‘Cortex-A7 – ARM Developer’. Accessed: 2021-02-10.
URL: <https://developer.arm.com/products/processors/cortex-a/cortex-a7>
- ARM (2014), ‘CoreLink Cache Coherent Interconnect Family – Arm Developer’. Accessed: 2021-01-06.
URL: <https://developer.arm.com/ip-products/system-ip/corelink-interconnect/corelink-cache-coherent-interconnect-family>
- Bailey, Peter E., Aniruddha Marathe, David K. Lowenthal, Barry Rountree & Martin Schulz (2015), Finding the limits of power-constrained application performance, in ‘Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis’, SC ’15, ACM, New York, NY, USA, pp. 79:1–79:12.
URL: <http://doi.acm.org/10.1145/2807591.2807637>
- Balaprakash, Prasanna, Ananta Tiwari & Stefan M. Wild (2014), ‘Multi objective optimization of HPC kernels for performance, power, and energy’, *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* **8551**, 239–260.
- Barros, C.A., L.F.Q. Silveira, C.A. Valderrama & S. Xavier de Souza (2015), ‘Optimal processor dynamic-energy reduction for parallel workloads on heterogeneous multi-core architectures’, *Microprocessors and Microsystems* **39**(6), 418–425.
URL: <https://www.sciencedirect.com/science/article/pii/S0141933115000617>
- Barros, Carlos Avelino (2016), Redução do Consumo Energético de Aplicações Paralelas Eficientes em Arquiteturas Multicore, PhD thesis, Universidade Federal do Rio Grande do Norte.
- Basireddy, Karunakar R., Amit Kumar Singh, Bashir M. Al-Hashimi & Geoff V. Merrett (2020), ‘AdaMD: Adaptive Mapping and DVFS for Energy-Efficient Heterogeneous Multicores’, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **39**(10), 2206–2217.
URL: <https://ieeexplore.ieee.org/document/8796426/>
- Ben Abdallah, Abderazak (2017), ‘Heterogeneous Computing: An Emerging Paradigm of Embedded Systems Design’, *Computational Frameworks: Systems, Models and Applications* pp. 61–93.
- Benini, L., A. Bogliolo & G. De Micheli (2000), ‘A survey of design techniques for system-level dynamic power management’, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* **8**(3), 299–316.
- Bienia, Christian (2011), Benchmarking Modern Multiprocessors, PhD thesis, Princeton University, Princeton, NJ, USA. AAI3445564.

- Borkar, Shekhar (2007), Thousand core chips: A technology perspective, in ‘Proceedings of the 44th Annual Design Automation Conference’, DAC ’07, ACM, New York, NY, USA, pp. 746–749.
- Brodowski, Dominik (2005), ‘Current trends in Linux Kernel Power Management’. Accessed: 2020-02-28.
- Brodowski, Dominik & Nico Golde (2010), ‘CPU frequency and voltage scaling code in the Linux(TM) kernel odroid 3.10’, **7**(December), 1–5.
URL: <https://github.com/hardkernel/linux/blob/odroidxu3-3.10.y/Documentation/cpu-freq/governors.txt>
- Brodowski, Dominik, Nico Golde, Rafael J. Wysocki & Viresh Kumar (2010), ‘CPU frequency and voltage scaling code in the Linux(TM) kernel’, **7**(December), 1–5.
- Brown, Neil (2016), ‘Improvements in CPU frequency management’. Accessed: 2021-02-20.
URL: <https://lwn.net/Articles/682391/>
- Burd, T. D. & R. W. Brodersen (1995), Energy efficient CMOS microprocessor design, in ‘Proceedings of the Twenty-Eighth Annual Hawaii International Conference on System Sciences’, Vol. 1, pp. 288–297 vol.1.
- Cardoso, João M.P., José Gabriel F. Coutinho & Pedro C. Diniz (2017), Chapter 2 - high-performance embedded computing, in J. M.Cardoso, J. G. F.Coutinho & P. C.Diniz, eds, ‘Embedded Computing for High Performance’, Morgan Kaufmann, Boston, pp. 17–56.
URL: <https://www.sciencedirect.com/science/article/pii/B9780128041895000028>
- Chandrakasan, A. P. & R. W. Brodersen (1995), ‘Minimizing power consumption in digital CMOS circuits’, *Proceedings of the IEEE* **83**(4), 498–523.
- Che, Shuai, Jeremy W Sheaffer, Michael Boyer, Lukasz G Szafaryn, Liang Wang & Kevin Skadron (2010), A characterization of the Rodinia benchmark suite with comparison to contemporary CMP workloads, in ‘IEEE International Symposium on Workload Characterization, IISWC’10’.
- Che, Shuai, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W. Sheaffer, Sang Ha Lee & Kevin Skadron (2009), Rodinia: A benchmark suite for heterogeneous computing, in ‘Proceedings of the 2009 IEEE International Symposium on Workload Characterization, IISWC 2009’, IEEE, pp. 44–54.
URL: <http://ieeexplore.ieee.org/document/5306797/>
- Chitlur, Nagabhushan, Ganapati Srinivasa, Scott Hahn, P K Gupta, Dheeraj Reddy, David Koufaty, Paul Brett, Abirami Prabhakaran, Li Zhao, Nelson Ijhi, Suchit Subhaschandra, Sabina Grover, Xiaowei Jiang & Ravi Iyer (2012), QuickIA: Exploring heterogeneous architectures on real prototypes, in ‘IEEE International Symposium on High-Performance Comp Architecture’, IEEE, pp. 1–8.
URL: <http://ieeexplore.ieee.org/document/6169046/>

- Coutinho, D. A. M., K. Georgiou, K. I. Eder, J. Nunez-Yanez & S. Xavier-de-Souza (2019), Performance and energy efficiency trade-offs in single-isa heterogeneous multi-processing for parallel applications, *in* ‘2019 IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC)’, pp. 232–233.
- Coutinho, D. A. M. & S. Xavier-de-Souza (2013), Multicore scalability and efficiency analysis of the standard simplex algorithm, *in* ‘XLV Simpósio Brasileiro de Pesquisa Operacional’, Natal/RN - Brasil.
- Dave, Kinjal (2017), ‘System design considerations for embedded heterogeneous multiprocessing’. Accessed: 2021-06-02.
URL: <https://www.computer.org/publications/tech-news/heterogeneous-system-architecture/system-design-considerations-for-embedded-heterogeneous-multiprocessing>
- De Sensi, Daniele (2016), Predicting performance and power consumption of parallel applications, *in* ‘2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)’, pp. 200–207.
- De Sensi, Daniele, Massimo Torquati & Marco Danelutto (2016), ‘A reconfiguration algorithm for power-aware parallel applications’, *ACM Trans. Archit. Code Optim.* **13**(4).
URL: <https://doi.org/10.1145/3004054>
- De Sensi, Daniele, Tiziano De Matteis & Marco Danelutto (2018), ‘Simplifying self-adaptive and power-aware computing with nornir’, *Future Generation Computer Systems* **87**, 136–151.
URL: <https://www.sciencedirect.com/science/article/pii/S0167739X17326699>
- De Sensi, Daniele, Tiziano De Matteis, Massimo Torquati, Gabriele Mencagli & Marco Danelutto (2017), ‘Bringing parallel patterns out of the corner: The P³ARSEC Benchmark Suite’, *ACM Trans. Archit. Code Optim.* **14**(4).
URL: <https://doi.org/10.1145/3132710>
- Dennard, R.H., F.H. Gaensslen, Hwa-Nien Yu, V.L. Rideout, E. Bassous & A.R. LeBlanc (1974), ‘Design of ion-implanted mosfet’s with very small physical dimensions’, *IEEE Journal of Solid-State Circuits* **9**(5), 256–268.
- Derr, Simon (2004), ‘Cpusets’. Accessed: 2019-04-15.
URL: <https://www.kernel.org/doc/Documentation/cgroup-v1/cpusets.txt/>
- De Sensi, Daniele, Massimo Torquati & Marco Danelutto (2017), ‘Mammut: High-level management of system knobs and sensors’, *SoftwareX* **6**, 150–154.
URL: <https://www.sciencedirect.com/science/article/pii/S2352711017300225>
- Donyanavard, B., T. Mück, S. Sarma & N. Dutt (2016), Sparta: Runtime task allocation for energy efficient heterogeneous manycores, *in* ‘2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)’, pp. 1–10.

- Eder, Kerstin, John P. Gallagher, Pedro López-García, Henk Muller, Zorana Banković, Kyriakos Georgiou, Rémy Haemmerlé, Manuel V. Hermenegildo, Bishoksan Kafle, Steve Kerrison, Maja Kirkeby, Maximiliano Klemen, Xueliang Li, Umer Liqat, Jeremy Morse, Morten Rhiger & Mads Rosendahl (2016), ‘EN-TRA: Whole-systems energy transparency’, *Microprocessors and Microsystems* **47**, 278–286.
- Endrei, Mark, Chao Jin, Minh Ngoc Dinh, David Abramson, Heidi Poxon, Luiz DeRose & Bronis R. de Supinski (2017), A bottleneck-centric tuning policy for optimizing energy in parallel programs, *in* ‘Parallel Computing is Everywhere, Proceedings of the International Conference on Parallel Computing, ParCo 2017, 12-15 September 2017, Bologna, Italy’, pp. 265–276.
URL: <https://doi.org/10.3233/978-1-61499-843-3-265>
- Endrei, Mark, Chao Jin, Minh Ngoc Dinh, David Abramson, Heidi Poxon, Luiz DeRose & Bronis R. de Supinski (2018), Energy efficiency modeling of parallel applications, *in* ‘Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis’, SC ’18, IEEE Press, Piscataway, NJ, USA, pp. 17:1–17:13.
URL: <http://dl.acm.org/citation.cfm?id=3291656.3291679>
- Esmailzadeh, H., E. Blem, R. St. Amant, K. Sankaralingam & D. Burger (2012), ‘Dark Silicon and the End of Multicore Scaling’, *IEEE Micro* **32**(3), 122–134.
URL: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6175879>
- Fagas, Giorgos, John P. Gallagher, Luca Gammaitoni & Douglas J. Paul (2017), Energy challenges for ICT, *in* G.Fagas, L.Gammaitoni, J. P.Gallagher & D. J.Paul, eds, ‘ICT - Energy Concepts for Energy Efficiency and Sustainability’, IntechOpen, Rijeka, chapter 1.
URL: <https://doi.org/10.5772/66678>
- Ferreboeuf, Hugues, Françoise Berthoud, Philippe Bihouix, Pierre Fabre, Daniel Kaplan, Laurent Lefèvre, Alexandre Monnin, Olivier Ridoux, Samuli Vaija, Marc Vautier, Xavier Verne & Alain Ducass (2019), Lean ICT-towards digital sobriety., Technical report, The Shift Project. Accessed: 2021-05-25.
URL: https://theshiftproject.org/wp-content/uploads/2019/03/Lean-ICT-Report_The-Shift-Project_2019.pdf
- Flynn, David (2012), An ARM Perspective on Addressing Low-Power Energy-Efficient SoC Designs, *in* ‘Proceedings of the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design’, ISLPED ’12, Association for Computing Machinery, New York, NY, USA, p. 73–78.
URL: <https://doi.org/10.1145/2333660.2333680>
- Furtunato, Alex F. A., Kyriakos Georgiou, Kerstin Eder & Samuel Xavier-De-Souza (2020), ‘When parallel speedups hit the memory wall’, *IEEE Access* **8**, 79225–79238.

- Garey, Michael R. & David S. Johnson (1990), *Computers and Intractability; A Guide to the Theory of NP-Completeness*, W. H. Freeman & Co., USA.
- Georgiou, Kyriakos, Samuel Xavier-de Souza & Kerstin Eder (2017), ‘The IoT energy challenge: A software perspective’, *IEEE Embedded Systems Letters* pp. 1–1.
URL: <http://ieeexplore.ieee.org/document/8012513/>
- Goodrum, Matthew A., Michael J. Trotter, Alla Aksel, Scott T. Acton & Kevin Skadron (2012), Parallelization of particle filter algorithms, *in* A. L. Varbanescu, A. Molnos & R. van Nieuwpoort, eds, ‘Computer Architecture’, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 139–149.
- Govind Wathan (2017), ‘Where does big.LITTLE fit in the world of DynamIQ? - Processors blog - Processors - Arm Community’. Accessed: 2021-02-10.
URL: https://community.arm.com/developer/ip-products/processors/b/processors-ip-blog/posts/where-does-big-little-fit-in-the-world-of-dynamiq?_ga=2.39798691.1008928676.1612810453-47051251.1609964180
- Greenhalgh, P. (2011), Big.LITTLE processing with ARM Cortex-A15 & Cortex-A7, Technical Report September 2011, ARM. Accessed: 2021-02-10.
URL: <https://web.archive.org/>
- Gupta, Ujjwal, Chetan Arvind Patil, Ganapati Bhat, Prabhat Mishra & Umit Y. Ogras (2017), ‘DyPO: Dynamic Pareto-Optimal Configuration Selection for Heterogeneous’, *ACM Transactions on Embedded Computing Systems* **16**(5s), 1–20.
- Gustafson, John L. (1988), ‘Reevaluating amdahl’s law’, *Communications of the ACM* **31**, 532–533.
- Hardkernel (2014), ‘en:xu3_hardware [ODROID Wiki]’. Accessed: 2019-04-12.
URL: <https://www.hardkernel.com/shop/odroid-xu3/>
- Hardkernel (2017), ‘SmartPower2 with 15V/4A – ODROID’. Accessed: 2021-01-06.
URL: <https://www.hardkernel.com/shop/smartpower2-with-15v-4a/>
- Herbert, Sebastian & Diana Marculescu (2007), Analysis of dynamic voltage/frequency scaling in chip-multiprocessors, *in* ‘Proceedings of the 2007 international symposium on Low power electronics and design (ISLPED ’07)’, pp. 38–43.
- Hill, Mark D. & Michael R. Marty (2008), ‘Amdahl’s law in the multicore era’, *Computer* **41**(7), 33–38.
- Huang, Wei, Karthick Rajamani, Mircea R. Stan & Kevin Skadron (2011), ‘Scaling with design constraints: Predicting the future of big chips’, *IEEE Micro* **31**(4), 16–29.

- Jin, Chao, Bronis R de Supinski, David Abramson, Heidi Poxon, Luiz DeRose, Minh Ngoc Dinh, Mark Endrei & Elizabeth R Jessup (2017), ‘A survey on software methods to improve the energy efficiency of parallel computing’, *The International Journal of High Performance Computing Applications* **31**(6), 517–549.
URL: <https://doi.org/10.1177/1094342016665471>
- Kale, Vivek (2018), ‘Openmp dynamic schedule’. Accessed: 2019-10-28.
URL: https://www.openmp.org/wp-content/uploads/SC17-Kale-LoopSchedforOMP_BoothTalk.pdf
- Kephart, J. O. & D. M. Chess (2003), ‘The vision of autonomic computing’, *Computer* **36**(1), 41–50.
- Kerrisk, Michael (2018), ‘sysfs(5) — Linux manual page’. Accessed: 2021-02-20.
URL: <https://man7.org/linux/man-pages/man5/sysfs.5.html>
- Kim, Myungsun, Kibeom Kim, James R. Geraci & Seongsoo Hong (2014), ‘Utilization-aware load balancing for the energy efficient operation of the big.LITTLE processor’, *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2014* pp. 1–4.
URL: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6800437>
- Kim, Nam Sung, Todd Austin, David Blaauw, Trevor Mudge, Krisztián Flautner, Jie S. Hu, Mary Jane Irwin, Mahmut Kandemir & Vijaykrishnan Narayanan (2003), ‘Leakage current: Moore’s law meets static power’, *Computer* **36**(12), 68–75.
URL: <https://doi.org/10.1109/MC.2003.1250885>
- King, Colin Ian (2014), ‘Stress-ng github’. Accessed: 2019-04-15.
URL: <https://github.com/ColinIanKing/stress-ng/>
- Koufaty, David, Dheeraj Reddy & Scott Hahn (2010), Bias scheduling in heterogeneous multi-core architectures, in ‘Proceedings of the 5th European conference on Computer systems - EuroSys ’10’, ACM Press, New York, New York, USA, p. 125.
URL: <http://portal.acm.org/citation.cfm?doid=1755913.1755928>
- Kumar, R., D.M. Tullsen, P. Ranganathan, N.P. Jouppi & K.I. Farkas (2004), Single-ISA heterogeneous multi-core architectures for multithreaded workload performance, in ‘Proceedings. 31st Annual International Symposium on Computer Architecture, 2004.’, pp. 64–75.
- Kundan, Shivam, Ourania Spantidi & Iraklis Anagnostopoulos (2021), ‘On-line frequency-based performance and power estimation for clustered multi-processor systems’, *Computers and Electrical Engineering* **90**(January), 106971.
URL: <https://doi.org/10.1016/j.compeleceng.2021.106971>

- Lanier, Travis (2010), ‘Exploring the Design of the Cortex-A15 Processor : ARM’s next generation mobile applications processor’.
- Loghin, Dumitrel & Yong Meng Teo (2018), ‘The time and energy efficiency of modern multicore systems’, *Parallel Computing* **86**, 1–10.
URL: <https://doi.org/10.1016/j.parco.2019.04.009>
- Luo, G., B. Guo, Y. Shen, H. Liao & L. Ren (2009), Analysis and optimization of embedded software energy consumption on the source code and algorithm level, in ‘2009 Fourth International Conference on Embedded and Multimedia Computing’.
- Manumachu, Ravindranath Reddy & Alexey Lastovetsky (2018), ‘Bi-Objective Optimization of Data-Parallel Applications on Homogeneous Multicore Clusters for Performance and Energy’, *IEEE Transactions on Computers* **67**(2), 160–177.
- Marculescu, Radu, Umit Y. Ogras, Li-Shiuan Peh, Natalie Enright Jerger & Yatin Hoskote (2009), ‘Outstanding research problems in noc design: System, microarchitecture, and circuit perspectives’, *Trans. Comp.-Aided Des. Integ. Cir. Sys.* **28**(1), 3–21.
URL: <https://doi.org/10.1109/TCAD.2008.2010691>
- Marwedel, Peter, Jürgen Teich, Georgia Kouveli, Iuliana Bacivarov, Lothar Thiele, Soonhoi Ha, Chanhee Lee, Qiang Xu & Lin Huang (2011), Mapping of applications to mpsocs, in ‘Proceedings of the Seventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis’, CODES+ISSS ’11, Association for Computing Machinery, New York, NY, USA, p. 109–118.
URL: <https://doi.org/10.1145/2039370.2039390>
- Menon, Nishanth (2009), ‘Operating Performance Points (OPP) Library — The Linux Kernel documentation’. Accessed: 2021-02-20.
URL: <https://www.kernel.org/doc/html/latest/power/opp.html>
- Mittal, Sparsh (2014), ‘A survey of techniques for improving energy efficiency in embedded computing systems’, *International Journal of Computer Aided Engineering and Technology* **6**(4), 440–459.
- Modarressi, Mehdi & Hamid Sarbazi-Azad (2018), *Topology Specialization for Networks-on-Chip in the Dark Silicon Era*, Vol. 110, 1 edn, Elsevier Inc.
URL: <http://dx.doi.org/10.1016/bs.adcom.2018.03.009>
- Moore, Gordon E. (1998), ‘Cramming more components onto integrated circuits’, *Proceedings of the IEEE* **86**(1), 82–85.

- Morad, T. Y., U. C. Weiser, A. Kolodny, M. Valero & E. Ayguade (2006), ‘Performance, Power Efficiency and Scalability of Asymmetric Cluster Chip Multiprocessors’, *IEEE Computer Architecture Letters* **5**(1), 4–4.
URL: <http://ieeexplore.ieee.org/document/1650136/>
- Morokoff, William J. & Russel E. Caflisch (1994), ‘Quasi-random sequences and their discrepancies’, *SIAM J. Scientific Computing* **15**(6), 1251–1279.
URL: <https://doi.org/10.1137/0915077>
- Muralidhar, Rajeev, Renata Borovica-Gajic & Rajkumar Buyya (2020), ‘Energy Efficient Computing Systems: Architectures, Abstractions and Modeling to Techniques and Standards’, *arXiv* .
URL: <http://arxiv.org/abs/2007.09976>
- Nelder, J. A. & R. Mead (1965), ‘A Simplex Method for Function Minimization’, *The Computer Journal* **7**(4), 308–313.
URL: <https://doi.org/10.1093/comjnl/7.4.308>
- Nguyen, Hubert (2013), ‘What Is ARM .LITTLE?’. Accessed: 2021-02-11.
URL: <https://www.ubergizmo.com/2013/01/what-is-arm-big-little/>
- Nocedal, Jorge & Stephen J. Wright (2006), *Numerical Optimization*, 2 edn, Springer, New York, NY, USA.
- NVIDIA (2016), ‘NVIDIA CUDA ZONE’. Accessed: 2019-10-28.
URL: <https://developer.nvidia.com/cuda-zone>
- OpenMP (2021), ‘Home - OpenMP’. Accessed: 2021-05-04.
URL: <https://www.openmp.org/>
- Oshana, Robert (2009), ‘The keys to success in multicore application development - Embedded.com’. Accessed: 2021-08-21.
URL: <https://www.embedded.com/the-keys-to-success-in-multicore-application-development/>
- Özisikyilmaz, Berkin, Ramanathan Narayanan, Joseph Zambreno, Gokhan Memik & Alok Choudhary (2006), An architectural characterization study of data mining and bioinformatics workloads, in ‘Proceedings of the 2006 IEEE International Symposium on Workload Characterization, IISWC - 2006’, pp. 61–70.
- Pallipadi, Asv & Alexey Starikovskiy (2006), ‘The ondemand governor: past, present and future’, *Proceedings of the Linux Symposium* pp. 215–230.
URL: <http://www2.ic.uff.br/>
- Patterson, David A. & John L. Hennessy (2014), *Computer Organization and Design : The Hardware/Software Interface*, Elsevier Science.

- Pham, D.C., T. Aipperspach, D. Boerstler, M. Bolliger, R. Chaudhry, D. Cox, P. Harvey, P.M. Harvey, H.P. Hofstee, C. Johns, J. Kahle, A. Kameyama, J. Keaty, Y. Masubuchi, M. Pham, J. Pille, S. Posluszny, M. Riley, D.L. Stasiak, M. Suzuki, O. Takahashi, J. Warnock, S. Weitzel, D. Wendel & K. Yazawa (2006), ‘Overview of the architecture, circuit design, and physical implementation of a first-generation Cell processor’, *IEEE Journal of Solid-State Circuits* **41**(1), 179–196.
- Pollack, F. J. (1999), New microarchitecture challenges in the coming generations of CMOS process technologies, in ‘2012 45th Annual IEEE/ACM International Symposium on Microarchitecture’, IEEE Computer Society, Los Alamitos, CA, USA, p. 2.
URL: <https://doi.ieeecomputersociety.org/10.1109/MICRO.1999.10004>
- Pouwelse, Johan, Koen Langendoen & Henk J. Sips (2003), ‘Application-Directed Voltage Scaling’, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* **11**(5), 812–826.
- Qualcomm (2019), Qualcomm Snapdragon 810 processor, Technical report, Qualcomm. Accessed: 2019-02-28.
URL: <https://www.qualcomm.com/media/documents/files/snapdragon-810-processor-product-brief.pdf>
- Ramanathan, R. M. (2006), Intel® Multi-Core Processors, Technical report, Intel Corporation. Accessed: 2019-02-20.
URL: <https://www.all-electronics.de/wp-content/uploads/migrated/document/112084/139-intel-multicore.pdf>
- Reddy, Basireddy Karunakar (2019), Runtime Energy Management of Concurrent Applications for Multi-core Platforms, PhD thesis, University of Southampton.
- Rupp, Karl (2018), ‘42 Years of Microprocessor Trend Data’. Accessed: 2021-01-25.
URL: <https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/>
- Saez, Juan Carlos, Adrian Pousa, Fernando Castro, Daniel Chaver & Manuel Prieto-Matias (2017), ‘Towards completely fair scheduling on asymmetric single-ISA multicore processors’, *Journal of Parallel and Distributed Computing* **102**, 115–131.
URL: <http://linkinghub.elsevier.com/retrieve/pii/S0743731516301873>
- Samsung (2014), ‘The big.little octa-core mobile processor with hmp solution’. Accessed: 2021-01-06.
URL: <https://www.samsung.com/semiconductor/>
- Samsung (2018), ‘Exynos 7 octa 7420 processor: Specs, features | samsung exynos’. Accessed: 2021-04-02.
URL: <https://www.samsung.com/semiconductor>

- Samsung (2019), Samsung Exynos 7, Technical report, Samsung. Accessed: 2019-02-28.
URL: <https://www.samsung.com/semiconductor/processor/mobile-processor/>
- Sarma, Santanu, T. Muck, Luis A. D. Bathen, N. Dutt & A. Nicolau (2015), ‘Smart-Balance : A sensing-driven Linux load balancer for energy efficiency of heterogeneous MPSoCs’, *Design Automation Conference (DAC) 2015* pp. 1–6.
URL: <http://dl.acm.org/citation.cfm?doid=2744769.2744911>
- Sawalha, Lina, Sonya Wolff, Monte P. Tull & Ronald D. Barnes (2011), ‘Phase-guided scheduling on single-ISA heterogeneous multicore processors’, *Proceedings - 2011 14th Euromicro Conference on Digital System Design: Architectures, Methods and Tools, DSD 2011* pp. 736–745.
- Sen, Rathijit & David A Wood (2017), ‘Pareto Governors for Energy-Optimal Computing’, *ACM Trans. Archit. Code Optim.* **14**(1), 6:1—6:25.
URL: <http://doi.acm.org/10.1145/3046682>
- Seo, Youngbin, Jeongki Kim & Euseong Seo (2012), ‘Effectiveness analysis of DVFS and DPM in mobile devices’, *Journal of Computer Science and Technology* **27**(4), 781–790.
- Shamsa, Elham, Anil Kanduri, Pasi Liljeberg & Amir M. Rahmani (2021), ‘Concurrent Application Bias Scheduling for Energy Efficiency of Heterogeneous Multi-Core platforms’, *IEEE Transactions on Computers* **9340**(c), 1–13.
- Shyamkumar (2019), ‘Dynamic IQ In Mobiles.’. Accessed: 2021-08-19.
URL: <https://medium.com/@shyamambady/dynamic-iq-in-mobiles-b3ba65dda1d1>
- Singer, Saša & John Nelder (2009), ‘Nelder-Mead algorithm’. Accessed: 2021-07-15.
URL: http://www.scholarpedia.org/article/Nelder-Mead_algorithm
- Singh, A. K., M. Shafique, A. Kumar & J. Henkel (2013), Mapping on multi/many-core systems: Survey of current and emerging trends, in ‘2013 50th ACM/E-DAC/IEEE Design Automation Conference (DAC)’, pp. 1–10.
- Singh, Amit Kumar (2013), Run-Time Mapping Techniques for Noc-Based Heterogeneous Mpsoc Platforms, PhD thesis, School of Computer Engineering A.
- Suite, Phoronix Test (2021), ‘Open-source, automated benchmarking’. Accessed: 2019-04-17.
URL: <https://www.phoronix-test-suite.com/>
- Suleiman, D, M Ibrahim & I Hamarash (2005), ‘Dynamic voltage frequency scaling (DVFS) for microprocessors power and energy reduction’, *4th International Conference on ...* pp. 5–9.
URL: http://www.emo.org.tr/ekler/035226640b6b89f_ek.pdf

- Sutter, Herb (2005), ‘The free lunch is over: A fundamental turn toward concurrency in software’, *Dr. Dobbs’s Journal* **30**(3), 202–210.
- Tan, Ying, Wei Liu & Qinru Qiu (2009), Adaptive power management using reinforcement learning, *in* ‘IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers, ICCAD’, pp. 461–467.
- Tanenbaum, Andrew S. & Todd Austin (2005), *Structured Computer Organization*, 5 edn, Pearson.
- Tang, Adrian, Simha Sethumadhavan & Salvatore Stolfo (2017), CLKSCREW: Exposing the perils of security-oblivious energy management, *in* ‘26th USENIX Security Symposium (USENIX Security 17)’, USENIX Association, Vancouver, BC, pp. 1057–1074.
URL: <https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/tang>
- Team, OpenCV (2021), ‘OpenCL’. Accessed: 2019-10-28.
URL: <https://opencv.org/opencv/>
- Triggs, Robert (2017), ‘Everything you need to know about ARM’s DynamIQ’. Accessed: 2021-08-19.
URL: <https://www.androidauthority.com/arm-dynamiq-need-to-know-770349/>
- Truong, Dean N, Wayne H Cheng, Tinoosh Mohsenin, Zhiyi Yu, Anthony T Jacobson, Gouri Landge, Michael J Meeuwsen, Christine Watnik, Anh T Tran, Zhibin Xiao, Eric W Work, Jeremy W Webb, Paul V Mejia & Bevan M Baas (2009), A 167-processor computational platform in 65 nm CMOS, *in* ‘IEEE Journal of Solid-State Circuits’, Vol. 44, pp. 1130–1144.
- Tzilis, Stavros, Pedro Trancoso & Ioannis Sourdis (2019), ‘Energy-Efficient Runtime Management of Heterogeneous Multicores using Online Projection’, *ACM Transactions on Architecture and Code Optimization* **15**(4), 1–26.
- Usman, S., S. U. Khan & S. Khan (2013), A comparative study of voltage/frequency scaling in noc, *in* ‘IEEE International Conference on Electro-Information Technology , EIT 2013’, pp. 1–5.
- Vangal, Sriram R., Jason Howard, Gregory Ruhl, Saurabh Dighe, Howard Wilson, James Tschanz, David Finan, Arvind Singh, Tiju Jacob, Shailendra Jain, Vasantha Erraguntla, Clark Roberts, Yatin Hoskote, Nitin Borkar & Shekhar Borkar (2008), ‘An 80-Tile Sub-100-W TeraFLOPS Processor in 65-nm CMOS’, *IEEE Journal of Solid-State Circuits* **43**(1), 29–41.
- Vasilakis, Evangelos, Ioannis Sourdis, Vassilis Papaefstathiou, Antonis Psathakis & Manolis G.H. Katevenis (2017), ‘Modeling energy-performance tradeoffs in ARM big.LITTLE architectures’, *2017 27th International Symposium on Power*

- and Timing Modeling, Optimization and Simulation, PATMOS 2017* **2017-Janua**, 1–8.
- Venkatachalam, Vasanth & Michael Franz (2005), ‘Power reduction techniques for microprocessor systems’, *ACM Computing Surveys* **37**(3), 195–237.
- Wang, Gang, Wenming Li & Xiali Hei (2015), ‘Energy-aware real-time scheduling on Heterogeneous Multi-Processor’, *2015 49th Annual Conference on Information Sciences and Systems, CISS 2015* pp. 1–7.
- Wathan, Govind (2017), ‘Arm DynamIQ Technology for the next era of compute - Processors blog - Processors - Arm Community’. Accessed: 2021-08-19.
URL: <https://community.arm.com/developer/ip-products/processors/b/processors-ip-blog/posts/arm-dynamiq-technology-for-the-next-era-of-compute>
- Wolf, Marilyn (2014), Chapter 5 - Multiprocessor Architectures, *in* M.Wolf, ed., ‘High-Performance Embedded Computing (Second Edition)’, 2 edn, Morgan Kaufmann, Boston, pp. 243–299.
- Woo, Dong Hyuk & Hsien Hsin S. Lee (2008), ‘Extending Amdahl’s law for energy-efficient computing in the many-core era’, *Computer* **41**(12), 24–31.