μSystems Research Group

School of Engineering

Investigation Into Runtime Workload Classification And Management For Energy-Efficient Many-Core Systems

Ali Majeed M. Aalsaud

Technical Report Series

NCL-EEE-MICRO-TR-2019-215

August 2019

Contact: a.m.m.aalsaud@ncl.ac.uk

Supported by EPSRC (project PRiME, grant EP/K034448/1).

NCL-EEE-MICRO-TR-2019-215

μSystems Research Group
School of Engineering
Merz Court
Newcastle University
Newcastle upon Tyne, NE1 7RU, UK

http://async.org.uk/

Investigation Into Runtime Workload Classification And Management For Energy-Efficient Many-Core Systems

Ali Majeed M. Aalsaud

August 2019

INVESTIGATION INTO RUNTIME WORKLOAD CLASSIFICATION AND MANAGEMENT FOR ENERGY-EFFICIENT MANY-CORE SYSTEMS

Ali Majeed M. Aalsaud

A Thesis Submitted for the Degree of
Doctor of Philosophy at Newcastle University

School of Electrical and Electronic Engineering
Faculty of Science, Agriculture and Engineering

August 2019

DECLARATION

I hereby declare that this thesis is my own work and effort and that it has not been submitted anywhere for any award. Where other sources of information have been used, they have been acknowledged.

Newcastle upon Tyne August 2019

Ali Aalsaud

CERTIFICATE OF APPROVAL

I confirm that, to the best of my knowledge, this thesis is from the student's own work and effort, and all other sources of information used have been acknowledged. This thesis has been submitted with my approval.

ALEX YAKOVLEV
FEI XIA
RISHAD SHAFIK

To my greatest supporters that is my wonderful parents, my beloved wife, Nouranse, my lovely daughters, and my lovely son Elia.
— Ali

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my supervisors Prof. Alex Yakovlev, Dr. Fei Xia and Dr.Rishad Shafik for their support and guidance through my studies. They have been and always will be a source of inspiration and my role model as a researcher.

I am grateful to my sponsor Iraqi Ministry of Higher Education and Scientific Research and Mustansiriyah University for funding my Ph.D. study through their scholarship program.

I am also grateful to my colleagues and friends in the School of Electrical and Electonic Engineering, especially those in Microelectronic Systems research group, at Newcastle University for their assistance and guidance in my studies.

I would like to express my great thanks to (PRiME) project members for their support and useful discussions. I am thankful to Dr. Ashur Rafiev for his supporting through continues discussions, experimental works, and preparing mthread benchmark programs.

Finally, I would like to offer my special regards to all the staff of the school of Electrical and Electronic Engineering in Newcastle university.

Last but not least, I would like to thank my beautiful family for their continuous support and motivation throughout my Ph.D. journey.

ABSTRACT

Recent advances in semiconductor technology have facilitated placing many cores on a single chip. This has led to increases in system architecture complexity with diverse application workloads, with single or multiple applications running concurrently. Determining the most energy-efficient system configuration, i.e. the number of parallel threads, their core allocations and operating frequencies, tailored for each kind of workload and application concurrency scenario is extremely challenging because of the multifaceted relationships between these configuration knobs. Modelling and classifying the workloads can greatly simplify the runtime formulation of these relationships, delivering on energy efficiency, which is the key aim of this thesis.

This thesis is focused on the development of new models for classifying single- and multi-application workloads in relation to how these workloads depend on the aforementioned system configurations. Underpinning these models, we implement and practically validate low-cost runtime methodologies for energy-efficient many-core processors.

This thesis makes four major contributions. Firstly, a comprehensive study is presented that profiles the power consumption and performance characteristics of a multi-threaded many-core system workload, associating power consumption and performance with multiple concurrent applications. These applications are exercised on a heterogeneous platform generating varying system workloads, viz. CPU-intensive or memory-intensive or a combination of both. Fundamental to this study is an investigation of the tradeoffs between inter-application concurrency with performance and power consumption under different system configurations.

The second is a novel model-based runtime optimization approach with the aim of achieving maximized power normalized performance considering dynamic variations of workload and application scenarios. Using real experimental measurements on a heterogeneous platform with a number of PARSEC benchmark applications, we study power normalized performance (in terms of IPS/Watt) underpinned with analytical power and performance models, derived through multivariate linear regression (MLR). Using these models we show that CPU intensive applications behave differently in IPS/Watt compared to memory intensive applications in both sequential and concurrent application scenarios. Furthermore, this approach demonstrate that it is possible to continuously adapt system configuration through a per-application runtime optimization algorithm, which can improve the IPS/Watt compared to the existing approach. Runtime overheads

are at least three cycles for each frequency to determine the control action.

To reduce overheads and complexity, a novel model-free runtime optimization approach with the aim of maximizing power-normalized performance considering dynamic workload variations has been proposed. This approach is the third contribution. This approach is based on workload classification. This classification is supported by analysis of data collected from a comprehensive study investigating the tradeoffs between inter-application concurrency with performance and power under different system configurations. Extensive experiments have been carried out on heterogeneous and homogeneous platforms with synthetic and standard benchmark applications to develop the control policies and validate our approach. These experiments show that workload classification into CPU-intensive and memory-intensive types provides the foundation for scalable energy minimization with low complexity.

The fourth contribution combines workload classification with model based multivariate linear regression. The first approach has been used to reduce the problem complexity, and the second approach has been used for optimization in a reduced decision space using linear-regression. This approach further improves IPS/Watt significantly compared to existing approaches.

This thesis presents a new runtime governor framework which interfaces runtime management algorithms with system monitors and actuators. This tool is not tied down to the specific control algorithms presented in this thesis and therefore has much wider applications.

PUBLICATIONS

List of publications:

1. **Aalsaud, Ali**, Rishad Shafik, Ashur Rafiev, Fie Xia, Sheng Yang, and Alex Yakovlev. "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, pp. 368-373. ACM, 2016.

2. **Aalsaud, Ali**, Ashur Rafiev, Fei Xia, Rishad Shafik, and Alex Yakovlev. "Model-Free Runtime Management of Concurrent Workloads for Energy-Efficient Many-Core Heterogeneous Systems." In 2018 28th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS), pp. 206-213. IEEE, 2018.

3. **Aalsaud, Ali**, Haider Alrudainv, Rishad Shafik, Fei Xia, and Alex Yakovlev. "MEMS-Based Runtime Idle Energy Minimization for Bursty Workloads in Heterogeneous Many-Core Systems." In 2018 28th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS), pp. 198-205. IEEE, 2018.

4. **Aalsaud, Ali**, Ashur Rafiev, Fei Xia, Rishad Shafik, and Alex Yakovlev. "Reduced-Complexity Runtime Management of Concurrent Workloads for Energy-Efficient Many-Core Systems." Journal paper under preparation.

5. Gensh, Rem, **Ali Aalsaud**, Ashur Rafiev, Fei Xia, Alexei Iliasov, Alexander Romanovsky, and Alex Yakovlev. Experiments with odroid-xu3 board. Newcastle University, Computing Science, 2015.

I also contributed in the following works:

1. Xia, Fei, Ashur Rafiev, **Ali Aalsaud**, Mohammed Al-Hayanni, James Davis, Joshua Levine, Andrey Mokhov et al. "Voltage, Throughput, Power, Reliability, and Multicore Scaling." Computer 50, no. 8 (2017): 34-45.

2. Rafiev, Aashur, Fei Xia, Alexei Iliasov, Rem Gensh, **Ali Aalsaud**, Alexander Romanovsky, and Alex Yakovlev. "Order graphs and

cross-layer parametric significance-driven modelling." In Application of Concurrency to System Design (ACSD), 2015 15th International Conference on, pp. 110-119. IEEE, 2015.

3. Rafiev, Ashur, F. Xia, Alexei Iliasov, Rem Gensh, **Ali Aalsaud**, Alexander Romanovsky, and Alexandre Yakovlev. "Selective abstraction and stochastic methods for scalable power modelling of heterogeneous systems." In Specification and Design Languages (FDL), 2016 Forum on, pp. 1-7. IEEE, 2016.

4. Rafiev, Ashur, Andrey Mokhov, Fei Xia, Alexei Iliasov, Rem Gensh, **Ali Aalsaud**, Alexander Romanovsky, and Alex Yakovlev. "Resource-Driven Modelling for Managing Model Fidelity." In Model-Implementation Fidelity in Cyber Physical System Design, pp. 25-55. Springer, Cham, 2017.

5. Romanovsky, Alexander, and Alex Yakovlev. "Power-proportional modelling fidelity Ashur Rafiev, Fei Xia, Alexei Iliasov, Rem Gensh, **Ali Aalsaud**." (2015).

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ACRONYMS

MLR Multivariate linear regression

OL offline

RT runtime

TM task mapping

WLC workload classification

OS operating system dark silicon management (DaSiM)

DRAM Dynamic random-access memory

DVS Dynamic voltage scaling

DFS dynamic frequency scaling

DCT dynamic concurrency throttling

CMP chip-multiprocessor

CMPs chip-multiprocessors

CMOS complementary metal-oxide-semiconductor

P polynomial

NP non-polynomial

SMDP semi-Markov decision process

DaSiM dark silicon management

LR linear regression

DPM dynamic power management

DSP Digital signal processing

DVFS dynamic voltage frequency scaling

RL Reinforcement learning

TD temporal difference

RTM runtime management

OH overheads

FPGA field-programmable gate array

MDP Markov Decision Process

GPU graphics processing unit

LPM local power manager

GPM global power manager

MRPI Memory Reads Per Instruction linear programming (LP)

LP linear programming

ITRS international technology roadmap for semiconductors

PDN power delivery network

SoC system-on-chip

Part I

Thesis Chapters

1

INTRODUCTION

1.1 MOTIVATION

In the last four decades, contemporary computing systems, including embedded and high performance systems, are exhibiting increased complexities in two dimensions. In one dimension, the number and type of computing resources (cores) are growing in hardware platforms, and in the other, an increasing diversity of applications are being executed concurrently on these platforms [17, 81, 125]. In terms of system architecture, an important trend is to increase the number and type of cores on a chip while reducing their individual complexity. Figure 1.1 is from the international technology roadmap for semiconductors (ITRS) which expects that the overall number of cores that can be placed in a single chip will dramatically grow with technology scaling [91].

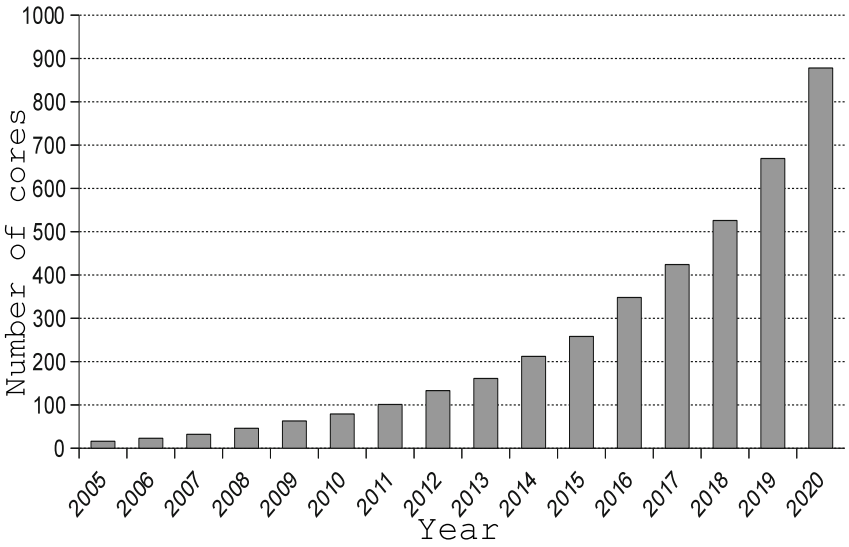


Figure 1.1: Processing element number is projected to scale exponentially according to ITRS [91] .

In the last decades, the clock frequencies of CPU have increased rapidly to meet growing performance needs [70]. Despite the rapidly increase in the recent technology scaling [15], the speed of transistor may not increase at the historic rate due to threshold leakage current, and supply voltage scaling described by Moore's Law [73, 94]. This results in power consumption not scaling with technology feature size reduction [85]. Moreover, according to the ITRS, by the year

2020, the semiconductor technology will see a consistent rise in chip power consumption as opposed to requirement as can be seen in Figure 1.2 [92]. Consequently, power and energy consumption has become a limiting factor to continued technology scaling and performance improvements.

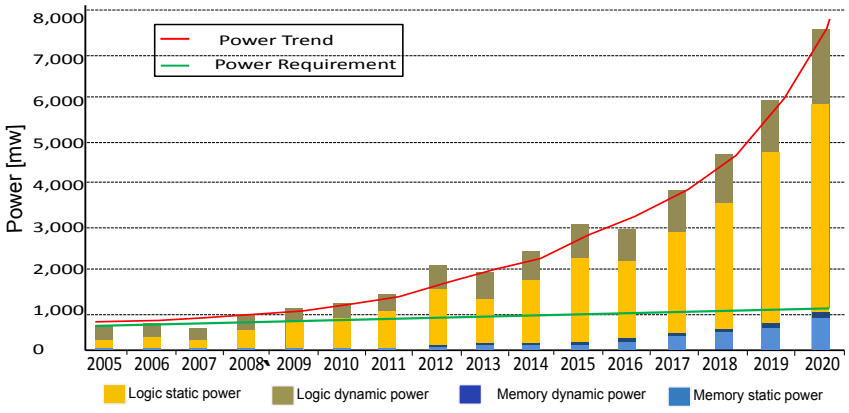


Figure 1.2: Power trends until 2020 (Source: International Technology Roadmap for Semiconductors [92].

In terms of executing multiple applications, parallelization has been used to maintain a reasonable balance between energy consumption and performance in computing platforms [35].

On the other hand, managing hardware resources, to achieve energy efficiency, under different application scenarios (single or concurrent) is proving highly challenging due to runtime state-space expansion [14]. This has led to techniques for mitigating power consumption and performance degradation concerns. Power-aware platform design techniques including many-core architectures with the provision for dynamic task mapping and dynamic voltage frequency scaling (DVFS) have been the preferred tools for system designers over the years [79, 125].

Existing approaches for energy efficiency can be categorized into two types: offline (OL) and runtime (RT). In OL approaches, the system is extensively reasoned to derive energy and performance models [78, 4]. In RT approaches, the models are typically learnt using monitored information [128, 117]. Since RT modelling is costly in terms of resources, often OL and RT are complementarily coupled [4].

Modern applications exercise underline hardware in different ways, generating varying tradeoffs between power and performance. For example, CPU intensive applications tend to have higher computing workloads with less memory band-width. On the other hand memory intensive applications typically exercise higher memory bandwidth with low CPU workloads [111]. When these applications run concurrently the workloads generated by the hardware may vary significantly

compare to that of a single application. Hence, energy efficiency cannot be automatically guaranteed using the existing approaches that are agnostic of concurrent application workloads and behaviours [4].

This thesis opens a new research trend in the development of new framework design to dynamically select the number and type of cores with their frequencies for single and concurrent applications in heterogeneous many-core systems in order to improve energy efficiency. To achieve that we present new models, techniques and architectures.

1.2 STATEMENT OF ORIGINALITY

The major contributions of this thesis can be summarized as follows:

- A comprehensive literature study of power management techniques for many-core systems is presented. This study analyzes the advantages, disadvantages and limitations of the previous techniques of power management. The conclusions from this study can form a basis for any research aiming to utilize these management techniques' advantages and addressing their challenges.

- Experiments with the Odroid-XU3 platform were carried out in order to examine the power management of heterogeneous systems using directly measured values from the performance counters and built-in monitors, these experiments include managing the number of big cores, number of LITTLE cores, different core allocation, and the operating frequencies as a function of workload type. Furthermore, other experiments were carried out to investigate the impact of the CPU duty cycle with idle-wait state power, and controlling the number of active cores on the CPU performance and power tradeoffs [30].

- Developing a new performance counter for measuring the on-chip power consumption and performance caused by executing single and concurrent applications on heterogeneous many core systems. This performance counter can monitor system performance events (e.g. cache misses, cycles, instruction retired) and capturing the voltage, current, power, and temperature directly from the sensors of hardware platforms.

- Propose model-based runtime optimization approach for concurrent applications, practically implemented and demonstrated on a heterogeneous many-core system. Multivariate linear regression (MLR) is used to model power and performance tradeoffs expressed as IPS/Watt, determine the optimal system configuration (i.e. the number of parallel threads, their core allocations and operating frequencies) tailored for each kind of workload,

and maximize IPS/Watt for single and concurrent application scenarios using low-cost runtime adaptation algorithm [4].

- Propose a low-complexity, model-free and low-cost runtime approach for synergistic controls of dynamic voltage frequency scaling (DVFS) and task mapping (TM). Fundamental to this approach is an empirical and data-driven method, which classifies applications based on their memory and CPU requirements. The objective is to derive DVFS and TM policies, tailored to the classified workloads without requiring any explicit modelling at runtime. Due to simplified runtime classification, our approach can significantly reduce overheads. Furthermore, the model-free classification approach based RT enhances scalability for any concurrent application mix, platform, and metric having linear complexity which is not affected by the system heterogeneity, and the number of concurrent applications [6].
- To reduce overhead and complexity, workload classification is combined with model-based (multivariate linear regression technique) in a novel low-complexity runtime optimization approach with the aim of achieving maximized energy efficiency considering dynamic variations of workload and application scenarios.
- Implement the new approach for low-complexity runtime power adaptation as a Linux power governor to determine the hardware configuration knobs, such as the number and type of cores with their frequencies tailored to the workload type and application scenario. Validate this governor through extensive experimentation to demonstrate significant IPS/Watt improvements.

1.3 THESIS ORGANIZATION

This thesis is organized into seven chapters, as shown in Figure 1.3.

Chapter 1 "Introduction": introduces the motivations, objectives, contributions and structure of this thesis.

Chapter 2 "Background and Literature Review": provides background information and summarizes the literature on topics relevant to this thesis. In addition, the emerging on-chip interconnects included in recent literature is reviewed and discussed in details.

Chapter 3 "Platform Exploration Experiments": presents several experiments with the Ordoid-XU3 board to reveal the impact of parallelism in different types of heterogeneous cores on performance, power consumption and idle power efficiency.

Chapter 4 "Model-based Runtime Management of Concurrent Workloads": proposes the novel runtime optimization approach for concurrent applications, practically implemented and demonstrated on a heterogeneous many-core system by using MLR to model power and performance tradeoffs expressed as IPS/Watt.

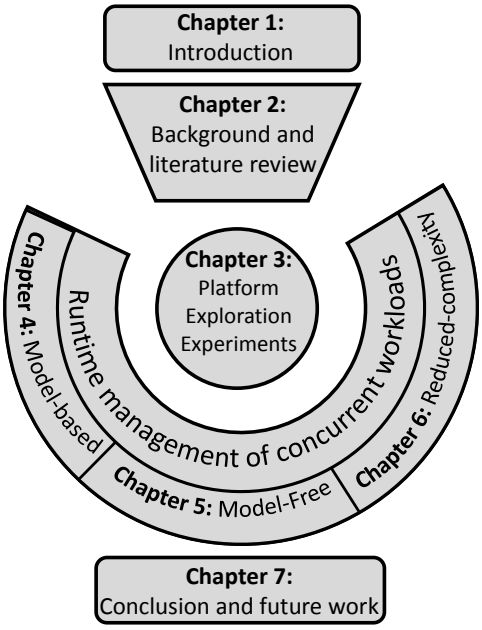


Figure 1.3: Thesis organization.

Chapter 5 "Model-free Runtime Management of Concurrent Workloads": proposes an runtime adaptation approach to improve the energy efficiency of a heterogeneous many-core system with concurrent workloads by using workload classification (WLC) techniques to derive DVFS and TM policies, tailored to the classified workloads.

Chapter 6 "Reduced-Complexity Runtime Management of Concurrent Workloads" develops the model-based runtime optimization approach by using the workload classification technique to reduce state-space size of the many-core heterogeneous system . Includes all possible core allocations and all possible DVFS combinations.

Chapter 7 "Conclusions and Future Work" summarizes the contributions of the thesis, discusses the implications of the presented research and draws the horizon for potential future work.

2

BACKGROUND AND LITERATURE REVIEW

This chapter presents the fundamental concepts involved in this thesis and gives a review of the existing work. The remainder of this chapter is organised as follows. Section 2.2 presents the principle of power consumption of complementary metal-oxide-semiconductor (CMOS) circuits of multiprocessors system on chip and Section 2.3 introduces the power management techniques which include dynamic voltage frequency scaling, many/multi-core architecture, heterogeneity of many core systems, dark silicon and workload scheduling. Finally, Section 2.4 gives an overview of the proposed techniques in the previous work relevant to the presented research.

2.1 INTRODUCTION

Over the last four decades, progress in information technology has changed every aspect of our lives – the ways we think, work, communicate, commute, and entertain ourselves. Modern computing systems, and especially mobile and embedded systems have had an enormous effect on our lives, including in areas such as the internet to consumer electronics, transporting, healthcare and manufacturing. However, these computing systems involve a performance-energy trade off as the current size of transistors is scaled down further [55]. This has led many researchers to investigate techniques to mitigate power consumption and performance degradation concerns. Such techniques include: power gating, dynamic voltage frequency scaling, emerging devices, and many-core architectures. The many-core technique has emerged as a consequence of recent advances in the integration density of transistors on a single chip, which has been achieved through the use of modern semiconductor processes [55]. However, despite the rapid recent increase in technology scaling [15], the speed of transistors may not increase at the historic rate due to threshold leakage current, with the supply voltage being scaled down, as shown in Figure 2.1.

2.2 POWER CONSUMPTION OF CPU

The dynamic power consumption and the static power consumption are the two components of the power consumption in CMOS processors as can be seen in (2.1)

$$P_{total} = P_{dynamic} + P_{static} \tag{2.1}$$

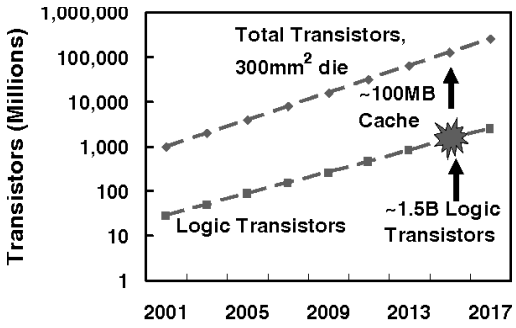


Figure 2.1: Transister integration capacity [15].

where $P_{dynamic}$ is the dynamic power consumption due to charging and discharging activities of load capacitance whenever the CMOS circuit is running a useful application i.e active operation, and P_{static} is the static power consumption due to several sources such as gate leakage and sub-threshold leakage whenever the CMOS circuits are not running useful computation i.e. idle mode [52][19].

Due to continued technology scaling the static power consumption is now a significant source of power consumption even in operation mode. The total power consumption (static and dynamic power) has to be optimized instead of simple dynamic power reduction. Design techniques exploration of the power consumption optimization need to work in a large dimension search space [46]. We discuss these in the subsections that follow.

2.3 POWER MANAGEMENT TECHNIQUES

Over the past decades, advances in chip fabrication has continued at a steady stride and have yielded substantial improvements in the power efficiency of CMOS chips. Power and energy always had a significant impact on processor design. However, a few recent studies propose power control algorithms for many-core architectures [34]. A number of power management techniques for many- and multiple-core processors have been proposed and these techniques can be classified into two approaches: power management during design time and power management at runtime, as can be seen in Figure 2.2. These two approaches, including sets of hardware and software techniques, have been used to reduce the power consumption while meeting system performance requirements.

In this chapter, many techniques for the management of power consumption are discussed, such as dynamic voltage frequency scaling, CPU-power and number of active cores, power-aware dark silicon management, and workload scheduling plus sharing resources.

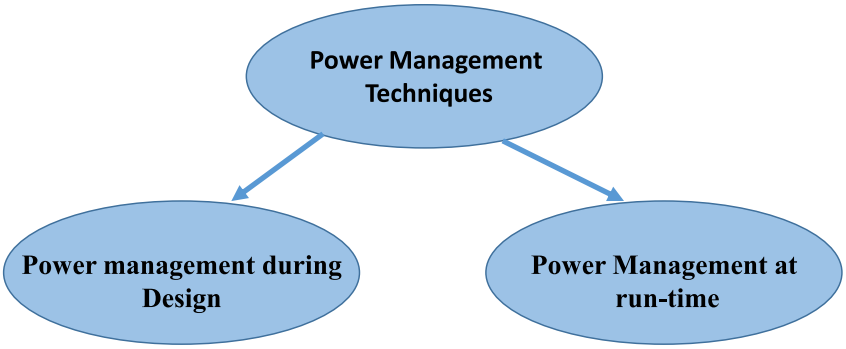


Figure 2.2: Approaches To Power Management

2.3.1 *Dynamic Voltage and Frequency Scaling*

Many studies have recently been conducted aiming to reduce the power consumption of many-core processors based on various techniques. These techniques include varying the clock frequency, and supply voltage correspondingly while a task is being processed, called dynamic voltage frequency scaling (DVFS) control [39, 58, 53, 31], this technique was introduced in the 1990's. DVFS is able to help decrease the power dissipation of CMOS circuits by reducing the supply voltage and/or frequency as shown in the equation below:

$$P_{dynamic} = \alpha C_{switched} f V^2 \tag{2.2}$$

where α is the switching probability or activity, $C_{switched}$ represents the effective switched capacitance of the transistor gates, f is the operating clock frequency, and V is the supply voltage [50, 57]. This equation shows that the $P_{dynamic}$ is proportional to the product of the operating frequency and the square of supply voltage. DVFS is a hardware power minimization technique which has been developed for embedded systems where the frequency and supply voltage are dynamically changed depending on the workload variation.

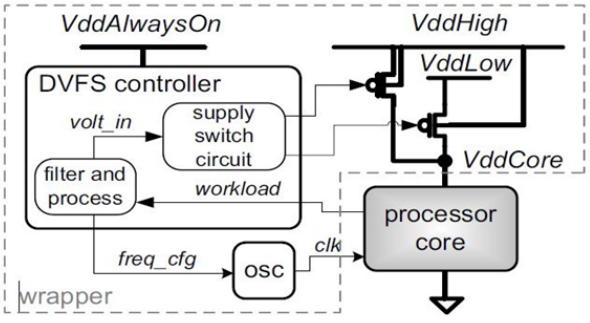


Figure 2.3: dynamic voltage frequency scaling with two voltage supply [57]

For instance, LeSueur et al. [57] circuits designed for dynamic voltage frequency scaling using two individual power supply voltages of 1.3 v and 0.8 v on a nine core processor as shown in Figure 2.3 and this technique resulted in an average energy 52% of the original energy on a JPEG application.

DVFS may be controlled at the system software level. For instance, DVFS is controlled in Linux with power governors [14], such as ondemand, performance, conservative, userspace and powersave. These governors use DVFS control to manage system power according to the knowledge and prediction of workload and user preference. Current Linux governors are, however, not able to optimize energy consumption, primarily because they select only either the maximum or minimum frequency depending on whether the workload is higher or lower than a given threshold [14]. This coarse-grain approach is not capable of taking advantage of the different degrees of parallelizability of applications and producing the most efficient scheduling.

2.3.2 *Many/multi-cores Systems*

In 1971, while microprocessors have been invented, performance enhancements from one era of processors to the following have been ruled via Pollack's regulation [15]. This states that the improvements in performance of the microprocessors is proportional to the square root of complexity (or area, assuming that the implementation makes use of the same CMOS technology). For instance, a dual-small core microprocessor can provide a performance improvement of 70-80 %, as compared to only 40% from a large monolithic core [15]. Each core is designed to run at a slower clock speed in order to reduce its energy per operation [45], while overall system performance can be recovered by operating both of the processors in parallel, as shown in Figure 2.4.

Despite the fact that, due to the generation scaling driven by Moore's law, a multi/many-core processor layout possibility has emerged which overcomes Pollack's regulation. This is because the use of multiple processors can offer close to-linear overall performance enhancement.

As the power consumption issue is becoming exacerbated with the newer micro-architecture design of microprocessors, parallel-computing has become widely employed in microprocessors. Therefore the multi/many-cores systems can be considered for solving these problems:

1. Saving the power consumption can be possible by turning on or off for each core.

2. It is possible to operate each core separately in reliable region (optimized voltage and clock frequency) depending on power budget.

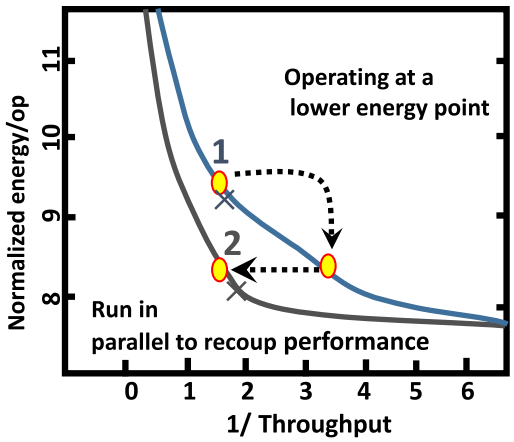


Figure 2.4: A single-core microprocessor that runs at a lower clock speed can be made to operate at lower energy per operation. Two single-core processors can be run in parallel to recover the overall system performance [45].

3. It is possible to distribute the heat across the die by distributing the application workloads among processor cores.

4. Establishing many core system which able to decide how and when to control the dynamic reconfiguration of processing cores with application needs.

The connection between core-power and number of cores has been studied to calculate the power dissipation in many-cores systems. For example, Intel Xeon CPU E5540 processors are used to investigate the relationship between processor power and number of active cores, as can be seen in Figure 2.5 which has a linear trend line [106].

2.3.3 *Heterogeneous Many-core Systems*

Advanced embedded systems such us mobile devises have enjoyed dramatic growth over the last decades, and their functionality has increased at a similar rate as they enable more immersive experiences. The expectations for more powerful features, greater flexibility, and high-performance are increasing dramatically [112].

Low power techniques have been used from clock power gating through dynamic voltage/frequency scaling, then the many cores which have been explained in previous sections. And most recently heterogeneous processing technology allows the use of multiple types of cores. There are many types of heterogeneity as shown below [88, 99]:

1. using different types of cores
    a) different types of CPU (such as ARM big.LITTLE)

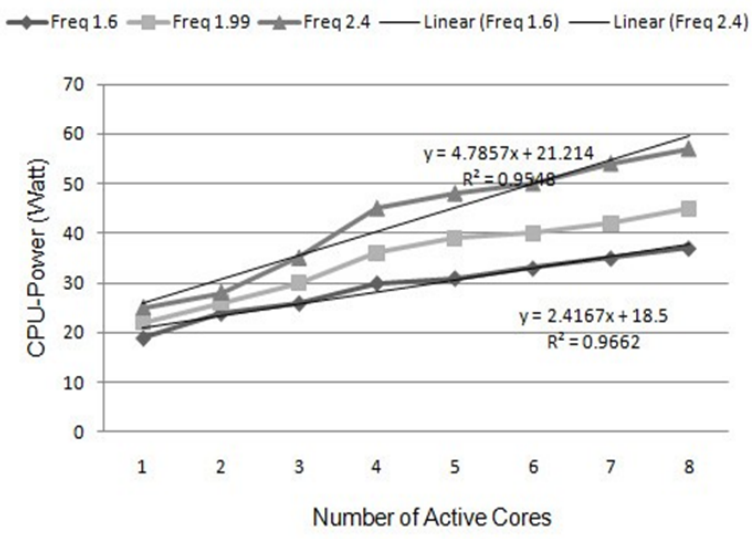


Figure 2.5: CPU-Power consumption relationship with number of active cores [106].

b) integrating graphics processing unit (GPU) with CPU

c) using specific types of programmable core such as accelerators

2. different types of hierarchies

3. using different types of software (operating system (OS), tools,...)

ARM big.LITTLE is the latest in a series of innovations, that have allowed high improvements in embedded system performance within a constant power budget [23]. LITTLE refers to smaller high efficiency core on the system-on-chip (SoC) which has been used to low performance applications for example, e-mail , web browsing, etc. While big refers to bigger higher performance cores which have been used for high performance applications for example, gaming, multimedia, etc. These cores are connected by a coherent interconnect, rustling in an embedded system [1]. In such many core systems the big cores can produce peak performance, and the little cores can consume minimal power. To ensure that tasks are paired dynamically to the appropriate core type, the OS must be aware of the performance requirement for each application type. Global task mapping software adjusts the kernel scheduler to be aware of the performance requirements for each thread. As performance changes the tasks are switched between the big and LITTLE cores to maintain the most efficient performance. Global task scheduling provides the most flexibility for tuning the performance and power balance in an SoC [131]. With the efforts being

no more complex and for a standard DVFS based system. The other types of heterogeneity are out of scope of this thesis.

2.3.4 *Power-aware dark silicon management*

In recent years, the dark silicon power management technique has been widely used in many-core systems. The dark silicon indicates that for a fixed power budget, only finite numbers of cores can be powered on (operating at high performance) in the reliable region while others remain unpowered (dark).

Shafique et al. [37] presented the basic flow of connections between the various elements of the dark silicon management (DaSiM) technique as can be seen in Figure 2.6. This technique is based on two concepts.

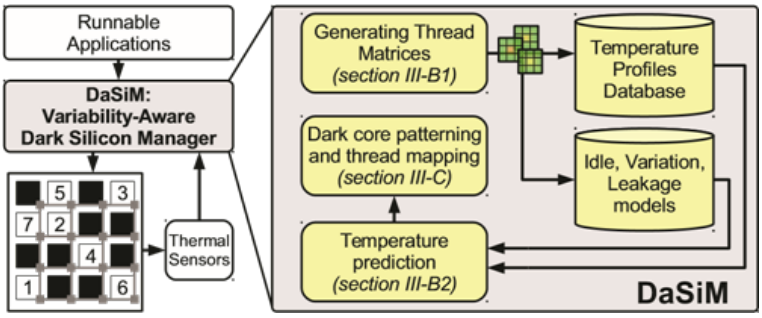


Figure 2.6: Flow of patterning, mapping, prediction (pmp) [37]

At first runtime application mapping and the dark cores technique is used to establish the power control of cores. Secondly, temperature prediction (section III-B2 in Figure 2.6) is used to establish the thermal matrix for each thread. A Gem 5 [11] simulator with McPAT [63] has been used in the experimental setup to demonstrate this technique.

2.3.5 *Workload scheduling and sharing resources*

In recent embedded systems and chip-multiprocessors (CMPs), cores are not totally independent processors but rather share particular on/off chip resources. These resources are the last level cash L2-L3, interconnects (memory bus), pre-fetchers and the Dynamic random-access memory (DRAM) controller. The requests coming from different threads and different cores are treated as if they were all requested from one single source, as can be seen in Figure 2.7 [132].

Workload scheduling or mapping means how the running applications will be distributed on the cores in a particular period of time to minimize power consumption [126].

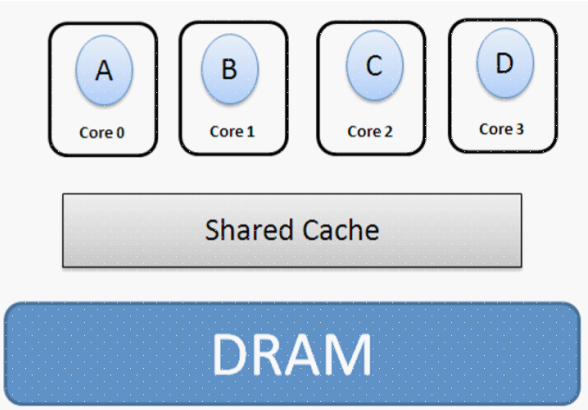


Figure 2.7: Examples of thread-to-core mapping in chip-multiprocessor (CMP) configurations [132]

2.4 LITERATURE REVIEW

Contemporary computing systems, including embedded and high performance systems, are exhibiting increased complexities in two dimensions. In one dimension, the number and type of computing resources (cores) are increasing in hardware platforms, and in the other, an increasing diversity of applications are being executed concurrently on these platforms [81], [79]. Managing hardware resources, to achieve energy efficiency, under different application scenarios (single or concurrent) is proving highly challenging due to runtime state-space expansion [36].

This section discusses a set of power management techniques carried out by different academic and industrial groups. This literature is by no means implies a total survey of power management research. Comprehensive survey studies have been presented in many papers for example, Mittal [71], Singh et al.[100], and Kallimani [49]. Specific work reviews related to the contributions of the thesis are presented in the appropriate chapters.

2.4.1 *Power Management Techniques During Design*

As power and energy consumption has become a limiting factor to continued technology scaling and performance improvements [14], techniques for improving power and energy efficiency have emerged.

In the past few years, there have been various studies in power management of embedded systems during design or offline time.

Some of these techniques have been presented in Section 2.3. Mittal [71] and Kallimani et al. [49] classify power management techniques during design into four categories :

1. Dynamic voltage scaling (DVS), dynamic frequency scaling (DFS), and dynamic voltage frequency scaling DVFS techniques which have been explained in Section 2.3 [16, 77, 54, 90, 29, 42, 22, 21, 130]

2. Dynamic power management or sometimes called low power mode management [40] .

3. Micro-architectural design techniques for specific components. For example, memory (cache, main, scratchpad) [72, 69, 104]

4. Digital signal processing (DSP) or GPUs or FPGAs have been used as unconventional-cores [109, 120, 12, 65]

In terms of using DVFS technique, Ma et al. [68] proposed a low power control approach for many-core processors executing single applications. This approach has three layers of design features as shown in Figure 2.8: firstly, adjusting the clock frequency of the chip depending on the power budget; secondly, dynamically group cores to run-the same applications (as also shown in [127]), and finally, modify the frequency of each core group (as also shown in [123]).

Beh et al. [7] suggested a new design for DVS for a complex system-on-chip SoC by dividing it into many blocks depending on suitable supply voltage levels and corresponding clock frequency to each processing element. These elements are CPU cores, IO, memory and DSP, which are called function blocks as can be seen in Figure 2.9.

Here the VRM represents a voltage regulator module, which is an electronic circuit designed to supply fixed voltage levels.

Per-core dynamic voltage frequency scaling DVFS is an effective power management technique to save power consumption. Kim et al. [54] designed a new power supply paradigm with an on/off regulator to implement DVFS per-core, as shown in Figure 2.10.

In this paradigm, many algorithms with different benchmarks have been adopted to implement per-core DVFS in an offline manner taking into account transition time and overheads effects.

The hardware designers of the embedded systems typically provide many operating modes to lower the power consumption. For example, a low power mode has been used to reduce the power consumption of the multi/many-core system when cores are idle. Hoeller et al. [40] presented a new paradigm for interfacing hardware and software components to manage the power consumption. Fundamental of this approach is that allowing applications to precise when some components are not being utilized and based on this data, particular components, cores or the entire system can be moved to low power mode.

In terms of using micro-architecture design techniques, Mittal and Zhang [72] presented a new cache memory design to save cache leakage power. This approach is based on software to predict the

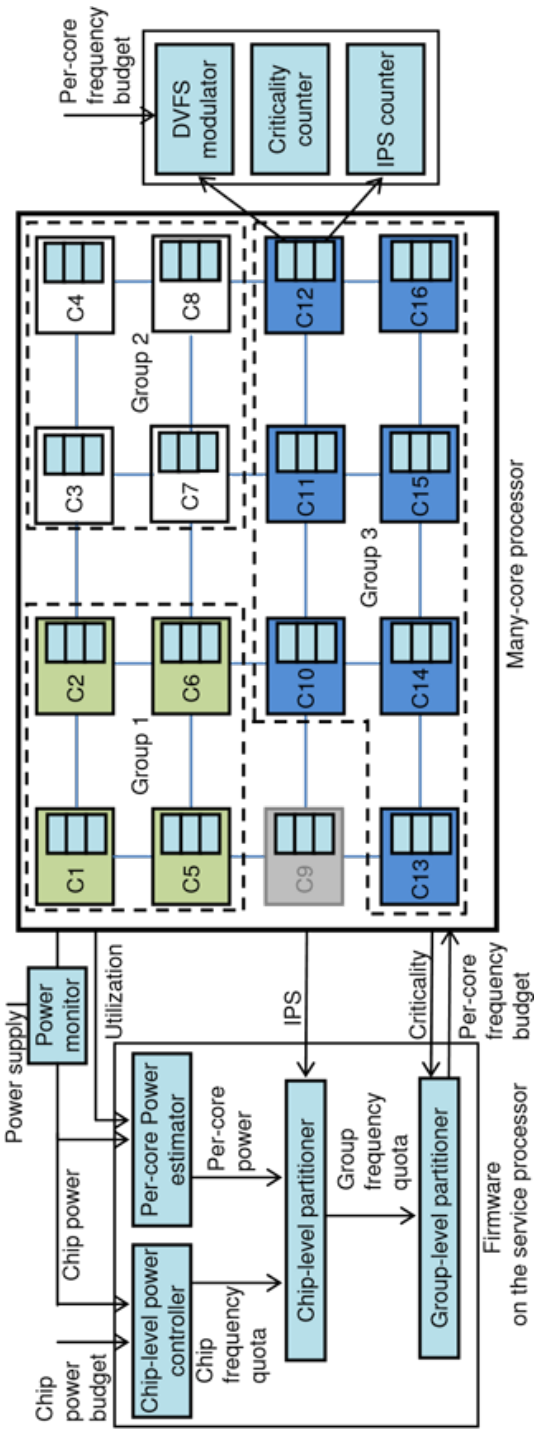


Figure 2.8: Three-layer power control architecture for a 16-core chip multiprocessor [68]

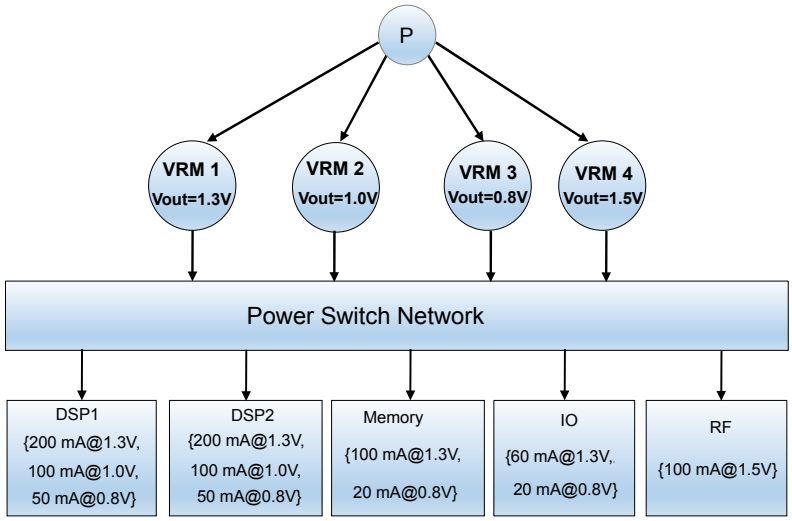


Figure 2.9: Proposed architecture of power delivery network (PDN) to support dynamic voltage scaling. The output voltage of each VRM is fixed [7]

power efficiency and cache usage for multiple cache configuration and change cache configurations for reducing the cache power.

2.4.2 *Runtime Power Management Techniques*

Modern embedded systems execute multiple applications, both sequentially and concurrently, on heterogeneous and homogeneous platforms. Determining the most energy-efficient system configuration (i.e. the number of parallel threads, their core allocations and operating frequencies) tailored for each kind of workload during execution is extremely challenging because the state space is very large and each application requires different optimization.

Recent years have witnessed various studies in power and energy optimization in embedded systems during execution time (i.e. runtime). These studies provide various methods of the ability to learn information without being clearly programmed, with regard to improve power and energy efficiency. The learned information during the operation is used to predict the appropriate task mapping and DVFS for future execution. In general, runtime power management techniques policies have three objectives [9];

1. The total power saving ought to be maximized.
2. The latency penalty is reduced.
3. The runtime overhead ought to be reduced to prevent prolonging of the kernel time.

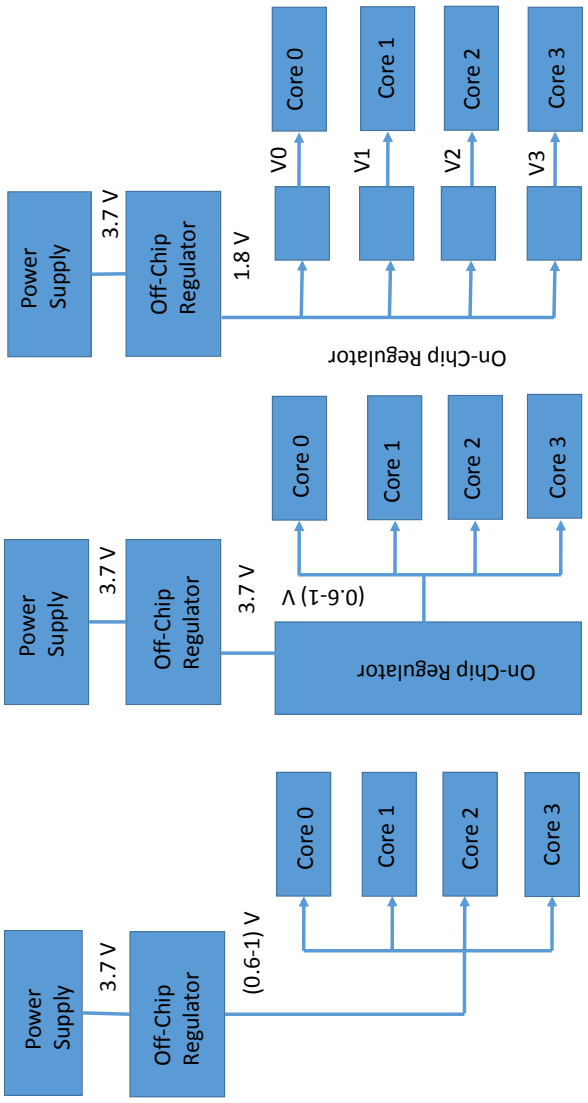


Figure 2.10: Three power-supply configurations for a 4-core CMP [54] .

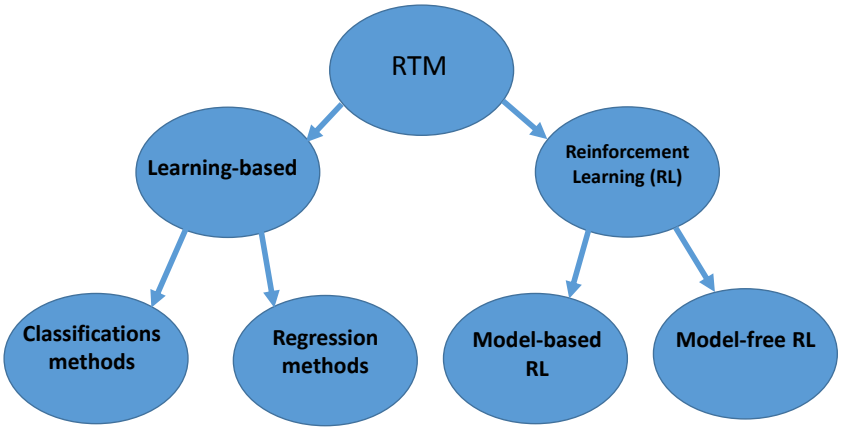


Figure 2.11: A taxonomy of runtime power management techniques.

Based on the principles they employ, existing studies can be largely categorised into two approaches: learning based and reinforcement learning, as can be seen in Figure 2.11.

2.4.2.1 *Learning based runtime management*

Many strategies have been adopted in this approach, which can be classified based on learning principle into two taxonomies i.e. classifications techniques and regression techniques. The first one has been used for predicting application response based on the applications type such us email, medical imaging, and games or based on application behaviour (CPU-intensive, memory-intensive, or both).

On the other hand, learning based regression techniques have been used for predicting continuous response for instance, changing in temperature or fluctuating in voltage or frequency. Regression techniques are used to learn model parameters from data collected at runtime, and the resulting models are used to derive next step control decisions. Existing work that proposed the use of regression in the runtime management are those by Sheng et al. [128], Shafik et al. [93], and [18, 41, 44, 62] . Sheng et al. presented an adaptive power minimization approach using practical implementation on heterogeneous platforms. Fundamental to this approach is the use of runtime linear regression (LR)-based modelling of the power and performance tradeoffs as can be seen in Figure 2.12. Using the model, the task mapping and DVFS are suitably chosen to meet the specified performance requirements.

Shafik et al.[93] proposed a new runtime adaptation approach for modern many core systems to reduce the power consumption based on parallel application. This approach depend on workload prediction with DVFS technique in runtime as can be seen in Figure 2.13. This adaptation is easy to adjust through performance annotations in

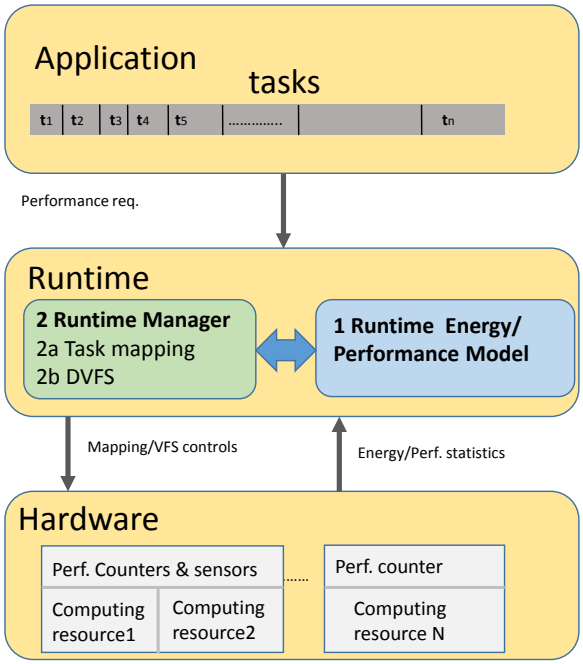


Figure 2.12: Block diagram of the proposed energy minimization approach [128].

sequential and parallel parts of workloads. This approach has been validated on Intel Xeon E5-2630 platform , which consist of 24 cores. NAS benchmark has been demonstrated on this platform based on OpenMP library. Furthermore, this approach shows that the energy consumption can be minimized significantly compared to existing approaches.

Another runtime approach for homogeneous many core systems has been adopted by Leech et al. [59], realizing power-aware performance adaptation for homogeneous many cores systems. This approach is based on power and performance models that can be determined during runtime by linear regression learning based on low complexity hypotheses of power and performance for a given operating frequency. The approach is extensively demonstrated using stereo vision applications running on a 61-core Intel Xeon Phi platform. DVFS with multi thread techniques have been used in runtime to achieve significant improvement in energy efficiency compared to existing approaches.

Juan et al. [48] proposed a new runtime approach based on using a model selection from machine learning. Using the adaptation model to determine the optimal dynamic voltage frequency operating points under-extend threshold, nominal, or turbo-mode conditions. The approach is demonstrated by Sniper and McPAT simulations running PARSEC and SPLASH-2 benchmarks at different operating frequencies. In this approach two types of experiments are carried

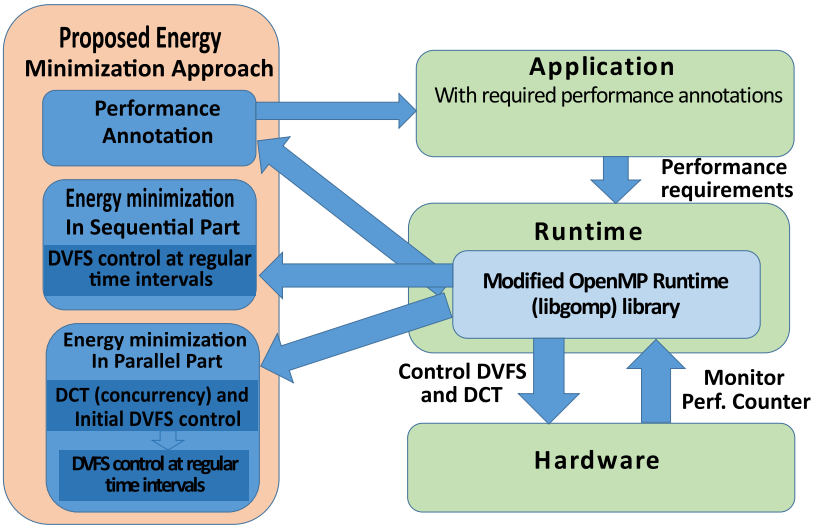


Figure 2.13: Proposed energy minimization approach [93]

out. Firstly: investigation of energy minimization under frequency requirements; secondly, investigation of performance maximizing under power budget. The experimental results show that the improvements in performance under power budget is less than the reduction in energy consumption under the performance condition.

The method of combining DVFS with dynamic concurrency throttling at runtime have been investigated recently. For instance, Jian Li and Jose F. Martinez [62] proposed a new runtime optimization approach for heterogeneous many core systems, fundamental to this study is that runtime optimization of concurrency with DVFS for parallel applications can achieve energy-efficient execution and high performance. This runtime optimization occurs in two-dimensional scenario, firstly; task mapping of active cores, secondly; the various frequency/voltage levels available.

Another prediction model for power/performance optimization in runtime for homogeneous systems has been proposed by Curtis et al. [26]. In this approach two knobs DVFS and dynamic concurrency throttling (DCT) have been used to reduce the dynamic power consumption. Using experimental measurements on two Intel Xeon E5320 quad-core processors platform with seven benchmarks from the OpenMP version of the NAS Parallel Benchmark suite to validate this approach. Multivariate linear regression is used to establish multi-dimensional empirical prediction model coefficients offline.

Table 2.1: Features and limitations of the supervised model-based learning techniques approaches.

Approach	Application	Platform	Validation	Runtime type	WLC	Control Knobs	Optimization
Juan et al. [48]	Single	Homogeneous	Simulation	Model-based learning	No	DVFS	Energy saving+ Performace improvement
Sridharan et al. [103]	Single	Homegenous	Simulation	Model-based learning	No	Task mapping	Energy saving+ Energy efficincy improvement
Ma et al.[67]	Single	Heterogenous	Simulation	Model-based learning (offline regression)	No	DVFS+Task mapping	Power saving+Performace improvement
Curtis et al. [25]	Concurrent	Homegenous	Practical	Model-based learning (offline regression)	No	DVFS+DCT	Power saving+Performace improvement
Sasaki et al. [89]	Single	Homegenous	Practical	Model-based learning (Multivarate linear regression)	No	DVFS+Number of active cores	Performace improvement
Wu et al. [124]	Single	Heterogenous	Practical	Model-based learning (linear regression)	No	DVFS	Performace improvement

Recently, numerous studies have focused on workload type with using mixture of two technique to optimize the power/performance for many core embedded systems [24, 102, 27, 28, 131]. For instant, the technique proposed in [27] first derives energy and cycles-per-instruction models. Then, these models have been used to characterize the workload at runtime by selecting the best dynamic power management (DPM) scenario together with DVFS setting for each core.

Sozzo et al. [102] proposed a new runtime Linux scheduler using practical implementation of their approaches on heterogeneous platforms. Fundamental of this approach is that optimize voltage/frequency under performance constraint, manages the task mapping of application threads among heterogeneous cores, and uses performance counters to monitor the varying in workload characteristics. The approach is demonstrated by Odroid XU3 running the Black and Scholes parallel model application of OpenMP. Moreover, The experimental results of this approach show a significant improvements in power consumption and performance compared to current Linux governors. Similarly, the work in [131] proposed a practical implementation approach on a set of web-browsing applications running on ARM big.LITTLE heterogeneous many core systems. In this approach linear regression technique is used to build power/performance models, and then use these models to determine the optimal core allocation for minimum power consumption.

Table 2.1 shows the most features and limitations of the existing approaches which employ supervised model-based learning techniques. It is clear that the researchers did not include the workload classification in these approaches.

In terms of using classification-based techniques, numerous studies have focused on using these techniques in dynamic power management with DVFS together at runtime [36, 56, 87, 24, 13, 113, 122]. For instance, Gupta et al.[36] proposed a new runtime approach based on workload classification. To build this classifier extensive offline experiments are made on heterogeneous many core platforms and Matlab is used to determine the classifier parameters offline. Pareto function is used to determine the optimal configuration. However, this classification is heavily based on offline analysis results, and assigns an application a fixed type, regardless of its operating context. It also requires the annotation of applications by application programmers through using a special API.

Another runtime workload classification approach for heterogeneous multi-core systems is proposed by Reddy et al. [87]. Memory Reads Per Instruction (MRPI) metric has been used to perform workload classification. This metric uses performance counters to determine the performance in terms of instruction per second (IPS) during application execution. The workload classification aims to determine the optimal voltage/frequency without any loss of performance. Using

experimental measurements on ARM big.LITTLE Odroid-XU3 platform with five applications from SPEC CPU2006 benchmarks and two applications from PARSEC to validate this approach. However, this work does not merge DVFS with task mapping techniques for workload on heterogeneous platform.

Table 2.2 shows the most features and limitations of the existing approaches based on supervised classification-based and model-based learning techniques.

Table 2.2: Features and limitations of the supervised classification-based and model-based learning techniques approaches.

Approach	Application	Platform	Validation	Runtime type	Workload Classification	Control Knobs	Optimization
Bitirgen et al. [13]	Single	Homegenous	Practical	Model-based learning (machine learning to predict performance model)	Offline classification based on memory bound	Task mapping	Performanc optimization
Van et al. [113]	Single	Heterogenous	Simulation	classification+ performance modelling	Partial WLC to big or small cores	Task mapping	Performace improvement
Wen et al. [122]	Concurrent	Heterogenous CPU/GPU	Practical	Model-based learning	Offline classification for OpenCL applications	Task mapping	Performace improvement
Cochran et al.[24]	Single	Homegenous	Practical	Model-based learning	Offline classification	DVFS+Task mapping	Power/Performance tradeoff
Gupta et al. [36, 34]	Single	Heterogenous	Practical	Model-based learning (Pareto)	Partial classification (offline determining the classifier parameters	DVFS+ Number of active cores	Energy efficincy improvement

2.4.2.2 *Reinforcement learning (RL)*

Reinforcement learning RL methods are a set of arrangements for ideal long-term activity choice such that activities take under consideration both immediate and delayed results [43]. Figure 2.14 shows the general description of the RL model which comprises of a finite state space S, an agent, a set of available actions A, and a reward function R [118]. Mathematically, RL [105] describes some kind of solution to the problems of the Markov Decision Process (MDP) identified by the group S, A, T , R, and π where:

1. S: a set of states $s \in S$
2. A: a set of actions $a \in A$
3. T: the transition function maps each state-action combine to a dissemination over successor states
4. R : the reinforcement function set three times the state-action-successor state to a stander return r maximizing the overall predicted future

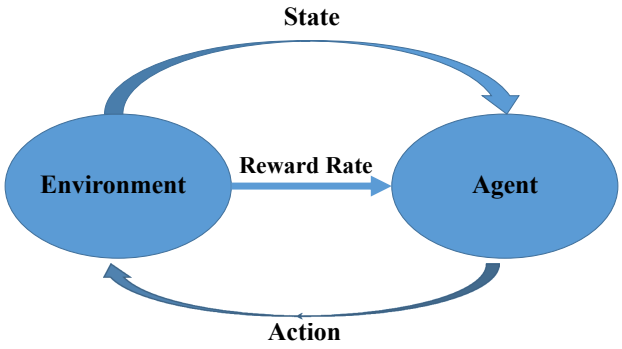


Figure 2.14: Agent-environment interaction Model.

The objective is to specify $a \rightarrow \pi(x)$ policy that sets each state to the action maximizing the entire anticipated future return of activities a in state s. RL methods can be classified into two types model-based and model-free. Model-based techniques evaluate a clear model of the environment and the agent which mean that the applies descriptions based on model decision, in options that reflect current preferences regarding results. While, Model-free approaches do without any explicit information of the elements of the environment or the results of actions and assess how good actions are implemented through trial and error learning [43].

In other words, model-based RL assume knowledge of the transition matrix T , the reward function R, the state S, and action spaces A which define the model. And, model-free RL strategies are used in

circumstances where agents do not know T and R where the choice trees are too complicated to assess.

2.4.2.3 *Model-Based RL*

Over the years substantial research has been carried out addressing runtime energy minimization and/or performance improvement approaches. The technique of learning reinforcement has been used for many core systems by Tan et al. [107] and Ye et al. [129]. There are a few solid similarities between dynamic power management and reinforcement learning: the next decision in DPM based on the current status and the previous statistics, while in the RL technique the agent monitors the state environment s_t at t time, tacking action a_t, and therefore gets a r_t reward. Because of the similarities in the trial and error operation, power management can be performed using reinforcement learning techniques [76].

In the past few years, there have been many studies in model-based RL. For instants, Figure 2.15 shows the general block diagram of reinforcement learning power management approach at runtime [76], where the agent represents the power manger which decides the action depend on Q-values. In This approach the multilevel framework has been investigated to reduce the state-space learning-based runtime for homogeneous systems. A set of experiments on real benchmarks is carried out on the cycle-accurate simulator to validate this approach. Furthermore, this approach shows a significant improvement in speed comparing with state-of-the-art work.

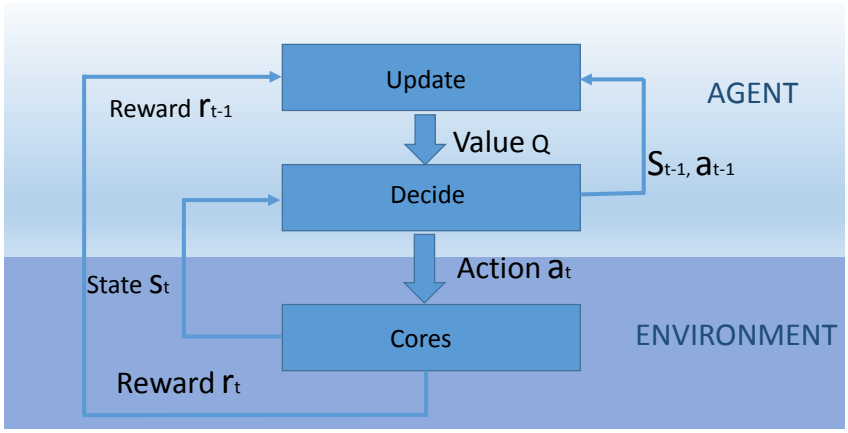


Figure 2.15: Overall flow of power management based on multilevel reinforcement based learning [76]

Table 2.3: Features and limitations of the existing approaches of the model based RL.

Approach	Application	Platform	Validation	RL Type	WLC	Control Knobs	Optimization
Shen et al.[97]	Single	Homegenous	Practical	Model- based	No	DVFS	Temperture, performanc, and energy
Juan-2012 et al.[47]	Single	Homegenous	Practical	Model- based	No	DVFS	Performance under power constraints
Shafik-2016 et al.[96]	Single	Homegenous	Practical	Model- based	No	VFS	Energy minimization
Wang-2010 et al.[116]	Single	Homegenous	Simulation	Model- based	No	DVFS	Energy efficiency improvements
Pan et al.[76]	Single	Homegenous	Simulation	Model- based	No	DPM	Energy saving
Prabha et al.[80]	Single	Homegenous	Simulation+ Practical	Model- based	No	DPM	Energy saving
Chen et al.[20]	Single	Homegenous	Simulation	Model- based	No	DVFS	Performance improvements
Ye et al.[129]	Single	Homegenous	Simulation	Model- based	No	Task mapping	Power-performance tradeoff
Tesauro et al.[108]	Single	Homegenous	Simulation	Model- based	No	Task mapping	Performance optimization for datacenter

Prabha and Monie [76] used model-based RL to save the power consumption of many core systems. Fundamental to their approach is using the RL techniques to turn off the idle components. The approach is extensively demonstrated using the SPLASH-2 benchmarks running on the multiprocessor performance simulator Multi2Sim and McPAT.

These approaches have considered a single-metric based optimization: primarily performance constrained power minimization, or performance improvement within a power budget. Furthermore, these approaches are primarily focused on single application workloads running on homogeneous many core systems.

2.4.2.4 *Model-Free RL*

A model-free runtime (RT) workload classification (WLC) approach with corresponding DVFS controls for dynamic power management DPM is proposed by Wang and Pedram [117]. This approach employs reinforcement learning, with the action space size a big concern for the authors. In this approach the semi-Markov decision process (SMDP) based on the temporal difference (TD) learning technique, but for only homogeneous systems at much higher granularities than CPU cores. In another work [119], a new architecture for hierarchical dynamic power managements based on model free reinforcement learning is proposed. In this approach, the reinforcement learning architecture consists of two layers: a component-Ievel local power manager (LPM) and a system-level global power manager (GPM). The latency and component power optimization is achieved by the first layer. The second layer interacts with the CPU scheduler to implement application scheduling. Furthermore, experimental results show that this approach can save the average power up to 31.1% compared to existing methods.

A recurring scheme in these approaches is that the energy efficiency is primarily focused on single-application workloads without considering its variations among concurrent applications. However, the same application can exhibit different energy/performance tradeoffs depending on whether it is running alone or concurrently with other different workloads. This is because:

1. the workload context switches within the application between memory- and CPU-intensive routines, and

2. architectural sharing between applications affects the energy/performance tradeoffs

Table 2.4: Features and limitations of the existing model-free reinforcement approaches.

Approach	Application	Platform	Validation	Workload Classification	Control Knobs	Optimization
Liu et al. [64]	Single	Homegenous	Simulation	No	DVFS	Power/ Performanc tradeoff
Wang et al. [117]	Single	Homegenous	Practical	Partial workload classification	DPM	Energy saving
Shen et al. [98]	Single	Homegenous	Simulation	No	DVFS	Power/ Performanc tradeoff
Wang et al. [119]	Single	Heterogenous	Simulation	Partial workload classification	Task mapping	Power and latency tradeoff

3 PLATFORM EXPLORATION EXPERIMENTS

This chapter describes various experiments carried out in order to find the correlation between frequency, power consumption and performance in a multi-threaded, multi-core heterogeneous system. The results of these experiments will provide ideas on how to reduce power and energy consumption without significant performance deteriorations. In addition, these experiments introduce new characteristics that must be considered in power/performance models. Furthermore, these results provide support for the initial validation of the parametric significance-driven modelling approach [83].

This chapter is organized as follows. Section 3.2 shows the experimental environment, the configuration of the system used in the experiments and the applications. The characterization experiments are described in Section 3.3, which includes DVFS, CPU-power and number of active cores, and duty cycling with idle-wait state. Section 3.4 concludes the work in this chapter.

3.1 INTRODUCTION

Over the past decades, advances in chip fabrication has continued at a steady stride and have yielded substantial improvements in the power efficiency of CMOS chips. Power and energy have always had a significant impact on processor design. A number of power management techniques for many-and multiple-core processors have been proposed that take advantage of the fact that an increase in the number of cores for the CPU processor architecture may be more energy efficient than increasing clock frequency. The popularity of heterogeneous architectures, containing two or more types of different CPUs is growing [15]. These systems offer better performance and concurrency, however it is necessary to improve the modelling in order to ensure optimal power and energy consumption.

This has led to techniques for mitigating power consumption and performance degradation concerns. Power-aware platform design including many-core architectures with the provision for dynamic task mapping and voltage frequency scaling DVFS have been the preferred tools for system designers over the years [15, 68].

The Odroid-XU3 board [2] provides facilities that support studies to better understand the nature of multi-core heterogeneous systems. The board provides the possibilities to apply techniques like voltage frequency scaling, affinity, and core disabling, which are used to

optimize the system operation in terms of performance and energy consumption.

For the first time, our study reveals the impact of parallelism in different types of heterogeneous cores on performance, power consumption and idle power efficiency [5]. The major contributions of this chapter are as follows:

- Investigate the CPU performance and power tradeoffs using directly measured values from the performance counters.
- Analysis of DVFS for many cores heterogeneous microprocessors A7 and A15.
- Investigate the CPU duty cycle with idle-wait state power, and controlling the number of active cores.

3.2 SYSTEM ARCHITECTURE AND PLATFORM DESCRIPTION

The popularity of heterogeneous architectures, containing two or more types of different CPUs is growing. These systems offer better performance and concurrency, however it is necessary to ensure optimal power and energy consumption. The Odroid-XU3 board supports techniques such as DVFS, affinity and core disabling, commonly used to optimize system operation in terms of performance and energy consumption [2] [101].

The Odroid-XU3 board [2] is a small eight-core computing device implemented on energy-efficient hardware. The board can run Ubuntu 14.04 and Android 4.4 operating systems. The main component of Odroid-XU3 is the 28nm Application Processor Exynos 5422. The architecture of the processor is shown in Figure 3.1. This System-on-Chip is based on the ARM big.LITTLE [1] heterogeneous architecture and consists of a high performance Cortext-A15 quad core processor block, a low power Cortex-A7 quad core block, a Mali-T628 GPU and 2GB DRAM LPDDR3.

The board contains four real time current sensors that give the possibility of the runtime measurement of power consumption on the four separate power domains: big (A15) CPUs, LITTLE (A7) CPUs, GPU and DRAM. In addition, there are also four temperature sensors for the each of the A15 CPUs and one temperature sensor for the GPU.

On the Odroid-XU3, for each power domain, the supply voltage (Vdd) and clock frequency can be tuned through a number of pre-set pairs of values. The performance-oriented Cortex-A15 block has a range of frequencies between 200MHz and 2000MHz with a 100MHz step, whilst the low-power Cortex-A7 quad core block can scale its frequencies between 200MHz and 1400MHz with a 100MHz step. DFS is applied for A15 when its frequency ranges between 200MHz and 700MHz (the Vdd stays constant in this region) or for A7 when its

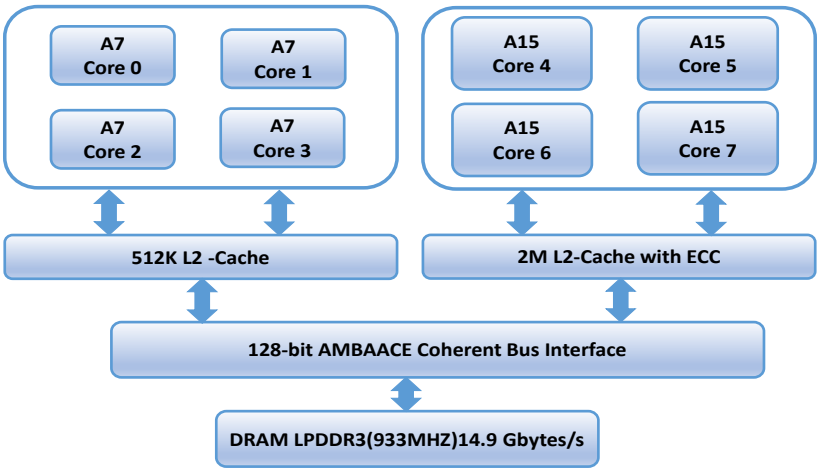


Figure 3.1: Exynos 5422 block diagram [2].

frequency ranges between 200MHz and 500MHz. DVFS is used when the frequency is 800MHz and above for A15 or 600MHz and above for A7.

The Cortex-A15 block is a high performance 32-bit quad core mobile processor using ARMv7-A instruction set. It has 32 KB instruction and 32 KB data caches. In addition, 2 MB of Level 2 Cache is provided. Each A15 core has integrated floating point unit VFPv4.

Cortex-A7 has the same architecture and feature set as Cortex-A15, however Cortex-A7 microarchitecture provides optimum energy efficiency. It has 512 KB Level 2 Cache. The LITTLE Cortex-A7 processor is more suitable for performing low power tasks like texting, background processes and audio.

3.2.1 *Applications Workload*

The PARSEC [10] benchmark suite attempts to represent both current and emerging workloads for multiprocessing hardware. It is a commonly used benchmark suite for evaluating concurrency and parallel processing. We experiment by using PARSEC on the Odroid-XU3 platform, whose heterogeneity can be representative of different design choices that can greatly affect workloads. PARSEC applications exhibit different memory behaviours, different data sharing patterns, and different workload partitions from most other benchmark suites in common use. The characteristics of applications, according to [10] , which are used in this work can be seen in Table 3.1.

Concurrency of processing applications: Modern embedded systems have facilitated placing multi/many core processor architectures on a single chip to improve performance. These embedded systems execute multiple applications, both sequentially and concurrently.

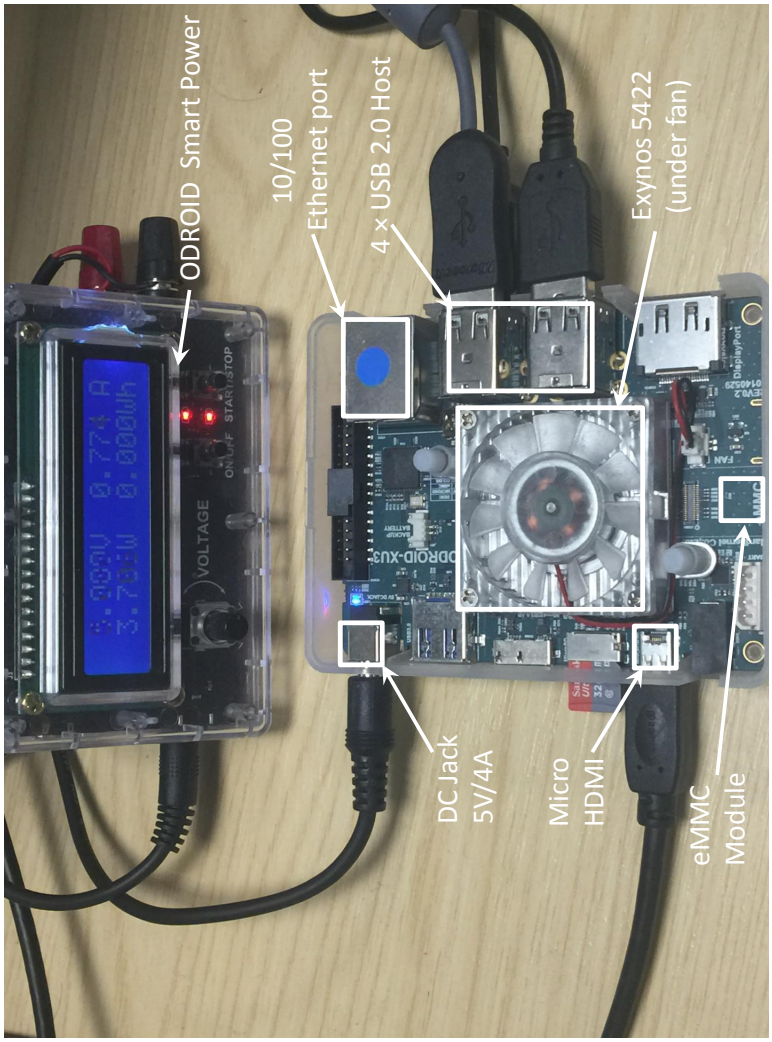


Figure 3.2: Exynos 5422 system set.

Concurrency can be characterized at different levels, from high level programming languages through to task definition at the operating system level [51]. Parallelism can be given by the processor architecture design to permit concurrency exploitation.

Table 3.1: Qualitative summary of the inherent key characteristics of PARSEC benchmarks [10].

Program	Application Domian	Application Type	Parallelization Model	Granularity	Working Set	Data Usage Sharing	Exchange
bodytrack	Computer Vision	CPU and memory intensive	data-parallel	medium	medium	high	medium
ferret	Similarity Search	CPU intensive	pipeline	medium	unbounded	high	high
fluidanimate	Animation	memory intensive	data-parallel	fine	large	low	medium
canneal	Engineering	CPU intensive	unstructured	medium	unbounded	high	high
freqmine	Data Mining	CPU intensive	data-parallel	fine	unbounded	high	medium
streamcluster	Data Mining	memory intensive	data-parallel	medium	medium	low	medium

Table 3.2: Performance Counter Events.

perf_eventt_name	Description
INST_RETIRED	Instruction architecturally executed.
BUS_CYCLE	Bus cycle
MEM_ACCESS	Data memory access.
L1I_CACHE	Instruction Cache access.
L1D_CACHE_WB	Data cache eviction.
L2D_CACHE	Level 2 data cache access
L2D_CACHE_WB	Level 2 data cache refill
L2D_CACHE_REFILL	Level 2 data cache write-back.

3.2.2 *Performance Counters*

In this work, we make use of performance counters to monitor system performance events (e.g. cache misses, cycles, instruction retired) and at the same time capture the voltage, current, power, and temperature directly from the sensors of Odroid-XU3. The performance counter consists of two modules: kernel module and a user space module.

In this chapter, we use performance counter readings to monitor system performance events (e.g. cache misses, cycles, instructions retired) and use readings from the built-in sensors of the Odroid-XU3 to monitor physical parameters including voltage, current, power and temperature.

The hardware performance counter readings are obtained using the method presented by Walker et al. [114]. In the user space module the event specification is the means to provide details of how each hardware performance counter should be set up. Table 3.2 lists examples of performance events, some of which are explained as follows:

1. INST_RETIRED is the retired instruction executed, and is part of the highly reported instruction per cycles (IPC) metric.
2. Cycles is the number of core clock cycles.
3. MEM_ACCESS is Memory Read or Write operation that causes a cache access to at least the level of data.
4. L1I_CACHE is level 1 instruction cache access.

3.3 CHARACTERIZATION EXPERIMENTS

Experiments with the Odroid-XU3 platform were carried out in order to examine the power consumption under different operation frequencies and voltages. The frequency of each block can be changed independently using special utility programs and the system scales

the operating voltage of the block to fit the chosen frequency. Eight cores in the board are numerated as follows: core 0, core 1, core 2 and core 3 belong to the A7 processor block, core 4, core 5, core 6 and core 7 belong to the A15 processor block. Three types of experiments were carried out:

- Dynamic voltage frequency scaling.
- Controlling the number of active cores.
- Duty cycle with idle-wait state.
- Performance evaluation.

3.3.1 *Dynamic voltage frequency scaling*

Many studies have recently been conducted aiming to reduce the power consumption of many-core processors based on various techniques. These techniques include varying the clock frequency, and supply voltage correspondingly while a task is being processed called DVFS.

In the first part of this experiment, voltage, current and power were measured on A7 and A15 power domains without any additional workload, with only Ubuntu 14.04 OS running.

Figure 3.3 and Figure 3.4 represent the voltage-frequency characteristics of A7 and A15 power domains in this experiment. It should be noted that below some frequency, voltage remains the same, but above this point, the voltage linearly increases. As an example, A7 has a voltage of 0.913V at frequencies 200MHz - 500MHz, meanwhile A15 has a voltage of 0.9125V at frequencies 200MHz - 700MHz. This experiment clarifies the voltage-frequency dependencies for A7 and A15 cores in Odroid-XU3 board.

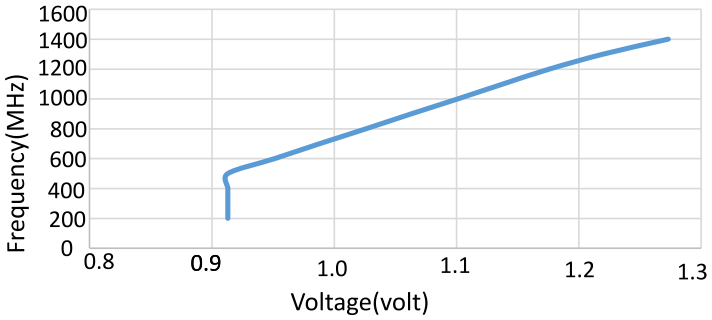


Figure 3.3: Cortex-A7 voltage-frequency characteristic.

cpufreq Linux governor provides the utility ***cpufrec-set***, which was used to change the frequency of all four cores in the domains of either A7 or A15.

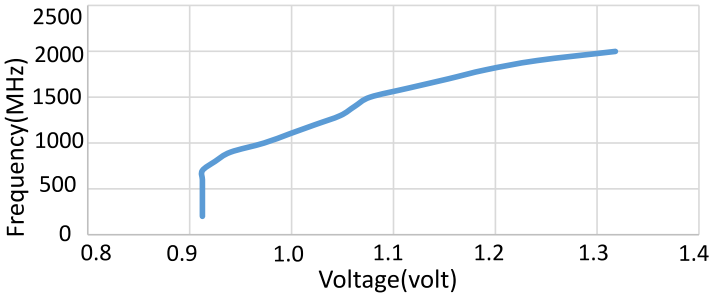


Figure 3.4: Cortex-A15 voltage-frequency characteristic.

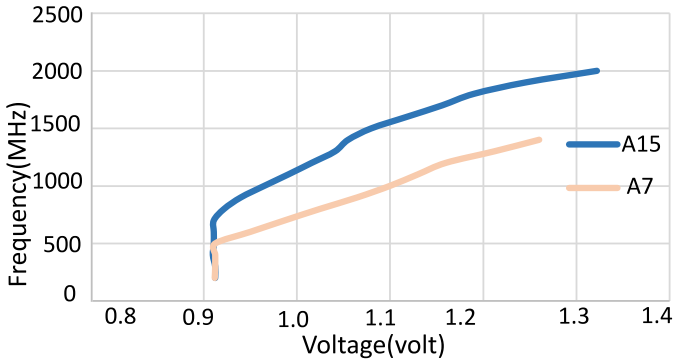


Figure 3.5: Cortex-A7 and Cortex-A15 voltage-frequency characteristic under 100% workload

For example, ***cpufrec-set -u 1200MHz -c 7*** sets maximum frequency 1200MHz for CPU core 7. Since it is possible to change the frequency only for all cores of the processor block at the same time, all four cores of A15 (4, 5, 6 and 7) will get the same frequency 1200MHz.

In the second part of the experiment, the same parameters were measured for each core with 100% loading. The workload was created by a custom stress test program, which has been written in C language. The program executes 50 million square-root operations. Without artificial delays in the code (like usleep function), this program creates 100% workload for a CPU core. Thread affinity was applied in order to execute the program on the specified CPU core. To bind the task to the CPU core ***taskset*** Linux command was used.

taskset SqrtStress -c 0 SqrtStress program will be executed on CPU core 0 (the first core of Cortex-A7 processor).

Experiments with the execution time of logarithm, addition, subtraction, multiplication and division operations gave the anticipated result. A15 was more than twice faster than A7 at the same frequency and almost three times faster at the maximum frequency. Unexpected results were received during experiments with the execution time of square-root operation. At the maximum frequency (2.0GHz) Cortex-A15 was just 1.2 times more productive than Cortex-A7 at the maxi-

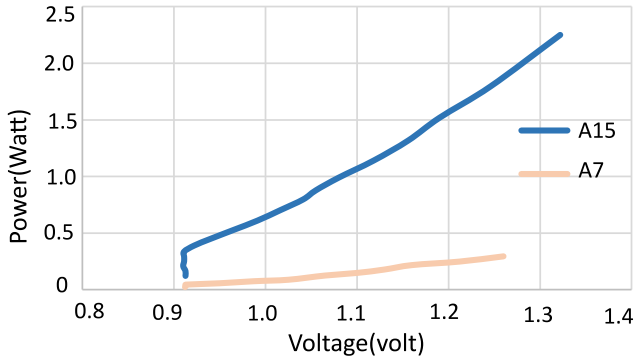


Figure 3.6: Cortex-A7 and Cortex-A15 voltage–power characteristic under 100% workload

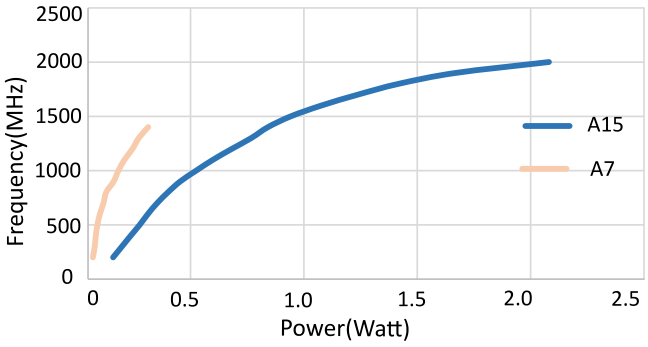


Figure 3.7: Cortex-A7 and Cortex-A15 power-frequency characteristic under 100% workload

mum frequency (1.4GHz), 10.9 seconds and 13.2 seconds for 50 million operations correspondingly. However, when the execution time was calculated at the same frequencies, A7 was faster than A15, for example at 1.0GHz frequency A15 core finished the task at 21.9 seconds, whereas A7 core required only 18.5 seconds and consumes a quarter of the power. The same trend was observed with sine and cosine functions.

In these experiments, it was observed that an A15 consumes four times or more power than an A7 when both are running at the same frequency, and up to an order of magnitude more power when both are running at the same voltage. Figure 3.8 shows the relationship between power consumption and the execution time for the two types of cores on the average running a range of different types of tasks.

Diagrams on Figure 3.5, Figure 3.6 and Figure 3.7 represent common trends for A7 and A15 processors independently of the calculation task. Diagrams on Figure 3.8 and Figure 3.9 depend on type of the operation. The line on diagram, which relates to the processor with better performance (for the predefined operation) will be situated below. In case of square-root calculations, A15 shows worse performance

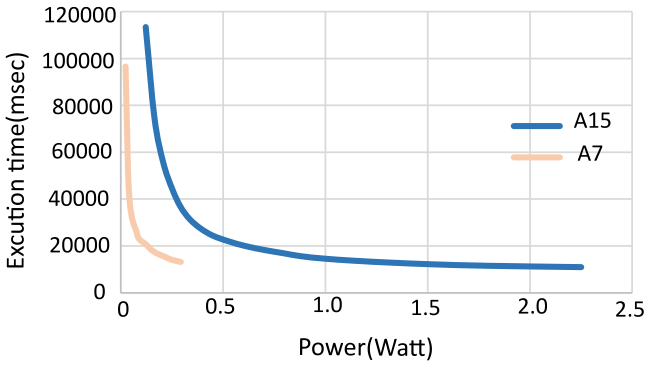


Figure 3.8: Cortex-A7 and Cortex-A15 power-execution time characteristic under 100% workload.

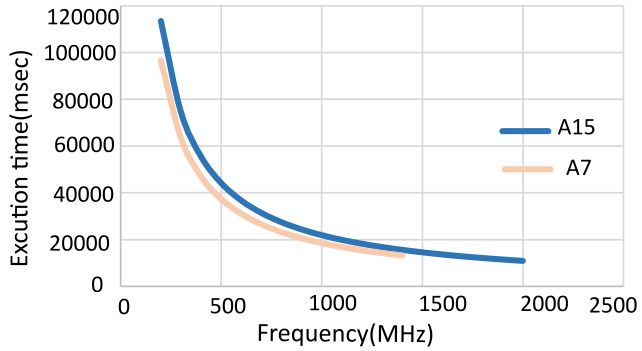


Figure 3.9: Cortex-A7 and Cortex-A15 frequency-execution time characteristic under 100% workload.

than A7 running at the same frequency, that is why A15 line is above the A7 line.

3.3.2 *CPU-power and number of active cores*

Recently reducing energy consumption has become a major concern for most computing systems. The connection between cores-power and number of active cores has been studied to calculate the power dissipation in many-cores systems. This experiment measures the same parameters while some of the cores in each block are disabled. It was carried out in order to investigate possible power and energy savings when the workload is not very high. Up to four A15 cores and up to three A7 cores can be disabled on Odroid-XU3. At least one A7 core must be running for the OS to be alive.

The following Linux command is used to disable a core:
echo 0 | sudo tee /sys/devices/system/cpu/cpu1/online
and to re-enable it again:
echo 1 | sudo tee /sys/devices/system/cpu/cpu1/online

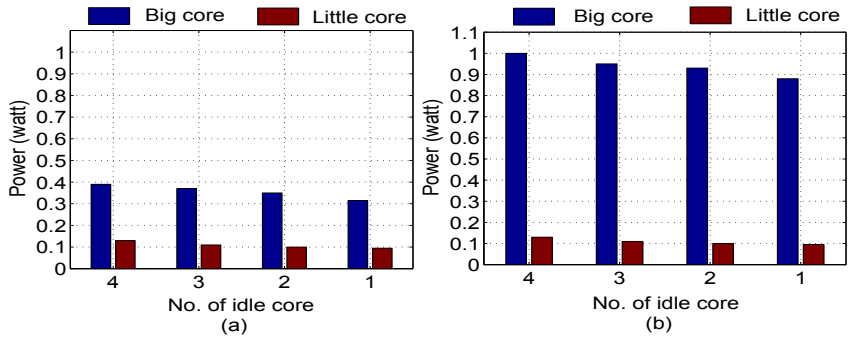


Figure 3.10: Experimental measurements of idle power by adopting Odroid-XU3 big.LITTLE platform (a) 1400MHz big.LITTLE; (b) 2000MHz big, 1400MHz LITTLE at 1400 MHz to 1 Watt at 2000 MHz.

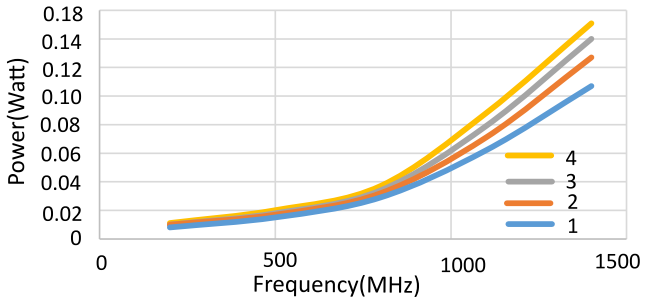


Figure 3.11: Power consumption of A7 domain with different number of active cores.

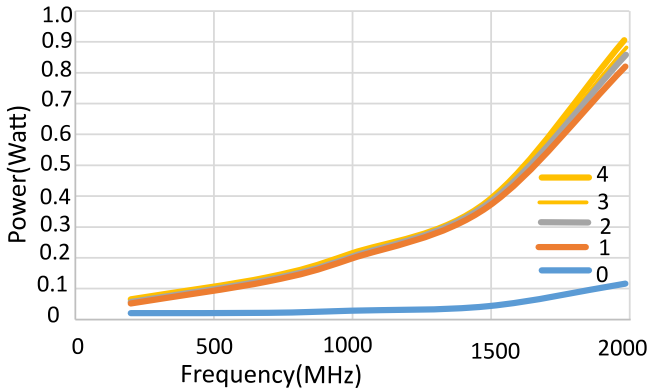


Figure 3.12: Power consumption of A15 domain with different number of active cores.

Figure 3.10 depicts the idle power measurements on Odroid-XU3 big.LITTLE platform for different core allocations and frequencies. The following two observations can be made. Firstly, with increasing number of inactive cores (big or LITTLE) the idle power consumption increases. As an example, when there is no additional workload and only OS is running and all four A15 cores are enabled, the idle power

of 4 big A15 inactive cores at 2000 MHz is 0.921 Watt, which drops to 0.888 Watt if one A15 core is disabled. If three A15 cores are disabled, A15 domain consumes 0.833W. However, when all four cores of A15 are disabled, the power consumption plummeted to 0.119W.

Secondly, the idle power is also dependant on the operating frequency. For instance, when parallel threads are allocated to LITTLE cores only, the idle power dissipation of 4 big inactive cores rises from 0.39 Watt at 1400 MHz to 1 Watt at 2000 MHz. Figure 3.11 and Figure 3.12 show the relation of power consumption of A7 domain and A15 domain with different number of active cores respectively.

Idle power contributes to unuseful energy consumption, essentially reducing the battery operational life time. To reduce the idle power, the traditional approach is to use power gating [5].

3.3.3 *Duty cycling with idle-wait state*

Figure 3.13 and Figure 3.14 show the experimental results for different CPU loadings. These results represent the power consumption and execution time of 50 million square root operations. The usleep function (C language) was used after every 100000 operations to put the thread into sleep state. We can create necessary CPU loading from about 0% to 100% by passing different arguments in the usleep function. As seen from the results duty cycling with idle-wait state is highly energy inefficient. The energy consumption required for calculations remains roughly the same, whereas total energy consumption increases when CPU loading is decreased. It is more efficient to execute the task as fast as possible than using this method of duty cycling.

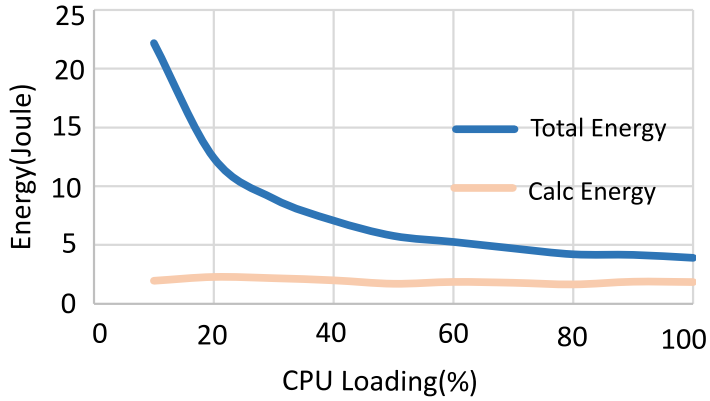


Figure 3.13: Dependence of total energy and calculation energy on A7 CPU loading

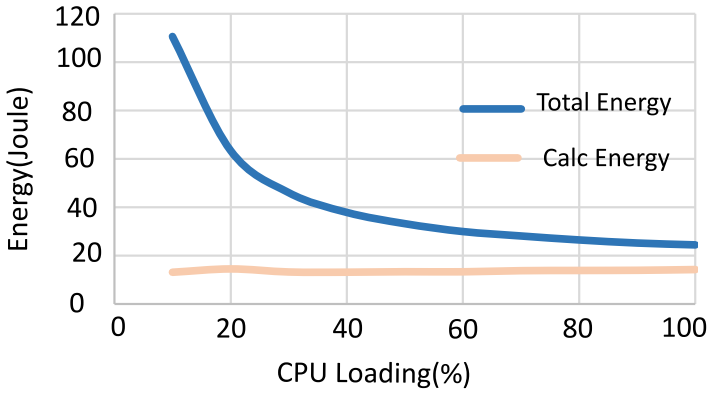


Figure 3.14: Dependence of total energy and calculation energy on A15 CPU loading

3.3.4 *Performance Evaluation*

In this section, the experiments present the application behaviour on a heterogeneous architecture and provides realistic values of the performance for the ARM big.LITTLE processors. This processor has 4 LITTLE cores and 4 big that can run at 13 and 19 different frequencies as described in Section 3.2 which lead to more than 4004 possible frequency and core allocation for each application. To collect performance characterization data, Ubuntu 14.04 kernel with performance counter tool has been used.

Figure 3.15 shows the real performance measurements (in terms of instruction per cycle IPC) for various thread-to-core frequency allocations. Two observations can be made. Firstly; memory-intensive applications typically have lower IPC than applications CPU heavy, as expected. Secondly, IPC is approximately a constant level in case of running big cores across the frequency range.

3.4 SUMMARY AND CONCLUSION

Several experiments were carried out in order to find out power, frequency and performance interplays on Odroid-XU3 board. Dynamic-frequency scaling is a very useful technique that can be applied for the adjustment to the system loading. The idle-wait state is very inefficient and it should be avoided whenever possible and not used to duty cycle operations in order to save power. Core disabling provides the possibility for substantial power and energy savings when the loading is low. These experiments give a deeper understanding of the benefits of heterogeneous architectures. The tradeoffs between performance and energy-consumption obtained during the experiments are very useful for the runtime modelling in order to achieve optimal system operation. Some of the results of these experiments, and further char-

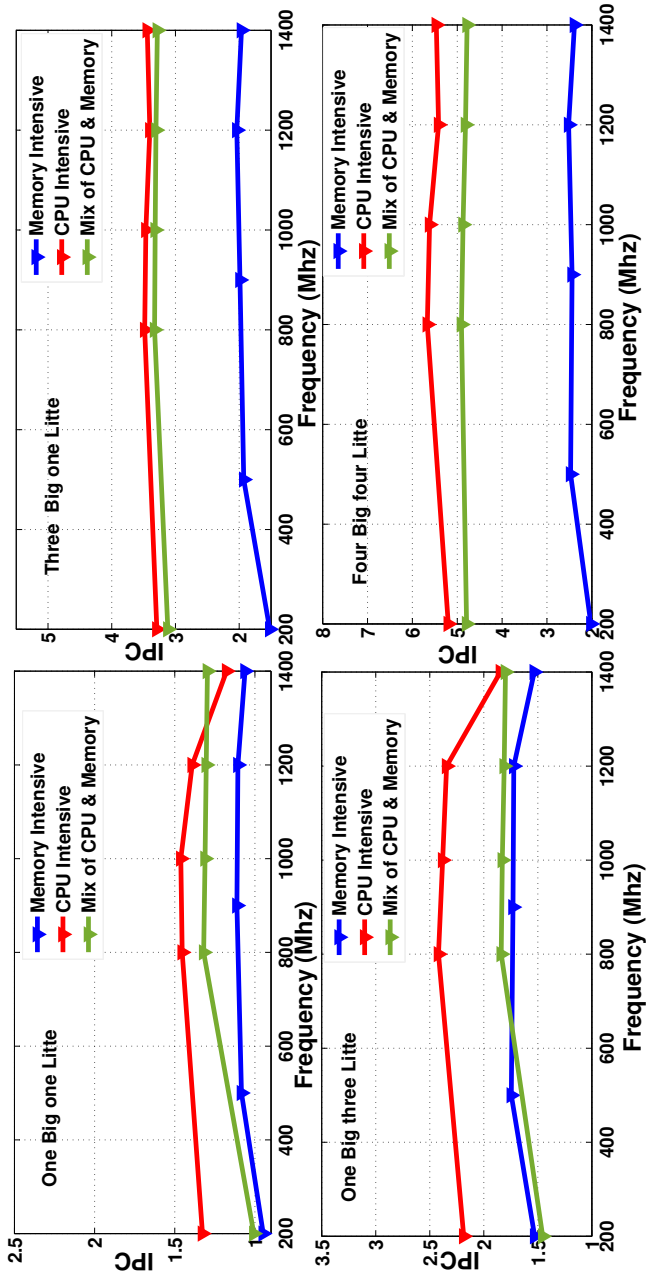


Figure 3.15: Total IPC for different workloads types

acterization experiments based on the same methods will be used in subsequent sections to help derive system models and workload classification schemes.

4

MODEL-BASED RUNTIME MANAGEMENT OF CONCURRENT WORKLOADS

As stated in Chapter 1, finding energy efficient system configurations can be a challenging problem because of the heterogeneity found in both hardware and software in modern embedded systems.

In this chapter, a novel model-based runtime optimization approach to maximize power normalized performance considering dynamic variation of workload and application scenarios is presented. Using real experimental measurements on an Odroid XU-3 heterogeneous platform with a number of PARSEC benchmark applications, we model power normalized performance (in terms of IPS/Watt) underpinning analytical power and performance models, derived through (Multivariate linear regression (MLR)).

The rest of this chapter is organized as follows. Section 4.2 shows the experimental environment, the configuration of system used in the experiment and the applications. The model-based approach is described in Section 4.3, which is the first step to produce power, performance and power normalized performance models for ARM heterogeneous processors using MLR. Section 4.4 presents experimental results. Section 4.5 concludes the work in this chapter.

4.1 INTRODUCTION

Running multiple software applications concurrently on the same platform is rapidly becoming the norm of modern computing, as are system platforms with higher complexity that cater to such uses. This increasing system complexity in both hardware and software emphasizes a major challenge for computing systems, especially mobile and embedded systems, namely the performance-energy tradeoff [55]. This has led to techniques for mitigating power consumption and performance degradation concerns. Power-aware platform design including many-core architectures with the provision for dynamic task mapping and DVFS have been the preferred tools for system designers over the years [15, 68].

In the past few years, there have been various studies in energy efficiency in embedded systems, as shown in Table 4.1. A power control approach for many-cores processors executing single application has been proposed in [68]. This approach has three layers of design features also shown by other researchers: firstly, adjusting the clock frequency of the chip depending on the power budget; secondly, dynamically group cores to run the same applications (as also shown in [127, 82]), and finally, modify the frequency of each core group (as also shown in [123, 95]).

Table 4.1: Features and limitations of the existing approaches.

Approach	Application	Validation	Key method
[68]-[95]	Single app	Simulation	Offline optimization, DVFS
[33]	Single app	Simulation	Runtime optimization, task mapping, DVFS
[66]	Single app	Simulation	Offline optimization, CPU only sched, DVFS
[32]	Single app	Simulation	Offline optimization, task mapping, DVFS
[128]	Single app	Implementation	Offline optimization, task mapping, DVFS
[74]	Single app	Implementation	Runtime optimization, FPGA only, DVFS
Proposed	Concurrent	Implementation	Runtime optimization, task mapping, DVFS

Among others, Goraczko et al. [33] and Luo et al. [66] proposed DVFS approaches with software task partitioning and mapping of single applications using a linear programming (LP) based optimization during runtime to minimize the power consumption. Goh et al. [32] proposed a similar approach of task mapping and scheduling for single applications described by synthetic task graphs.

Several other works have also shown power minimization approaches using practical implementation of their approaches on heterogeneous platforms. For example, Yang et al. [128] presented an adaptive power minimization approach using runtime linear regression-based modeling of the power and performance tradeoffs. Using the model, the task mapping and DVFS are suitably chosen to meet the specified performance requirements. Nabina and Nunez-Yanez [74] presented a similar DVFS approach for field-programmable gate array (FPGA)-based video motion compensation engines using runtime measurements of the underlying hardware.

A number of research have also shown analytical studies using simulation tools like gem5, together with McPAT [78, 8] for single applications. These works have used DVFS, task mapping, and offline optimization approaches to minimize the power consumption for varying workloads.

When these applications run concurrently the workloads generated by the hardware may vary significantly compare to that of a single application. Hence, energy efficiency cannot be automatically guaranteed using the existing approaches that are agnostic of concurrent application workloads and behaviors (Table 4.1).

In this work, we address the limitations of the above works and propose an adaptive approach, which monitors application scenarios at runtime. The aim is to determine the optimal system configuration such that power normalized performance can be maximized at all times. We adopt an experimental approach depending on profiling real power consumption and performance measurement for single and concurrent applications. For the first time, our study reveals the impact of parallelism in different types of heterogeneous cores on performance, power consumption and power efficiency in terms of IPS/Watt, which is the same as the number of instructions per unit of energy [115]. The major contributions of this chapter are as follows:

- Investigate the CPU performance in terms of instructions per cycle (IPC) and power tradeoffs using directly measured values from the performance counters.
- Use real application benchmarks, suitably chosen from a pool of available applications, including CPU intensive, memory intensive, and other combinations.
- Use MLR to model power and performance tradeoffs expressed as IPS/Watt.

- Maximize IPS/Watt for single and concurrent application scenarios using low-cost runtime adaptation algorithm.

To the best of our knowledge, this is the first runtime optimization approach for concurrent applications, practically implemented and demonstrated on a heterogeneous many-core system.

4.2 SYSTEM ARCHITECTURE AND APPLICATIONS

In this section, we describe the platforms, workload applications and performance counters used in this investigation. We studied heterogeneous parallel processing platform, which provide all the performance counters and power monitors we need for the methodology described in the previous section. We chose standard benchmark application workloads which provide a variety of degrees of concurrency and memory access and CPU usage scenarios. The hardware platform, PARSEC workloads applications and performance counters are further detailed below.

4.2.1 *Heterogeneous System*

The popularity of heterogeneous architectures, containing two or more types of different CPUs is growing [55]. These systems offer better performance and concurrency, however it is necessary to ensure optimal power and energy consumption. The Odroid-XU3 board supports techniques such as DVFS, affinity and core disabling, commonly used to optimize system operation in terms of performance and energy consumption [2] [101].

The Odroid-XU3 board is a small eight-core computing device implemented on energy-efficient hardware. The board can run Ubuntu 14.04 or Android 4.4 operating systems. The CPU of Odroid-XU3 platform is described in Chapter 3.

4.2.2 *Applications Workload*

The PARSEC benchmark suite attempts to represent both current and emerging workloads for multiprocessing hardware. It is a commonly used benchmark suite for evaluating concurrency and parallel processing. We therefore use PARSEC on the Odroid-XU3 platform, whose heterogeneity can be representative of different design choices that can greatly affect workloads. PARSEC applications exhibit different memory behaviours, different data sharing patterns, and different workload partitions from most other benchmark suites in common use [10].

Three applications (*ferret*, *bodytrack* and *fluidanimate*) are selected to represent CPU intensive, memory intensive, and CPU with memory

intensive respectively. Such a classification reduces the effort of model characterisation for combinations of concurrently running applications.

4.3 PROPOSED APPROACH

Our method studies single and concurrent application workloads being executed on heterogeneous hardware platform with parallel processing facilities, and derive optimal runtime management decisions from the results of power/performance models. Based on these models a runtime adaptation is derived to determine the most energy-efficient system configurations for different application scenarios.

4.3.1 *Modeling Power/Performance Tradeoffs*

Systems with large scale concurrency and complexity, e.g. computation systems built upon architectures with multiple and increasingly many processing cores with heterogeneity among the components, are becoming more popular and common-place . The hardware motivations are clear, as concurrency scaling can help delay the potential saturation of Moore's Law with current and future CMOS technology and better use the opportunities provided by the technology scaling. In this environment, software designs are increasingly focused towards greater concurrency and mapping to such many-core hardware [61, 84].

In this section, we develop runtime power and performance models using MLR. These models and their hypotheses are further detailed below.

4.3.1.1 *Power Model*

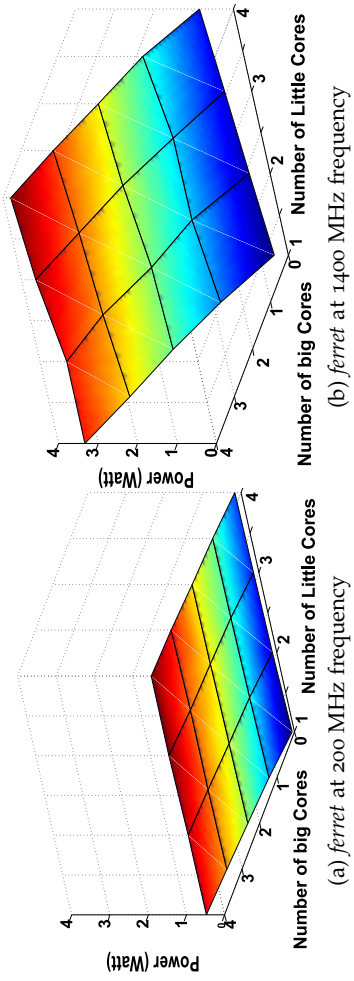
Many studies have recently been conducted aiming to reduce the power consumption of many-core processors based on various techniques. These techniques include varying the clock frequency and supply voltage correspondingly, while a task is being processed.

Power consumption can be divided into two parts: dynamic power and static power. Dynamic power is the switching overhead in transistors, so it is determined by runtime events. Static power is mainly determined by circuit technology, chip layout and operating temperature [78].

Experiments with the Odroid-XU3 platform were carried out in order to examine its power consumption under different operation frequencies and voltages. Using the same methods presented in Chapter 3, the frequency of each block can be changed independently using utility programs and the system scales the operating voltage of the block to fit the chosen frequency. The eight cores in the board are numbered as follows: core 0, core 1, core 2 and core 3 belong to the A7

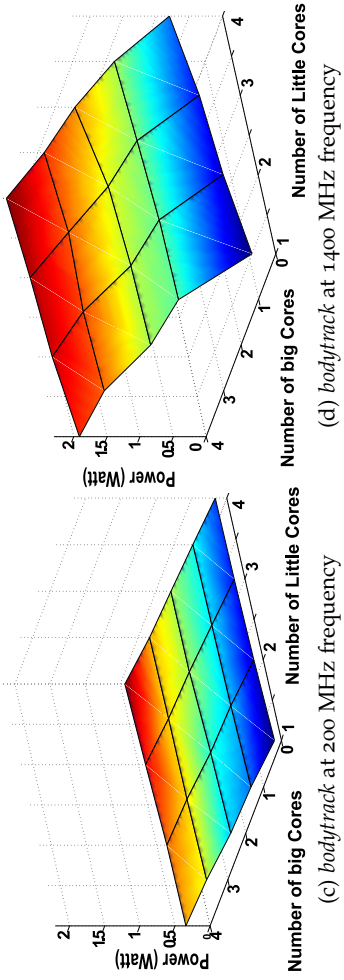
(LITTLE) processor block, core 4, core 5, core 6 and core 7 belong to the A15 (big) processor block. Which means, Odroid-XU3 architectures have 20 different core configurations for each frequency.

Figure 4.1 presents the power consumption of *ferret* and *bodytrack* applications for different thread-to-core allocations. As expected the following two observations can be made. Firstly, the power consumption increases as more cores are allocated for the given application. Secondly, the power is also dependant on the operating frequency which shows the impact of DVFS. For instance, when parallel threads are allocated to big cores only, the total power dissipation of 4 big cores rises from 0.52 Watt at 200 MHz to 3.815 Watt at 1400 MHz for the *ferret* application.



(a) *ferret* at 200 MHz frequency

(b) *ferret* at 1400 MHz frequency



(c) *bodytrack* at 200 MHz frequency

(d) *bodytrack* at 1400 MHz frequency

Figure 4.1: Total power for *ferret* and *bodytrack* applications at 200 MHz and 1400 MHz frequencies.

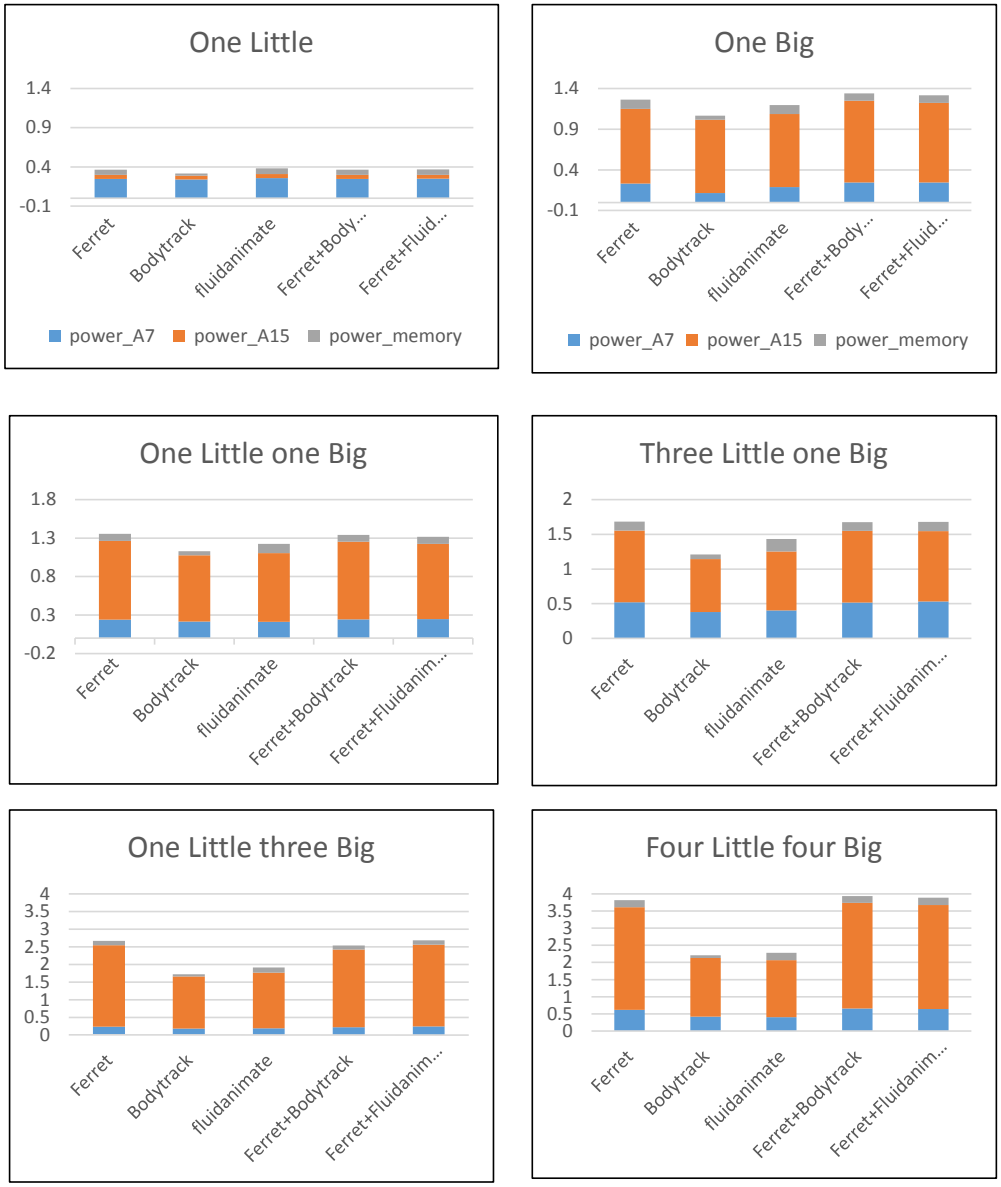


Figure 4.2: Total power for single and concurrent applications in different configuration running at 1400 MHz.

Table 4.2: Single Application Power Models.

Freq.	ferret				bodytrack				fluidanimate				Volt.	Volt.
(MHz)	α_{A7}	α_{A15}	ϵ_1	RS	α_{A7}	α_{A15}	ϵ_1	RS	α_{A7}	α_{A15}	ϵ_1	RS	A7	A15
200	10.0e-11	5.2e-10	0.112	0.97	12.6e-11	3.6e-10	0.08	0.98	2.5e-11	3.8e-10	0.14	0.94	0.91	0.91
800	8.1e-11	4.7e-10	0.182	0.98	3.2e-11	2.9e-10	0.22	0.94	2.2e-11	3.4e-10	0.20	0.93	1.00	0.91
1000	7.6e-11	4.7e-10	0.245	0.98	3.2e-11	2.6e-10	0.33	0.93	2.5e-11	2.7e-10	0.31	0.93	1.05	0.94
1200	7.9e-11	4.8e-10	0.27	0.98	3.0e-11	2.5e-10	0.43	0.92	1.8e-11	2.9e-10	0.48	0.91	1.13	0.99
1400	8.1e-11	4.5e-10	0.3	0.97	3.5e-11	2.5e-10	0.52	0.93	2.9e-11	2.6e-10	0.61	0.92	1.23	1.04

Table 4.3: Concurrent Application Power Models.

Freq.	Ferret+Bodytrack				Ferret+Fluidanimate				Volt.	Volt.
(MHz)	α_{A7}	α_{A15}	ϵ_1	RS	α_{A7}	α_{A15}	ϵ_1	RS	A7	A15
200	1.3e-10	5.5e-10	0.08	0.99	7.0e-11	4.7e-10	0.13	0.99	0.91	0.91
800	3.6e-10	1.9e-09	0.15	0.99	5.0e-11	4.7e-10	0.25	0.98	1.00	0.91
1000	4.73e-10	2.475e-09	0.1176	0.95	7.3e-11	4.6e-10	0.79	0.98	1.05	0.94
1200	4.587e-10	2.675e-09	0.12	0.96	7.6e-11	4.6e-10	0.80	0.99	1.13	0.99
1400	7.6e-10	3.8e-09	0.19	0.98	3.3e-12	4.6e-10	0.80	0.95	1.23	1.04

Figure 4.2 depicts the power distribution between the cores and the memory for different application scenarios. The following three observations can be made from the figure.

- Firstly, it is clearly seen that the total power consumption for A15 and A7 for a CPU intensive application (*ferret*) is higher than for a memory intensive application (*fluidanimate*).
- Secondly, in cases where threads are allocated to LITTLE cores only, the power of A15 cores is idle power and the total power dissipation for the big cores rise up from 0.39 Watt at 200 MHz to 2.22 Watt at 1400 MHz which shows the impact of DVFS on idle power.
- Finally as can be seen in Figure 4.2 the memory power is much smaller than the combined A7 and A15 processors power. The variation of memory power depends on applications and the execution scenarios.

Multivariate linear regression is used to determine the relation between power and the number of cores, types of cores (big, LITTLE), and frequency. This relation is hypothesized to fit the following expression based on theory [78]:

$$P = \sum_{i=1}^{K} \alpha_i x_i V_i^2 f_i + \epsilon_1(x), \tag{4.1}$$

where the first term($\sum_{i=1}^{K} \alpha_i x_i V_i^2 f_i$) of (4.1) represents the dynamic power, K is the number of group of cores, and the second term ($\epsilon(x)$) represents static and dynamic power of memory, leakage, and interconnect power. Coefficient α_i includes activity factor. In the case of Exynos 5422 (i.e. Odroid big.LITTLE) (4.1) can be approximated as:

$$P = \alpha_{A7} x_{A7} V_{A7}^2 f_{A7} + \alpha_{A15} x_{A15} V_{A15}^2 f_{A15} + \epsilon_1, \tag{4.2}$$

where x_{A7} is the number of LITTLE cores, x_{A15} is the number of big cores, V_{A7} and V_{A15} are the voltages of A7 and A15 cores respectively, f_{A7} and f_{A15} represent the frequencies for A7 and A15 respectively; $\alpha_{A7}, \alpha_{A15}$ are MLR coefficients. Tables 4.2 and 4.3 show the result of MLR.

All MLR procedures for these coefficients have returned root-squared (or RS) coefficient of determination values of 0.92 or better, confirming the applicability of this hypothesis.

4.3.1.2 *Performance Model*

Existing studies of performance in the many-core era based on Amdahl's law or Gustafson's law do not perfectly handle the behaviour

of a multi-threaded applications on heterogeneous multi-core platforms like Odroid-XU3. On the other hand, performance modeling of many-core heterogeneous systems by simulation techniques tend to be computationally intensive.

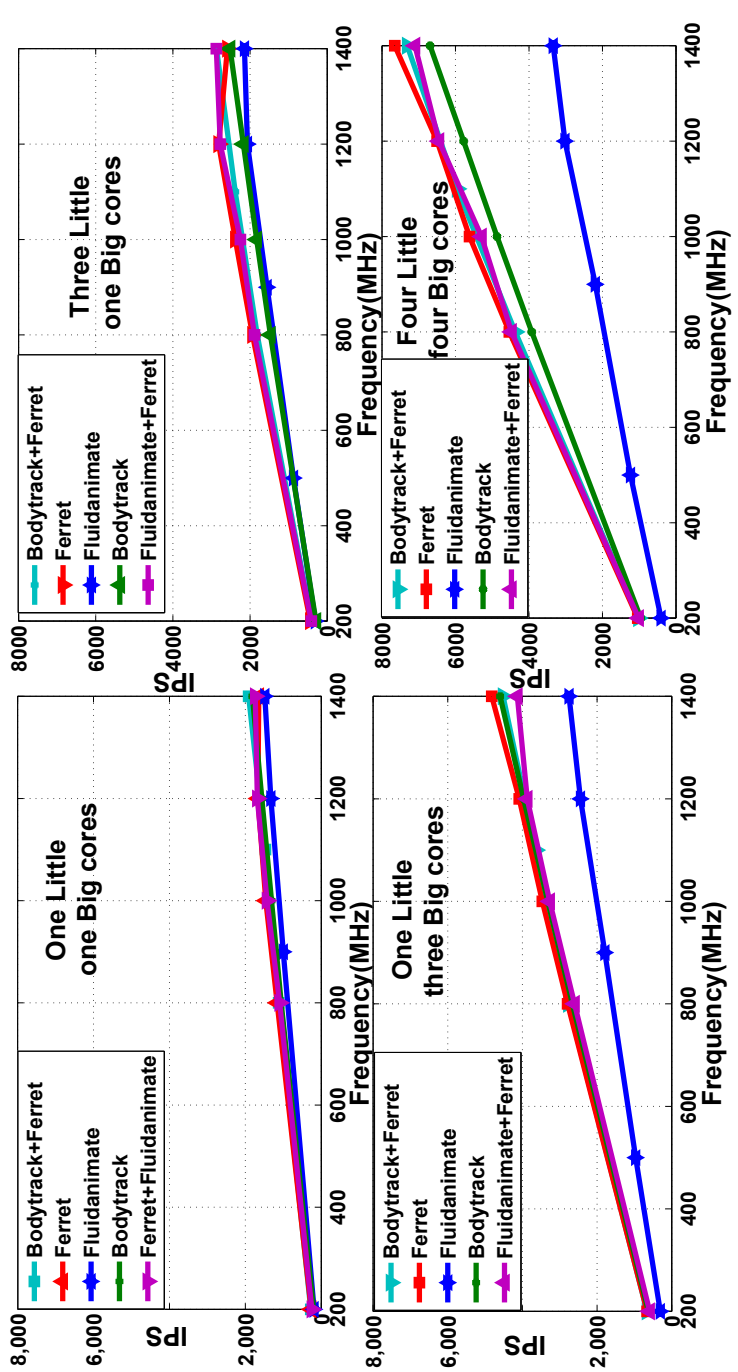


Figure 4.3: Total IPS for single and concurrent applications obtained from performance counters.

Table 4.4: Single Application Performance Models.

Freq.	ferret				bodytrack				fluidanimate			
(MHz)	α_{A7}	α_{A15}	ϵ_2	RS	α_{A7}	α_{A15}	ϵ_2	RS	α_{A7}	α_{A15}	ϵ_2	RS
200	0.43	0.87	0.15	0.99	0.23	0.97	0.14	0.99	0.20	3.87	0.6	0.88
800	0.46	0.94	0.14	0.99	0.23	0.97	0.14	0.99	0.21	3.43	0.52	0.91
1000	0.37	0.87	0.37	0.96	0.23	0.97	0.12	0.99	0.21	2.74	0.54	0.89
1200	0.43	0.91	0.16	0.99	0.25	0.98	0.15	0.99	0.21	2.92	0.52	0.89
1400	0.40	0.96	0.004	0.99	0.24	0.91	0.21	0.99	0.17	2.69	0.52	0.90

Table 4.5: Concurrent Application Performance Models.

Freq.	**Ferret+Bodytrack**				**Ferret+Fluidanimate**			
(MHz)	α_{A7}	α_{A15}	ϵ_2	RS	α_{A7}	α_{A15}	ϵ_2	RS
200	0.35	0.86	0.26	0.99	0.45	0.80	0.06	0.98
800	0.40	0.93	0.16	0.99	0.49	0.89	0.01	0.99
1000	0.38	0.93	0.13	0.98	0.43	0.86	0.13	0.99
1200	0.38	0.93	0.13	0.99	0.45	0.87	0.07	0.99
1400	0.34	0.93	0.11	0.99	0.41	0.84	0.10	0.99

In this work, the experiments present the application behaviour on a given architecture and provides realistic values of the performance for the ARM processors. We use Ubuntu 14.04 kernel with our performance counter tool designed to gather processor performance events, namely, instructions retired (retired branches, loads, stores, etc.) in order to depict the behaviour of a thread execution in the heterogeneous system.

Figure 4.3 shows the real performance measurements in terms of (IPS) for various thread-to-core allocations and frequencies. It can be observed that a memory intensive operation on its own has lower IPS than CPU-heavy operation, as expected. However, when running these types together, the overall IPS is high. The clock-independent performance metric is instructions per cycle (IPC) that can be derived from IPS by knowing the clock frequency. The performance counters provide the number of total instructions retired (retired branches, loads, stores, etc.).

IPC is approximately a constant level in case of running more big cores than LITTLE cores when increasing the frequency for the memory intensive applications. However, the experimental results of CPU intensive applications show that performance slightly decreases at 1400 MHz when we use more LITTLE cores. This shows that expected trends from theory may not be confirmed with experimental data in every case. The importance of the experimental approach must be recognized.

From the data in the above figures, it is apparent that the performance increases linearly with the number of big and LITTLE cores. Considering that models for runtime use should in principle be as simple as possible, we hypothesize that the relation between IPC and the numbers of group of cores in heterogeneous systems can be approximated by the following expression:

$$\mathrm{IPC} = \sum_{i=1}^{K} \alpha_i x_i + \epsilon_2(x), \tag{4.3}$$

where the first term ($\sum_{i=1}^{K} \alpha_i x_i$) represents the frequency-independent performance components that depend on architectural configuration, K is the total number of voltage islands (i.e. group of cores), and $\epsilon_2(x)$ is the error term. In the case of Exynos 5422 (i.e. Odroid big.LITTLE) (3) can be expressed as:

$$\mathrm{IPC} = \alpha_1 x_1 + \alpha_2 x_2 + \epsilon_2, \tag{4.4}$$

where α_1, α_2, and ϵ_2 are coefficients which need to be determined for each operating frequency. Using MLR, their values have been obtained as shown in Table 4.4 and 4.5, and x_1, x_2 are the numbers of

LITTLE and big cores respectively. All models have R-squared values greater than 0.95 showing the applicability of this model hypothesis.

4.3.1.3 *Power Normalized Performance Model*

Based on the power and performance outcomes, power normalized performance experiments indicate that the optimal system configuration corresponds to the highest IPS/Watt value for ARM heterogeneous processors. Figure 4.4 presents the experimental data of IPS/Watt for different application scenarios and architectural configurations. In the case of a single application, *bodytrack* (CPU and memory intensive) exhibited the highest IPS/Watt. This can be explained by its high IPS (Figure 4.3) and the lowest power (Figure 4.1). The IPS/Watt of *bodytrack* shows an increasing trend with higher number of cores allocated; in the case of four LITTLE four big it has the maximum value of 3.8×10^9 IPS/Watt, when operating at 800 MHz. However, with higher frequencies the power consumption increases, which reduces its IPS/Watt. Similar observations can be made for the other single application scenarios.

To investigate performance, power and power normalized performance for different CPU- and memory-intensive applications when running concurrently, another set of experiments were carried out using *fluidanimate*, *ferret* and *bodytrack* workloads. For example, in the case of concurrent applications the *bodytrack+ferret* (which is dominated by CPU intensive routines) shows higher IPS/Watt when compared with *fluidanimate+ferret* (which is a combination of CPU and memory intensive routines). To explain this further, Figure 4.4 depicts the individual and concurrent application scenarios of *bodytrack* and *ferret*. As can be seen, when these two similar workloads are combined as a concurrent application it shows higher IPS/Watt with increasing number of cores.

Figure 4.5 and Figure 4.4 present the experimental data on IPS/Watt. Maximum IPS/Watt can be found by searching through the range of operating frequencies and core configurations. Figure 4.4 shows that CPU-intensive applications behave similarly in single and concurrent situations. A memory-intensive operation on its own has lower IPS and power than CPU-heavy, as expected.

The above models have been derived from the offline characterization data. In order to achieve the optimal mode during normal device operation, runtime adaptation has to be used instead. To simplify runtime adaptation MLR is used, which exploits the same set of runtime observations to derive/predict power and performances with reasonable accuracy. The power and performance expressions in

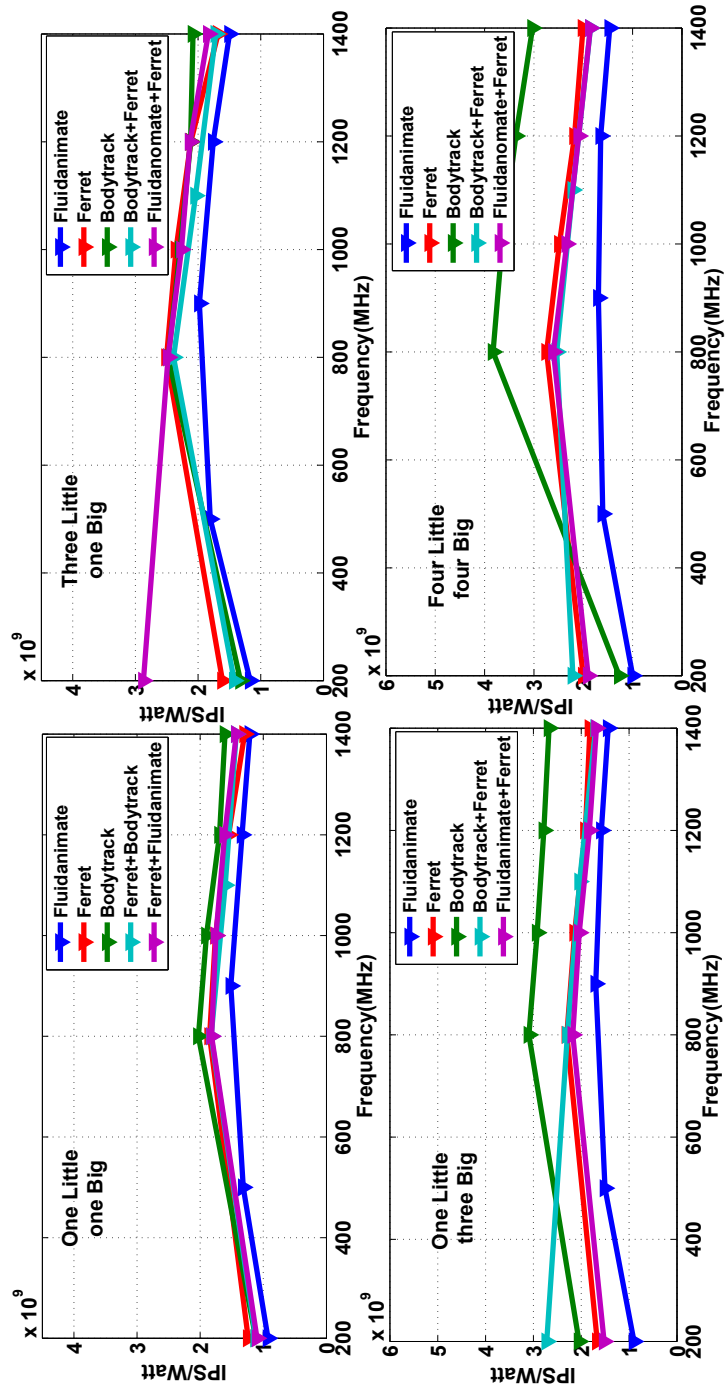


Figure 4.4: Total IPS/Watt for single and concurrent applications in different frequencies.

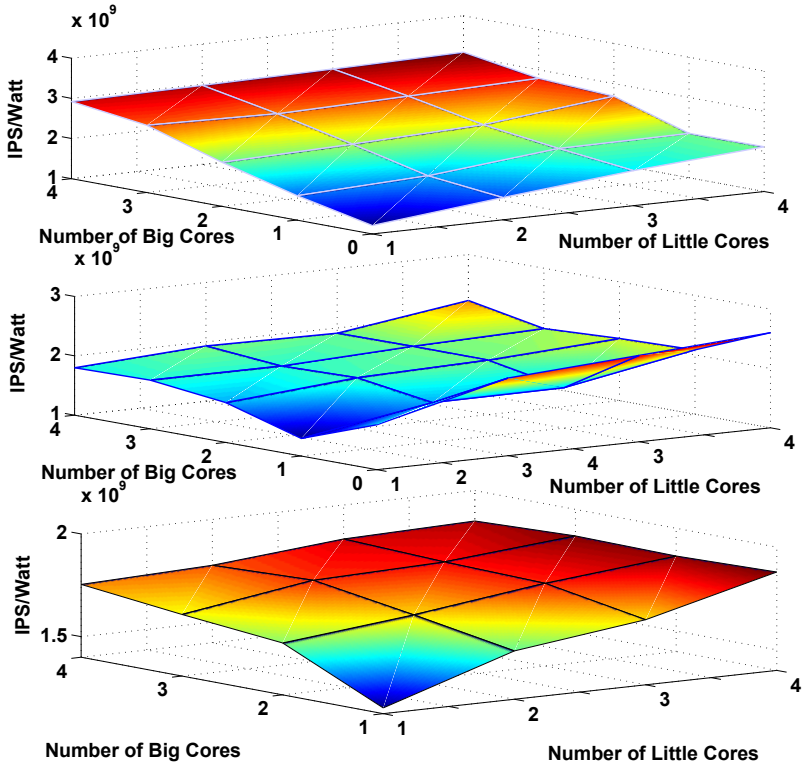


Figure 4.5: Total power normalized performance for different core-allocations at 1400 MHz for *bodytrack*, *ferret* and *bodytrack+ferret* applications.

(4.2) and (4.4) can be combined into a MLR expression as shown in the following:

$$\mathbf{D}^T = \mathbf{X}^T\mathbf{A} + \mathbf{E}^T, \tag{4.5}$$

where $\mathbf{D}$ is the determinant vector formed of power and performance as shown in (4.2) and (4.4); $\mathbf{X}$ is a predictor vector for the given determinant; $\mathbf{A}$ is the MLR coefficient vector for the given determinant ($a_i \in A$ where a_i is a coefficient expression in (4.2) including $V_i^2 f_i$ terms, i.e. $a_i = \alpha_i V_i^2 f_i$) and finally $\mathbf{E}$ is the error terms vector. Tables 4.2-4.5 show the MLR determinant values for different operating configurations (including both single and concurrent applications scenarios).

Other set of experiments are designed to show how the proposed runtime behaves when controlling different workload types and to validate our models by using another applications (*canneal*, *freqmine*, and *streamcluster*) from PARSEC benchmarks. Figure 4.6 shows the experimental results of power normalized performance which is expressed in terms of IPS/Watt compared to the results obtained by the proposed models.

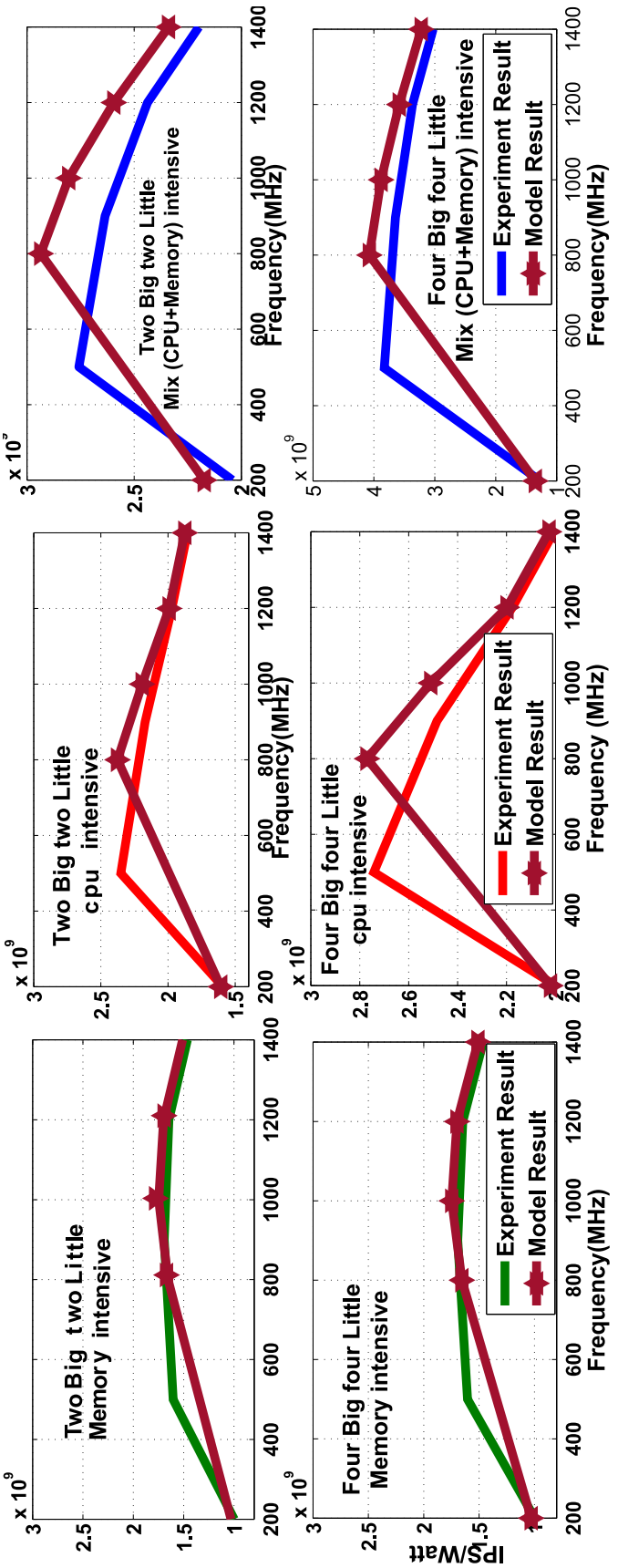


Figure 4.6: Total IPS/Watt for different workloads types.

In the rest of the chapter the models are used in the proposed governor to derive the optimal DVFS and core allocation decisions.

Algorithm 4.1 Runtime system adaptation algorithm to generate maximum IPS/Watt.

1: **procedure** Maximize IPS/Watt
2: input: Application scenario
3: output: System configuration {number of A15 cores, number of A7 cores, operating frequencies}
4: initialize: $(\rho_{max} \leftarrow 0)$
5: **for** $f \in [f_{min}, f_{max}]$ **do**
6: Collect power/perf. readings for 200 ms
7: Use MLR and (4.1) to find power model for f
8: Use MLR and (4.2) to find IPC model for f
9: IPS $\leftarrow$ IPC $\cdot f$
10: $\rho \leftarrow$ IPS/P
11: **if** $\rho > \rho_{max}$ **then**
12: $\rho_{max} \leftarrow \rho$
13: **end if**
14: **end for**
15: **return** system config. corresponding to ρ_{max}
16: **end procedure**

4.3.2 *Modelling offline and online*

These modelling methods can be applied either offline during design-time or online during runtime. For design-time, a number of applications belonging to different workload types can be pre-modelled establishing a model library which could help save time during runtime optimization. And if necessary, during runtime, more accurate models can be derived for unknown application workloads using the same methods presented in this section. Algorithm 4.1 describes how MLR may be used to derive power normalized performance models during runtime.

4.3.3 *Runtime Adaptation*

The runtime adaptation is aimed at finding optimal task mapping and DVFS control in the presence of various types of concurrent applications (CPU or memory intensive). It is based on gradually building a library of the power and performance models, obtained during a normal

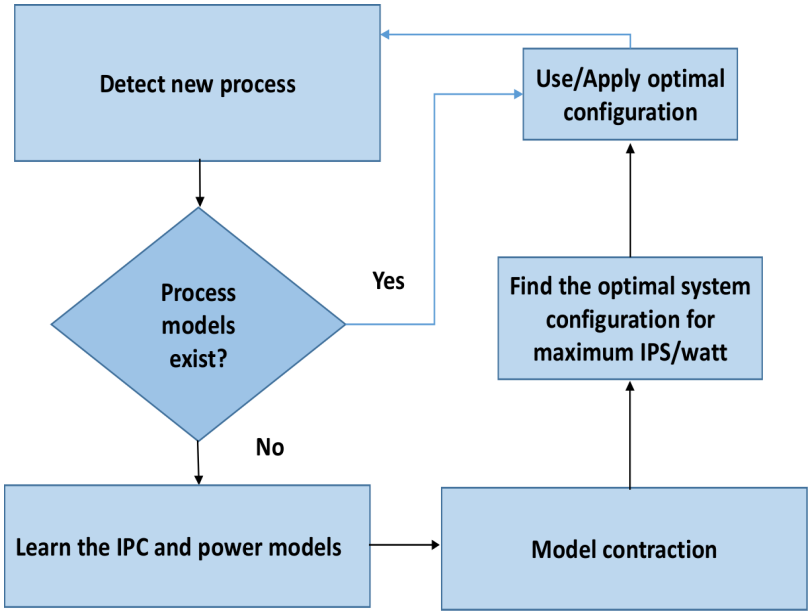


Figure 4.7: Flowchart of the proposed runtime adaptation cycle.

device operation. The flowchart of the proposed approach is shown in Figure 4.7.

The algorithm detects the arrival of a new application process and looks up its model in the library. If the model already exists in the library, it can be immediately used to compute the optimal point of operation, otherwise the learning process activates. As shown in Algorithm 4.1, the procedure collects power readings and performance counters for a number of 200 ms time intervals sweeping across the entire range of frequencies and uses MLR to derive the power and performance models according to Section 4.3.1. Once the models are obtained for all DVFS points, we can find an optimal system configuration by maximizing the IPS/Watt metric.

In the current version of the runtime control, the combined behaviour of concurrently running applications is captured by separate models. We are aware of the exponential growth when all combinations of applications are eventually stored in the library. This problem is addressed by compacting the models by similarity. The applications of the similar type tend to have similar values of $\alpha_1, \alpha_2, \epsilon_1, \epsilon_2$, hence these coefficients can be checked against a given threshold ε. The models within the threshold are combined and stored as a single model.

4.4 EXPERIMENT RESULTS

The proposed runtime adaptation algorithm has been used in a number of examples with PARSEC benchmark programs running on the Odroid-XU3 platform, and the resulting IPS/Watt metric has been compared with common scheduling approaches such as those used by the Linux ondemand governor [75]. The example execution cases include running the three benchmark programs on their own in a sequential way and running *ferret* with either *bodytrack* and *fluidanimate* in two-app concurrent situations. In all these cases, using the proposed algorithm resulted in IPS/Watt improvements (up to 125% in the case of *ferret* application) as shown in Figure 4.8. From Figure 4.8 the following observations can be made. Firstly, it is clearly seen that the improvements in IPS/Watt for CPU and memory intensive applications are approximately the same in the case of single applications. Secondly, in the case of concurrent applications the improvement in IPS/Watt is less than for single applications. Finally, the lowest improvement is recorded in the *bodytrack*-only case, where, compared to ondemand, an improvement of 26% can be observed.

The proposed runtime adaptation approach outperforms the ondemand governor with default Linux scheduler for the given applications. This can be explained as follows. The default Linux scheduler sets the number of parallel threads of these applications equal to the number of available cores. With the allocated cores the ondemand governor downscales the operating frequency when the CPU usage is lower than pre-defined thresholds points. However, when the CPU usage starts to increase it sets the operating frequency to the maximum points. Our approach allocates the number of parallel cores and their DVFS points based on the application workload types to ensure energy efficiency for all application scenarios.

The IPS/Watt improvements are recorded when the optimal runtime configuration obtained with our adaptation approach is running. However, the adaptation process itself may cause overheads in two ways. Firstly, the frequency needs to be swept through a number of different values. During the sweep each frequency value is held 200 ms (see Algorithm 4.1) so that our performance counters can be used to collect the relevant data and different thread-to-core allocations tested. Whilst the application is not idle during this time, it is not always running at the optimal frequency and/or core allocation. Secondly, the application needs to be truly paused during the following operations:

- when performance counters are extracted - this cost is very low, around 210 ns for each round of extraction (30 ns per performance counter and seven performance counters);
- when the model is established - using floating point calculations, modeling should complete within 2 ms and using fixed-point,

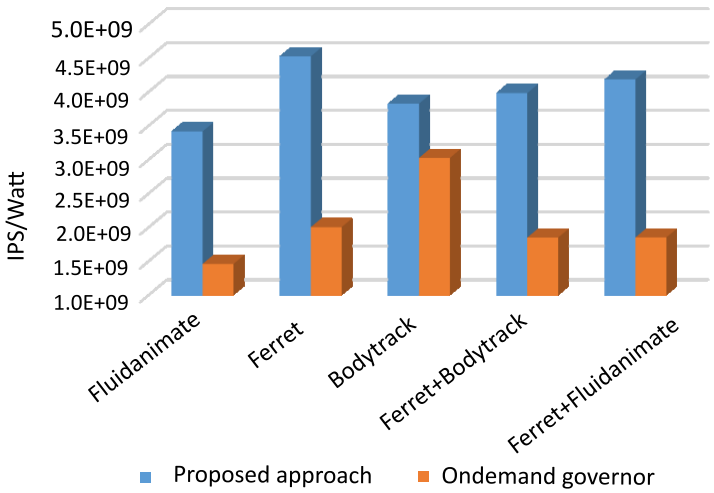


Figure 4.8: Comparative IPS/Watt between the proposed approach and ondemand governor [75] with all 8 cores allocated to the applications.

which we have not tried, should in principle reduce this cost further;

- when the operating frequency is tuned - each frequency change costs 300 ns for the ARM platform experimented here;
- when changing thread to core allocations - this is the highest application-pausing delay with experiments showing an average latency of 6.7 ms per change.

Considering all this, the true application pausing intervals add up to very little time during each round of runtime adaptation and the main issue is the total time needed to sweep through enough frequencies and core allocations during which the application is mostly not run optimally. This adaptation, however, is only carried out once per new application execution state, i.e. when an application starts or ends. In most real-world situations such events happen relatively sparsely. For instance, if these events happen every 100 seconds and we sweep through five frequencies, only during 1% of the time the system may not be optimal. The total true application pause time (in the tens of milliseconds at most) is negligible. The presented approach focuses on power-aware performance optimization, the absolute value for performance is not a major concern.

Although the main focus of this work is on combined power/performance metrics such as IPS/Watt, data on pure performance is also collected. This is presented in Figure 4.9. This data shows the effects on performance of the model-based control algorithm (see Algorithm 4.1 and Figure 4.7). Figure 4.9 presents the execution time of *ferret* application at 600 MHz and 1400 MHz for different thread-to-core allocations. As expected the following two observation can be made. Firstly, the

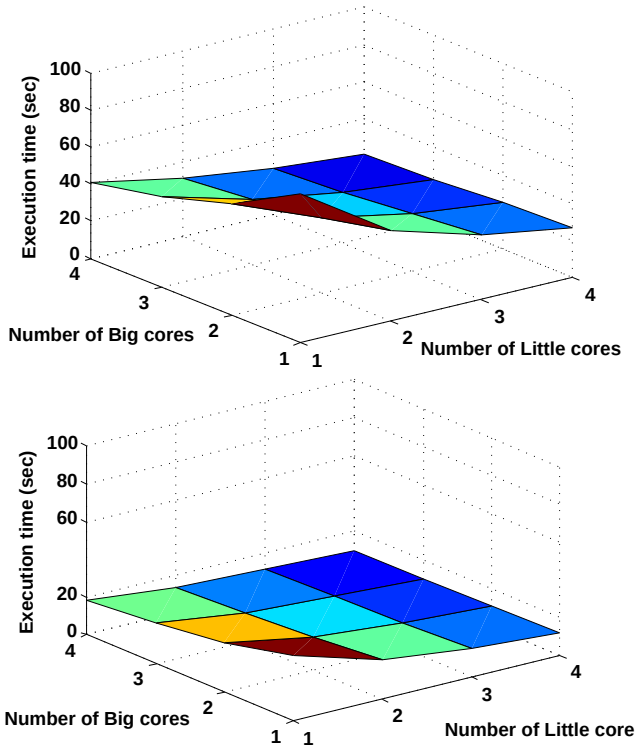


Figure 4.9: The execution time for different core allocation at 600 MHz and 1400 MHz respectively for *ferret* application.

execution time is dependant on the operating frequency. Secondly, the execution time decreases as more cores are allocated for the given application.

4.5 SUMMARY AND CONCLUSION

This chapter aims to propose and discuss power and performance for many core heterogeneous systems. We demonstrated a novel run-time approach, capable of power-aware performance adaptation under sequential and concurrent application scenarios in heterogeneous many-core systems. The approach is based on power and performance models that can be obtained during runtime by multivariate linear regression based on low-complexity hypotheses of power and performance for a given operating frequency. The approach is extensively evaluated using the PARSEC-3.0 benchmark suite running on the Odroid-XU3 heterogeneous platform. A selection of experimental results was presented to illustrate the kinds of tradeoffs in a variety of concurrent application scenarios, core allocations, and DVFS points, highlighting an improvement of power normalized performance which produced IPS/Watt improvements between 26% and 125% for a range of applications as can be seen in Figure 4.8. It is expected that modern embedded and high-performance system designers will benefit from the proposed approach in terms of a systematic power-aware performance optimization under variable workload and application scenarios.

5

MODEL-FREE RUNTIME MANAGEMENT OF CONCURRENT WORKLOADS

In order to achieve lower complexity than the model-based runtime optimization, a model-free approach to runtime adaptation based on workload classification is presented in this chapter. This classification is supported by analysis of data collected from a comprehensive study investigating the tradeoffs between inter-application concurrency with performance and power under different system configurations. Extensive experiments are carried out on Odroid XU3 heterogeneous and Intel Core i7 Sandybridge homogeneous platforms with synthetic and standard benchmark applications to develop the control policies and validate this approach.

The rest of this chapter is organized as follows. Section 5.1 shows the introduction and motivation of the model-free approach. Section 5.2 shows the experimental environment, the configuration of system used in the experiment and the applications. The system approach is described in Section 5.4, which include the workload classification and control decision making. Section 5.5 presents a case study of concurrent application with runtime management (RTM) stability, robustness and complexity. Moreover a comparative evaluation of the runtime management is presented in this section. Section 5.6 concludes the work in this chapter.

5.1 INTRODUCTION AND MOTIVATION

Contemporary computing systems, including embedded and high performance systems, are exhibiting increased complexities in two dimensions. In one dimension, the number and type of computing resources (cores) are growing in hardware platforms, and in the other, an increasing diversity of applications are being executed concurrently on these platforms [81] [79] [125]. Managing hardware resources to achieve energy efficiency, under different application scenarios (single or concurrent), is proving highly challenging due to runtime state-space expansion [36].

As energy consumption becomes a limiting factor to continued technology scaling and performance improvements [14], techniques for increasing energy efficiency have emerged. To provide control over power/performance tradeoffs, (DVFS) is integrated into contemporary devices, e.g. current Intel and ARM processors [71]. DVFS suitably scales voltage/frequency across a number of pre-determined operating points. These have different impacts on performance and power

consumption and hence their choices need to be made based on the application workload. Another technique for improving energy efficiency is the parallelization of workloads [125], including suitable (task mapping (TM)) to cores.

DVFS and TM may be synergistically controlled at the system software level for effective energy optimization. For instance, DVFS is controlled in Linux with power governors [75], such as *ondemand*, *performance*, *conservative*, *userspace* and *powersave*. These governors use pre-set voltage/frequency points to manage system power according to the knowledge and prediction of workload and user preference. Current Linux governors are, however, not able to optimize energy consumption efficiently, primarily because they are unable to couple DVFS and dynamic TM [75]. Further, these approaches, although serviceable, are not capable of taking advantage of the different degrees of parallelizability of individual applications that are typically seen in modern computing systems.

Mapping threads to cores (TM) is usually handled by a separate routine in the system software, for example the Linux scheduler [110]. The scheduler seeks to spread the thread workload of all applications across multiple available cores to achieve maximum utilization. This approach is functional but leaves rooms for improvement. For instance, there is no discrimination about the thread workload type when being scheduled [110], such as CPU-intensive or memory-intensive. Not taking the workload type into account results in indiscriminate sub-optimization in power and performance, leading to poor energy efficiency [4][86].

A number of approaches have been proposed over the years that consider energy optimization using offline (OL), runtime or a combination of both (see Table 5.1). A recurring scheme in these approaches is that the energy efficiency is primarily focused on single-application workloads without considering its variations among concurrent applications. However, the same application can exhibit different energy/performance trade-offs depending on whether it is running alone or concurrently with other different workloads. This is because:

- the workload context switches within the application between memory- and CPU-intensive routines, and
- architectural sharing between applications affect the energy/performance tradeoffs (see Section 5.5).

Table 5.1: Features of existing approachs and this work.

Approach	Platforms	Workload Classification	Validation	Apps	Controls	Size
[33] [66]	homogeneous	No	simulation	single	TM+DVFS	P
[128]	heterogeneous	No	simulation	single	RT, TM+DVFS	P
[74]	homogeneous	No	practical	single	RT, DVFS	L
[78]	heterogeneous	No	simulation	single	OL, TM+DVFS	P
[4]	heterogeneous	offline	practical	concurrent	RT, TM+DVFS	NP
[86]	heterogeneous	offline	practical	concurrent	RT, DVFS	NP
[117]	not CPUs	runtime	practical	concurrent	RT, DVFS	NP
This work	**heterogeneous**	**runtime**	**practical**	**concurrent**	**RT, TM+DVFS**	**L**

In this chapter, a runtime adaptation approach is developed to improve the energy efficiency of a heterogeneous many-core system with concurrent workloads. Core to this approach is an empirical and data-driven method, which classifies applications based on their memory and CPU requirements. The aim is to derive DVFS and TM policies, tailored to the classified workloads. Due to simplified runtime classification, the approach can significantly reduce overheads. Further, our model-free classification based RT enhances scalability for any concurrent application mix, platform, and metric having linear complexity (L) which is not affected by system heterogeneity, the number of concurrent applications, and using both DVFS and TM. In comparison, linear complexity was only achieved in existing work when dealing with single applications running on homogeneous systems with one of TM or DVFS, but not both (see Table 5.1 and Section 5.4 for details). Otherwise they display combinatorial polynomial (P) or non-polynomial (NP) complexities. This work makes the following specific contributions:

- using empirical observations and CPU performance counters, derive RT workload classification thresholds, expressed in terms of instructions per cycle (IPC);
- underpinned by the workload classification, propose a low-complexity and low-cost RT approach for synergistic controls of DVFS and TM;
- using synthetic and real-world benchmark applications with different concurrent combinations, investigate the approach's energy efficiency, measured by power-normalized performance in instructions per second (IPS) per Watt (IPS/Watt), i.e. instructions per Joule;
- implement the approach as a Linux power governor and validate through extensive experimentation with significant IPS/Watt improvements.

To the best of our knowledge, this is the first work that uses workload classification (WLC) during runtime to optimize both DVFS and TM for concurrent workloads on many-core heterogeneous platform (see Table5.1).

5.2 EXPERIMENTAL PLATFORM AND APPLICATIONS

Our experimental investigations use a many-core platform to illustrate the suitability of the proposed approach, when executing workloads on a number of heterogeneous cores. Further, we study scalability by executing a number of concurrent applications on this example platform.

The homegenous platform of choice is the Intel Core i7 Sandybridge CPU. It has (4) hard cores and (8) soft cores. Core i7 has a wide range of operating frequencies and voltages. To measure the CPU power a shunt resistor is inserted into the earth of the power coneection to the CPU [60, 111].

Likwid performance counters are used to record the performance events, such us instruction retired, memory access, unhalted cycles, and clock refrence [121].

The heterogenous platform of choice is the Odroid XU3 [2], which includes an SoC based on the ARM big.LITTLE architecture. Extensively described in Section 3.2, it has eight general processing ARM Cortex cores. Four of these are low-power A7 cores and the other four high-performance A15 cores. Each group of four cores of the same type constitutes a power domain, which is supplied with the same frequency and voltage, and the XU3 provides RT power monitoring per power domain, and per-domain operating points.

The A7 and A15 processor architectures also provide performance counters that record, per-core, instructions executed and clock active and idle cycles. This work uses the set of performance counters listed in Table 5.2.

Table 5.2: Performance counter events

Performance counter	Description
InstRet	Instructions executed
Cycles	Unhalted cycles on a core
Mem	Data memory access

In our investigation, we chose a number of different applications. A synthetic benchmark, called *mthreads* is developed, based on purely CPU-intensive stress enhanced with tunable memory access M, that is in linear relation to the real memory to computation ratio, to investigate the general CPU vs memory effects. In addition, a group of realistic application benchmarks from the PARSEC suite [3] included to span the range of CPU, memory, and mixed execution characteristics. Specifically, we chose the application *ferret* to represent CPU-intensive, *fluidanimate* to represent memory-intensive, and *bodytrack* to represent both CPU- and memory-intensive applications. It will be demonstrated later that *mthreads* is needed to represent pure CPU-only and memory-only tests because realistic applications, such as PARSEC benchmarks, have CPU- and memory-intensive contexts during their execution traces. This sets up one of the major motivations for classifying during runtime.

5.3 WORKLOAD CLASSIFICATION TAXONOMY

The taxonomy of workload classes chosen for this work reflects the desire of differentiating CPU-intensive workloads from memory-intensive ones and differentiating high-activity workloads from low-activity ones. The former concern follows previous findings that CPU-intensive workloads demand different core-allocation and DVFS decisions from memory-intensive ones [38], [86]. The latter is based on reasoning it is senseless to run a low-activity workload on a large number of cores at high clock frequencies. Workloads are classified into four classes:

- Class 0: low-activity workloads.
- Class 1: CPU-intensive workloads.
- Class 2: CPU- and memory-intensive workloads.
- Class 3: memory-intensive workloads.

Extensive explorative experiments are run in this work to investigate the validity of these general concepts. The experiments are based on a synthetic benchmark, called *mthreads*, which attempts to re-create in a controlled manner the effect of memory bottleneck on parallel threads. The program is based on repeated mix of CPU-intensive and memory-intensive operations, the ratio of each type is controlled by the parameter M. CPU operation is a simple integer calculation. Memory operation is implemented by randomly writing to a large (64MB) pre-allocated array. This is done to reduce the effect of caching. Parameter $M = 0$ gives CPU-intensive execution, $M = 1$ creates memory-intensive execution; the values in between provide a linear relation to the number of memory accesses per instruction. The execution is split into N identical parallel threads, each pinned to a specific core. Figure 5.1 presents the flowchart of the application.

Figure 5.2 shows the energy efficiency of *mthreads* running on 2-4 A7 cores (one of the A7 cores was reserved for the operating system in these experiments, hence the data does not cover the single core case) with M values ranging from 0 to 1. It can be seen that with memory-intensive tasks (larger M), it is better to use fewer cores, but with CPU-intensive tasks (smaller M), it is better to run more cores in parallel. This and other results sweeping through the frequency ranges and core combinations with *mthreads* confirm the validity of the classification taxonomy and establish a TM and DVFS strategy based on relative CPU and memory use rates. The full set of *mthreads* experimental data, supported by experiments with applications other than *mthreads*, is used to generate our runtime management (RTM) presented in subsequent sections.

The explorative experiments cover a much larger range of application and core combinations and frequency ranges than reported in

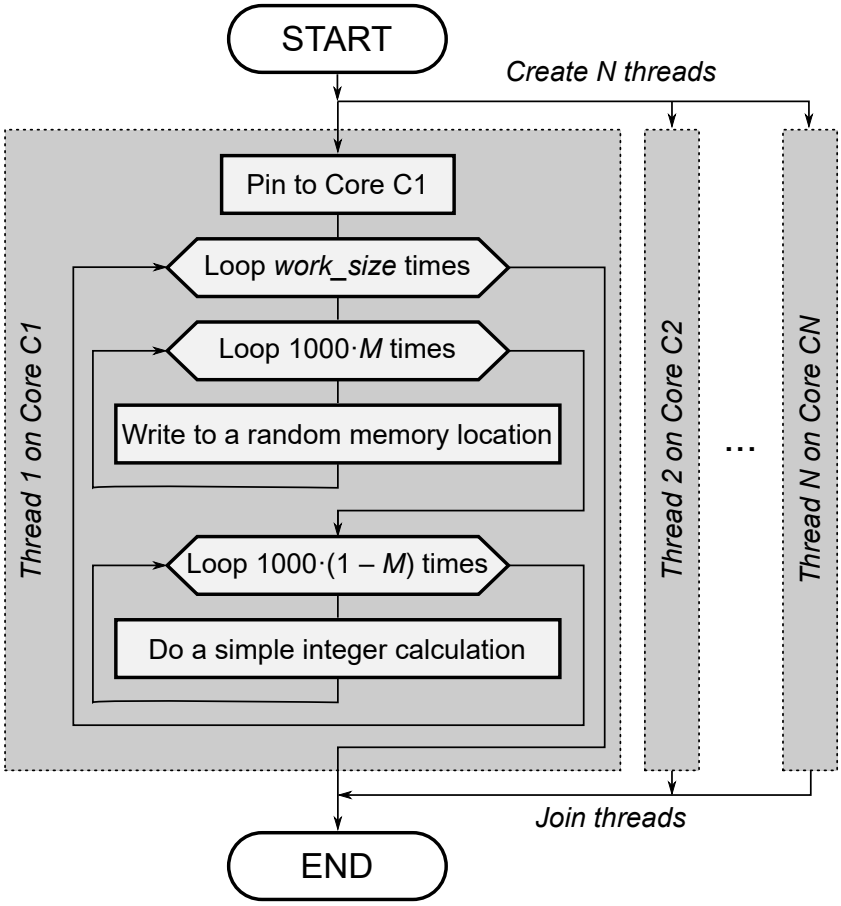


Figure 5.1: Flowchart of *mthreads* synthetic benchmark. M and N are controlled parameters.

these figures, and the results lead to the core allocation and frequency decision preferences presented in the next section, which form the foundation of the governor decision-making.

5.4 RUNTIME MANAGEMENT METHOD AND GOVERNOR DESIGN

This work proposes a runtime management approach leading to a governor based on workload classification. The general architecture of the RTM inside a system is given in Figure 5.3. In this section we explain the central RTM functions classification and control action based on monitors (e.g. performance counters) and actuators (e.g. core-allocation and DVFS).

Figure 5.3 presents the general architecture of RTM inside a system. In this section we explain the central RTM functions-classification and control action based on monitors (e.g. performance counters) and actuators (e.g. TM and DVFS).

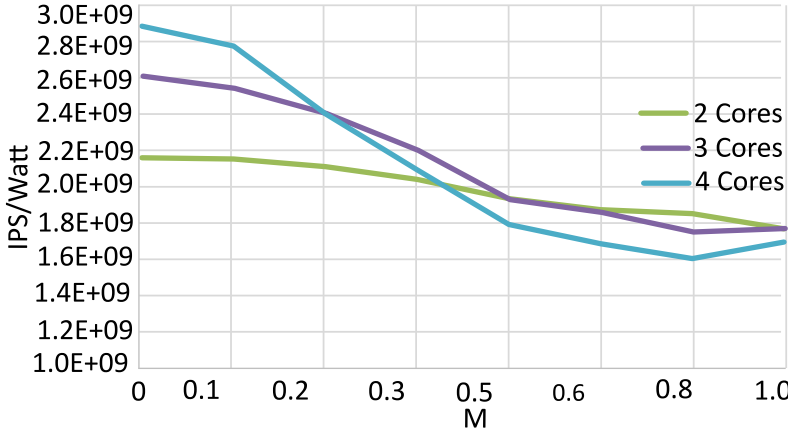


Figure 5.2: IPS/Watt for different memory use rates ($0 \leqslant M \leqslant 1$).

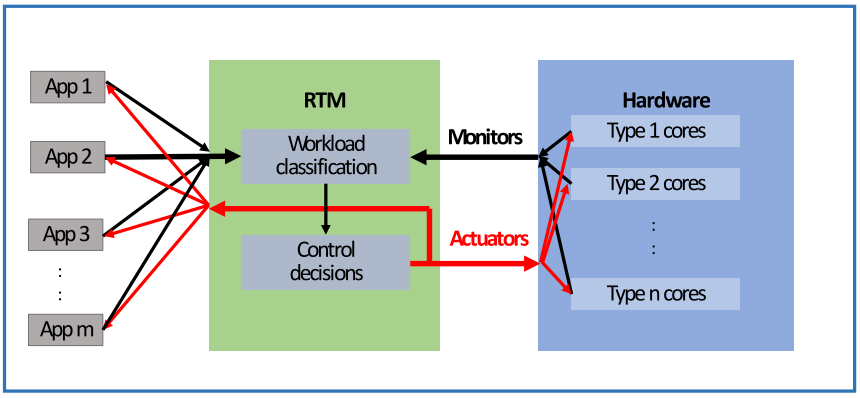


Figure 5.3: RTM architecture showing two-way interactions between concurrent applications and hardware cores.

The general approach does not specify the exact form of the taxonomy into which workloads are classified, the monitors and actuators the system need to have, or the design figure of merit. Our examples classify based on differentiating CPU and memory usages and the execution intensiveness, try to maximize IPS/Watt through core-allocation and DVFS, and get information from system performance counters.

5.4.1 *Workload classification*

Real applications do not have tunable memory use rates. As a result, information from performance counters (monitors) is used to derive the classes of all applications running on the system for each control decision cycle. This is based on calculating a number of metrics from performance counter values recorded at set time intervals, and then deriving the classes based on whether these metrics have crossed certain thresholds. The metrics and how they are calculated are given in Table 5.3 . These metrics are explained as follows.

Table 5.3: Metrics used to derive classification.

Metrics	Definitions
nipc	(InstRet/Cycles)(1/IPCmax)
iprc	InstRet/ClockRef
nnmipc	[(InstRet/Cycles)-(Mem/Cycles)](1/IPCmax)
cmr	(InstRet-Mem)/InstRet
uur	Cycles/ClockRef

Normalized instructions per clock (***nipc***): This metric measures how intensive the computation is for an application. IPCmax is the maximum IPC for the core type, obtainable from manufacturer literature or running highest IPC instructions such as nop in experiments, and Cycles is the unhalted cycles counted when execution is happening. This metric is therefore the IPC of an execution on a core during execution, normalized by the maximum IPC possible of that core type.

With normalized IPC, the classification threshold values can be generalized across different types of cores. Normalization allows nipc to be used independent of core types and architectures.

Instructions per reference clock (***iprc***): This metric contributes to determining how active the computation is for an application. ClockRef is the total number of clock cycles of the execution, calculated by ClockRef = Freq/Time with frequency and execution time from the operating system.

Normalized non-memory IPC (***nnmipc***): This metric measures how CPU-intensive the computation of an application is. This is pure CPU IPC normalized by the maximum IPC for the core type.

CPU to memory ratio (*cmr*): This metric measures how relatively CPU- and memory-intensive an application is

Unhalted clock to reference clock ratio (*urr*): This metric contributes to determining how active an application is.

The general relationship between these metrics and the application classes are clear, e.g. the higher cmr and nnmipc are, the more CPU-intensive a computation is.

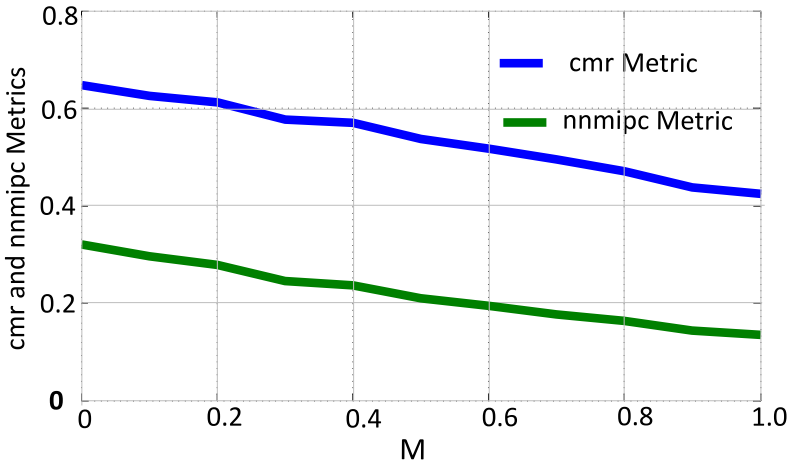


Figure 5.4: *mthreads* and their Performance Counter Metrics on Hetrogenous Many-core Systems.

A workload can be classified by comparing the values of metrics to thresholds. Decision-making may not require all metrics. The choice of metrics and thresholds and be made by analysing characterization experiment results.

From analysing the relationship between M and the list of metrics from *mthreads* experiments, we find that nnmpic shows the best spread of values with regard to corresponding to different values of M (See Figure 5.4). This leads to more straightforward arrangements of threshold values between different application classes.

Referring to the declared classes in PARSEC applications (*ferret* is claimed to be CPU-intensive, for instance [3] , this hypothesis is confirmed.

As a result, we choose nnmipc to differentiate CPU and memory usage rates and urr for differentiating low and high activity. Then thresholds (Table 5.6) are determined based on our *mthreads* characterization database. The other metrics may work better on other platforms and are included here as examples of potential candidates depending on how a *mthreads*-like characterization program behaves on a platform with regard to the relationships between M values and the metrics.

We list the relationships between these metrics and PARSEC benchmarks on heterogenous and homegenous (complete programs running in isolation) in Table 5.4 and Table 5.5 respectively. Figure 5.4 shows

the relation between *mthreads* and their performance counter metrics. It can be seen that nnmipc provides a better differentiation between the different types of applications, which is supported by observations of *mthreads* studies.

Table 5.4: PARSEC applications and their performance counter metrics on hetrogenous many-core systems

Applications	Instruction retired	Memory Access	Unhalted Cycles	ClockRef	*nnmipc*	*nipc*	*iprc*	*cmr*	*uur*
Bodytrack	352157530	93195406	4.22E+08	700000000	0.306	0.417	0.503	0.754	0.603
Ferret	799110436	249929313	7.13E+08	701400000	0.384	0.560	0.978	0.739	1.017
Fluidanimate	498349493	174527925	7.86E+08	721400000	0.206	0.317	0.690	0.723	1.088
streamcluster	399031486	167251625	6.97E+08	700000000	0.166	0.286	0.570	0.465	0.995

Table 5.5: PARSEC applications and their performance counter metrics on Intel Core i7 Sandybridge CPU

Applications	INSTR RETIRED	MEM L1 HIT	MEM L1 MISS	Memory Access	CPU_CLK _UNHALTED	*iprc*	*nnmipc*	*cmr*
Bodytrack	487860378	99482801	3781071	103263872	167661158	0.727	0.573	0.788
Caneal	117671816	27831604	1570087	29401691	180238196	0.714	0.586	0.750
Fluidanimate	1890311496	514182187	1871798	516053985	679974229	0.6949	0.505	0.727
Freqmine	1985142236	541986455	1830559	543817013	572360214	0.867	0.629	0.726
Streamcluster	1023862294	334608291	2805409	337413700	691607919	0.370	0.248	0.670

A workload can be classified by comparing the values of relevant metrics to thresholds. Thresholds may be determined through extensive characterization experiments, which is the case in this work because we have accumulated enough experimental data for this. The threshold values used and how they determine application classes are listed in Table 5.6.

Table 5.6: Classification details.

Metric ranges	Class
urr of all cores [0, 0.11]	0: low-activity
nnmipc per-core [0.3, 1)	1: CPU-intensive
nnmipc per-core [0.25, 0.3)	2: CPU+memory
nnmipc per-core [0, 0.25)	3: memory-intensive

5.4.2 *Control decision making*

This section presents an RTM control algorithm that uses application classes to derive its decisions. The behaviour is specified in the form of two tables: a threshold table (Table 5.6), used for determining application classes, and a decision table (Table 5.3), providing a preferred action model for each application class.

Algorithm 5.1 Inside the RTM cycle.

1: Collect monitor data
2: **for** each application **do**
3: Compute classification metrics {Section 5.4.1}
4: Use metric and threshold table to determine application class {Table 5.3}
5: Use decision table to find core allocation and frequency preferences {Table 5.6}
6: Distribute the resources between the applications according to the preferences
7: Wait for $T_{control}$ {Section 5.4.2}
8: **end for**
9: **return**

The introduction of new concurrent applications or any other change in the system may cause an application to change its behaviour during its execution. It is therefore important to classify and re-classify regularly. The RTM works in a dedicated thread, which performs clas-

sification and decision making action every given timeframe. The list of actions performed every RTM cycle is shown in Algorithm 5.1.

Table 5.7: RTM control decisions.

Class	frequency	A7	A15
0	min	single	none
1	max	none	max
2	min	max	none
3	max	single	none
unclassified	min	single	none

In Algorithm 5.1 $T_{control}$ is the time between two RTM control cycles. The RTM determines the TM and DVFS of power domains once each control cycle, and these decisions keep constant before the next control cycle. The data from the system monitors (performance counters and power meters) is collected asynchronously. Every core has a dedicated monitor thread, which spends most of its time in a sleep state and wakes every $T_{control}$ to read the performance counter registers. The readings are saved in the RTM memory. This means that the RTM always has the latest data, which is at most $T_{control}$ old. This is mainly done because ARM performance counter registers can be accessed only from code on the same CPU core. In this case, periodic monitoring has been empirically shown to be more efficient. In our experiments we have chosen $T_{control} = 500ms$, which has shown a good balance between RT overhead and energy minimization. The time the RTM takes (i.e. RT overhead) is negligible compared to 500ms for the size of our system. This interval can be reduced with slightly higher overheads, or increased with less energy efficiency tradeoffs.

The RTM uses monitor data to calculate the classification metrics discussed in Section 5.4. These metrics form a profile for each application, which is compared against the thresholds (Table 5.6). Each row of the table represents a class of applications and contains a pre-defined value range for each classification metric. An application is considered to belong to a class, if its profile satisfies every range in a row. If an application does not satisfy any class, it is marked as "unclassified" and gets a special action from the decision table. An application is also unclassified when it first joins the execution. In that case it goes to an A15 core for classification.

The decision table (Table 5.7) contains the following preferences for each application class, related to system actuators (DVFS and core allocation decisions): number of A7 cores, number of A15 cores, and clock frequencies. Number of cores can take one of the following values: none, single, or maximum. Frequency preference can be minimum or maximum. The CPU-intensive application class (Class 1) runs on the maximum number of available A15 cores at the maximum frequency

```
const int class_decision_freq[NUM_CLASSES+1] = {
        DECISION_MIN, DECISION_MAX, DECISION_MIN, DECISION_MIN, DECISION_MIN
};
const int class_decision_a7cores[NUM_CLASSES+1] = {
        DECISION_MIN, DECISION_NONE, DECISION_MAX, DECISION_MIN, DECISION_MIN
};
const int class_decision_a15cores[NUM_CLASSES+1] = {
        DECISION_NONE, DECISION_MAX, DECISION_MAX, DECISION_NONE, DECISION_NONE
};
```

Figure 5.5: Code implementing Table 4

as this has shown to give the best energy efficiency (in terms of power normalized performance) in our previous observations [75]. Figure 5.5 is the portion of code that implements the decision table . This can be found in the RTM model free program in Appendix **??**.

Table 5.6 and Table 5.7 are constructed offline in this work based on large amounts of experimental data, with those involving PARSEC playing only a supporting role. For instance, although *ferret* is regarded as CPU-intensive, it is so only on average and has non CPU-intensive phases (see Section 5.5). Therefore Table 5.7 is obtained mainly from analysing experimental results from our synthetic benchmark *mthreads* (which has no phases), with PARSEC only used for checking if there are gross disagreements (none was found). Because of the empirical nature of the process, true optimality is not claimed.

In this work, we assume that there are always more cores than running applications, without losing generality. The RTM attempts to satisfy the preferences of all running applications. In the case of conflicts between frequency preferences, the priority is given to the maximum frequency. When multiple applications request cores of the same type, the RTM distributes all available cores of that type. When these conflicting applications are of different classes, each application is guaranteed at least a single core. Core allocation (TM) is done through the following algorithm.

Algotirhm 5.2 shows the procedure ApplyDecision for mapping the RTM decisions to the core affinity masks. RTM provides a decision for each app and for each core type $d_{j,i} \in \{\text{NONE}, \text{MIN}, \text{MAX}\}$, where $j \in \{A7, A15\}$ is the core type and $1 \leqslant i \leqslant m$ is the app index, given the total number of apps m. The decisions are arranged in arrays $D_{A7} = (d_{A7,1}, \ldots, d_{A7,m})$ and $D_{A15} = (d_{A15,1}, \ldots, d_{A15,m})$. Additional constants used by the algorithm are: n_{A7}, n_{A15} are the total number of little and big cores respectively, and the IDs of cores by type are listed in the pre-defined $C_{A7} = \left(c_{A7,1}, \ldots, c_{A7,n_{A7}}\right), C_{A15} = \left(c_{A15,1}, \ldots, c_{A15,n_{A15}}\right)$. The complexity of the algorithm is linear to m. The result of the algorithm is the set of core IDs C_i, which can be used to call the `sched_setaffinity` function for the respective app i.

Algorithm 5.2 Mapping the RTM decisions to the core affinites

procedure ApplyDecision(D_{A7}, D_{A15})
$(r_{A7,1}, \ldots, r_{A7,m}) \leftarrow$ ReqCores (D_{A7}, n_{A7}) {Get per-app number of little cores}
$(r_{A15,1} \ldots, r_{A15,m}) \leftarrow$ ReqCores (D_{A15}, n_{A15}) {Get per-app number of big cores}
for $1 \leqslant i \leqslant m$ **do**
  $C_{i,A7} \leftarrow$ (next $r_{A7,i}$ elements from C_{A7})
  $C_{i,A15} \leftarrow$ (next $r_{A15,i}$ elements from C_{A15})
  $C_i \leftarrow C_{i,A7} \cup C_{i,A15}$ {Use C_i to set core affinity mask for the app i.}
end for
end procedure

function REQCORES($d_1, \ldots, d_m$), n
$k_{MIN} \leftarrow$ count $(d_i = \text{MIN})$ for $1 \leqslant i \leqslant m$
$k_{MAX} \leftarrow$ count $(d_i = \text{MAX})$ for $1 \leqslant i \leqslant m$
if $k_{MAX} > 0$ **then**
  $v \leftarrow \lfloor (n - k_{MIN}) / k_{MAX} \rfloor$ {v is the MAX number of cores}
  $w \leftarrow (n - k_{MIN}) \mod k_{MAX}$ {w is the remainder}
end if
for $1 \leqslant i \leqslant m$ **do**
  if $d_i = \text{MAX}$ **then**
    if $w > 0$ **then**
      $r_i \leftarrow v + 1$
      $w \leftarrow w - 1$ {Distribute the remainder}
    else
      $r_i \leftarrow v$
    end if
  else if $d_i = \text{MIN}$ **then**
    $r_i \leftarrow 1$
  else
    $r_i \leftarrow 0$
  end if
end for
return $(r_1, \ldots, r_m)$
end function

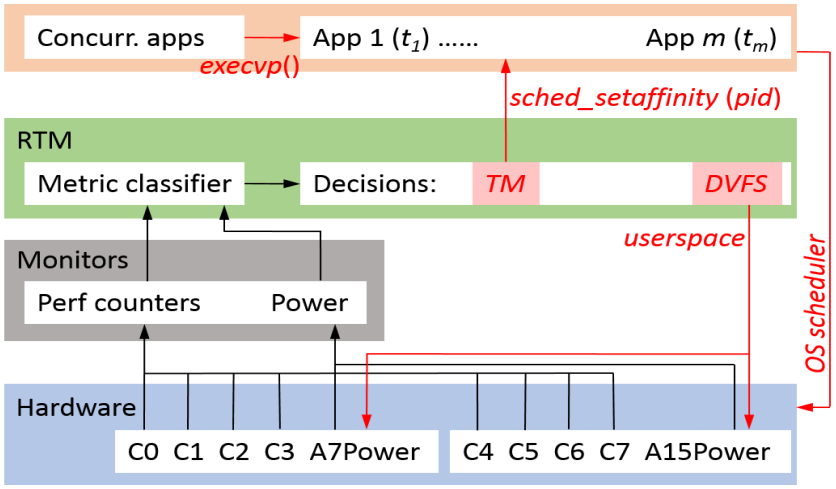


Figure 5.6: Governor implementation based on RTM.

5.4.3 *RTM govenor design*

The governor implementation is described in Figure 5.6, which refines Figure 5.3. At time *ti* application *i* is added to the execution via the system function ***execvp***(). The RTM makes TM and DVFS decisions based on metric classification results, which depends on hardware performance counters and power monitors to directly and indirectly collect all the information needed, making it possible for the RTM to avoid instrumenting applications. The TM actuation is done indirectly via system functions, for instance, core pinning is done using ***sched_affinity***(***pid***), where ***pid*** is the process ID of an application. DVFS is actuated through the ***userspace*** governor as part of ***cpufreq*** utilities.

5.5 EXPERIMENTAL RESULTS

Extensive experiments have been carried out with a large number of application scenarios running on the XU3 platform. These experiments include running single applications on their own and a number of concurrent applications. In the concurrent scenarios, multiple copies of the same application and single copies of different applications of the same class and different classes have all been tested.

5.5.1 *A Case Study of Concurrent Applications*

An example execution trace with three applications is shown in Figure 5.7. Parts at the beginning and end of the run contain single and dual application scenarios. The horizontal axis is time, and the vertical axis denotes TM and DVFS decisions. Cores Co-C3 are A7 cores and C4-C7 are A15 cores. The figure shows application classes and the

core(s) on which they run at any time. This is described by numbers, for instance, 2(3) on core C1 means that App 2 is classified as of Class 3 and runs on C1 for a particular time window. "1/u" means that App 1 is being unclassified. In this example trace, App 1 is *ferret*, App 2 is *fluidanimate*, and App 3 is square root calculation. As can be seen in this concurrent execution scenario, all three workloads, including the conventional Linux CPU-stress application, square root calculation, exhibit multi-class phase behaviour.

Table 5.8: The Power, Frequency, Number of Cores, Classification results for *ferret* Application.

Time	pid	class	ncores	Number A7_cores	Number A15_cores	Frequency(KHz) A7	Frequency(KHz) A15	Power(Watt) A7	Power(Watt) A15
0	3386	3	0	0	0	200000	200000	0.014	0.097
540	3386	3	1	0	1	200000	200000	0.014	0.096
1094	3386	1	1	0	1	200000	200000	0.018	0.189
1718	3386	1	1	1	0	200000	200000	0.032	0.128
2274	3386	1	1	1	0	200000	200000	0.032	0.118
2888	3386	1	1	1	0	200000	200000	0.027	0.080
3488	3386	1	3	0	3	200000	1900000	0.035	2.808
4014	3386	1	3	0	3	200000	1900000	0.026	4.489
4587	3386	1	3	0	3	200000	1900000	0.033	4.351
5145	3386	1	3	0	3	200000	1900000	0.036	4.362
5720	3386	1	3	0	3	200000	1900000	0.036	4.478
6330	3386	1	3	0	3	200000	1900000	0.027	4.502
6921	3386	1	3	0	3	200000	1900000	0.051	4.761
7484	3386	1	3	0	3	200000	1900000	0.038	4.450
8104	3386	1	3	0	3	200000	1900000	0.050	4.554
8669	3386	0	0	0	0	200000	1900000	0.039	4.400

The lower part of the figure shows the corresponding power and IPS traces. Both parameters are dominated by the A15 cores.

As can be seen in Figure 5.7, initial classifications are carried out on C4, but according to Algorithm 5.2, when C4-C6 are in application execution, C7 is reserved for this purpose, which is not needed in this trace. The reservation of dedicated cores for initial classification fits well for architectures where the number of cores is so large that we can assume that the number of applications is always smaller than the number of cores. This is not an overly restrictive assumption for modern (e.g. the Odroid XU3) and future systems with continuously increasing numbers of cores.

5.5.2 *Per-interval Re-classification*

The method does not employ per-application classfication. In other word, an application is not classified once and keep its class. Instead, a scheme of RT per-interval classification is adopted.

RT re-classification happens for all running applications at every $T_{control}$ = 500ms control cycle on their running core(s), according to Algorithms 5.1 and 5.2. Figure 5.7 shows the motivation for this re-classification. The same application can have memory usage phases and belong to different classes at different times. This means that offline classification methods, which give each application an invariable class, is unsuitable for efficient energy minimization.

Figure 5.8 shows core allocation during workload classifications for single memory intensive application , two concurrent memory applications, and three concurrent memory applications. On the other hand, Figure 5.9 shows core allocation of CPU intensive application, and two CPU concurrent applications. Table 5.8 gives power, frequency, classfication with number of cores for the CPU application.

Figure 5.10 shows example traces of the PARSEC apps *ferret* and *fluidanimate* being classified whilst running as single applications. It can be seen that the same application can have different CPU/memory behaviours and get classified into different classes. This is not surprising as the same application can have CPU-intensive phases when it does not access memory and memory-intensive phases where there is a lot of memory access. In addition, it is also possible for an application to behave as belonging to different classes when mapped to different numbers of cores. The classification can also be influenced by whether an application is running alone or running in parallel with other applications, if we compare Figure 5.7 and Figure 5.10. These are all strong motivations for RT re-classification. The result of classification affects an application's IPS (see Figure 5.7) and power (see Figure 5.10).

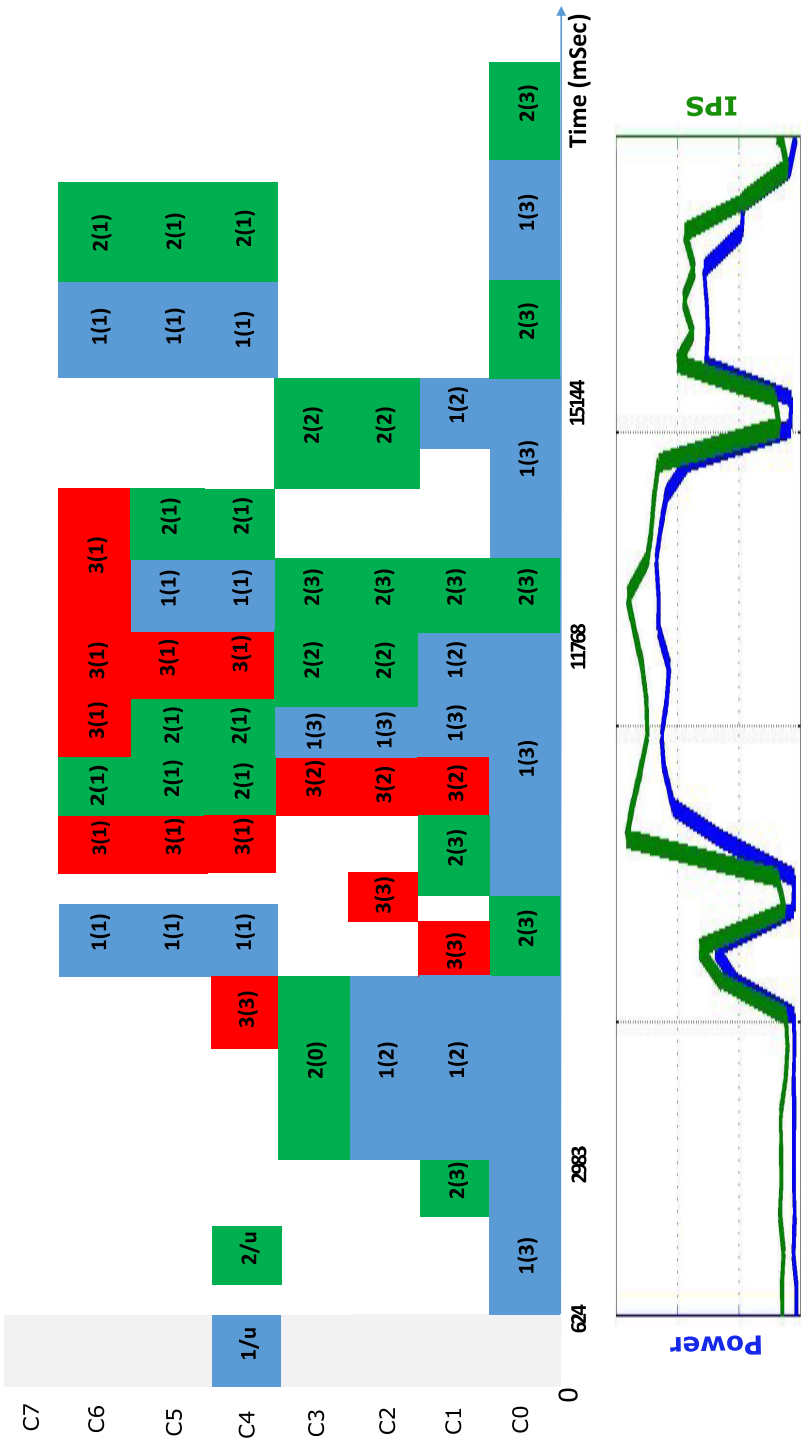


Figure 5.7: Execution trace with TM and DVFS decisions.

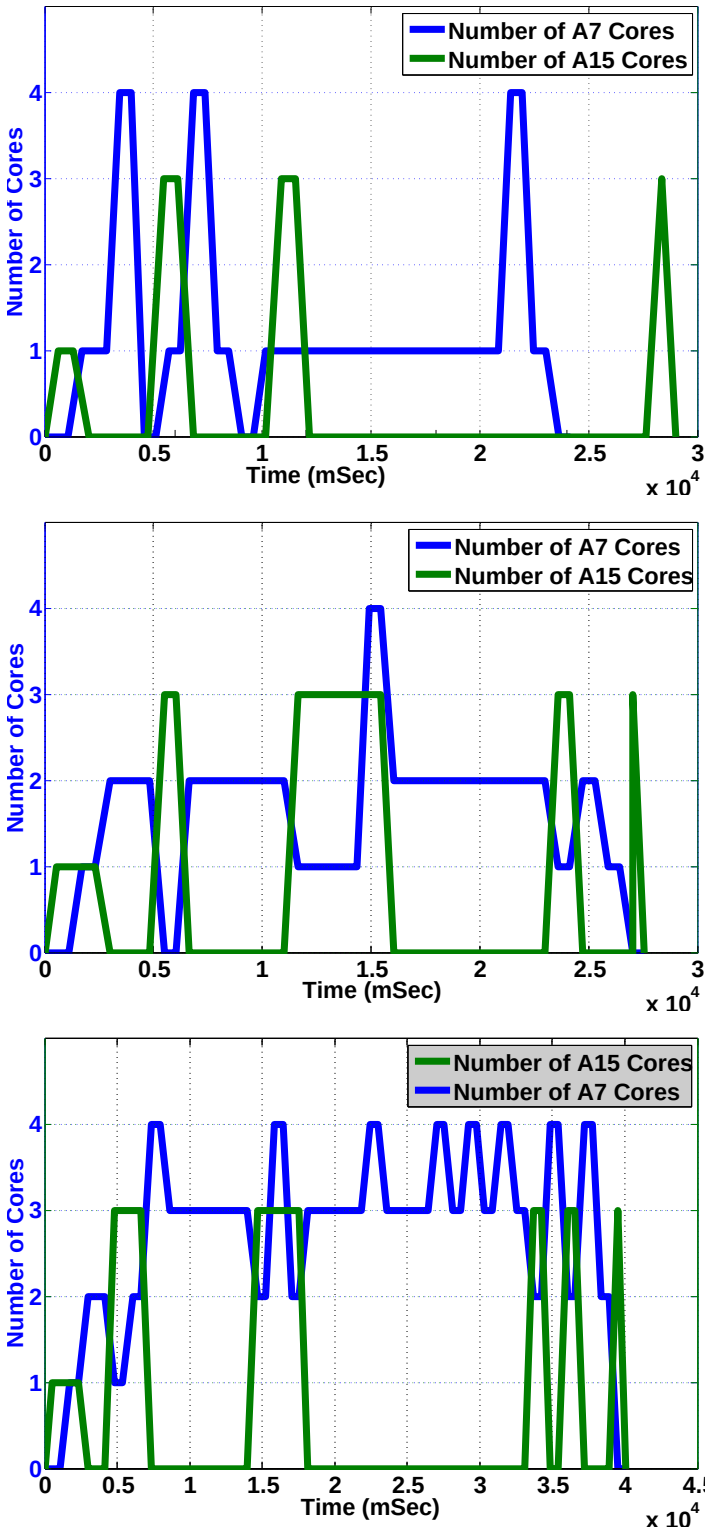


Figure 5.8: Core allocation for Fluidanimate application, Two Memory concurrent applications , Three Memory concurrent applications.

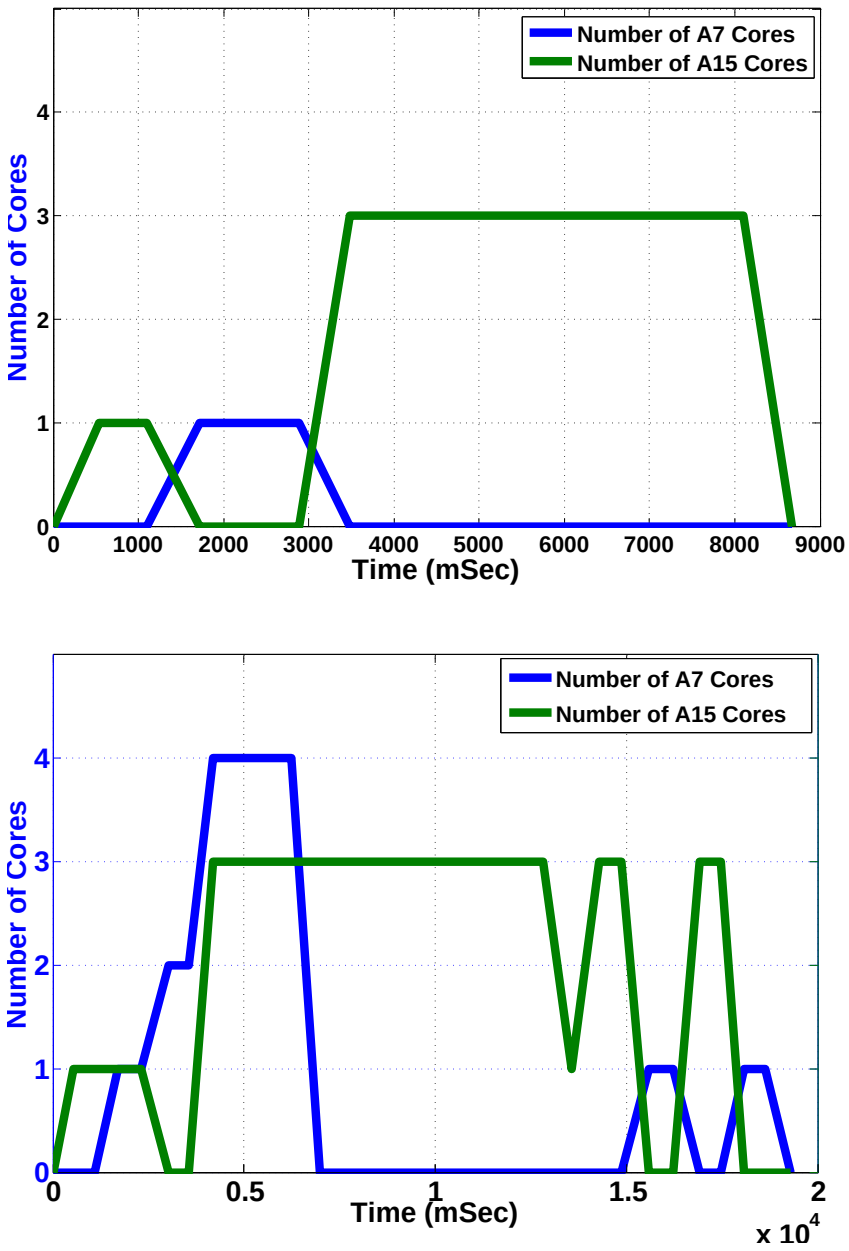


Figure 5.9: Core allocation for Ferret application, Two CPU concurrent applications.

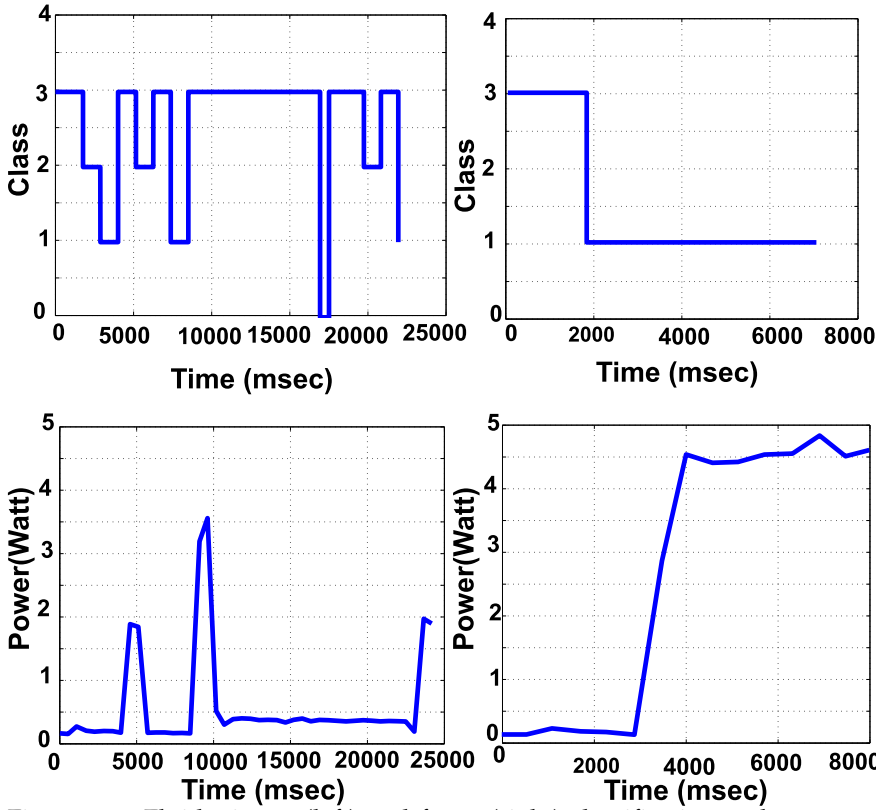


Figure 5.10: Fluidanimate (left) and ferret (right) classification and power traces .

5.5.3 *RTM stability, tobustness and complexity*

Algorithm 5.1 can oscillate between two different sets of classification and control decisions in alternating cycles. This may indicate a loss of stability. The reasons for such oscillations have been isolated into the following cases:

- The control cycle length coincides with an application's CPU and memory phase changes.
- An application's behaviour takes it close to particular threshold values, and different instances of evaluation put it on different sides of the thresholds.
- An application is not very parallelizable. When it is classified on a single core, it behaves as CPU-intensive, but when it is classified on multiple cores, it behaves as low-activity. This causes it to oscillate between Class 0 and Class 1 in alternating cycles.

We address these issues as follows. Case 1 rarely happens and when it happens it disappears quickly, because of the very low probability of an application's phase cycles holding constant and coinciding with the control cycle length. This can be addressed, in the rare case when it is necessary, by tuning the control cycle length slightly if oscillations persist.

Case 2 also happens rarely. In general, increasing the number of classes and reducing the distances between control decisions of adjacent classes reduce the RTM's sensitivity to threshold accuracy, hence Case 2 robustness does not have to be a problem, and thresholds (Table 5.7) and decisions (Table 5.9) can be tuned both offline and during runtime.

Case 3 is by far the most common. It is dealt with through adaptation. We put in an extra class, "low-parallelizability", and give it a single big core. This class can only be found after two control cycles, different from the other classes, but this effectively eliminates Case 3 oscillations.

5.5.4 *Comparative evaluation of the RTM*

Complexity: Our RTM has a complexity of $O(N_{app} \cdot N_{class} + N_{core})$, where N_{app} is the number of applications running, N_{class} is the number of classes in the taxonomy, and N_{core} is the number of cores. N_{class} is usually a constant of small value, which can be used to trade robustness and quality with cost. The RTM's computation complexity is therefore linear to the number of applications running and the number of cores. In addition, the basic algorithm itself is a low-cost lookup-table approach with the table sizes linear to N_{class}.

Schemes found in existing work, with e.g. model-based which is presented in Chapter 4 [4], machine-learning [100], linear programming [66], or regression techniques [128] [4], have a decision state space size of $(N_{A7DVFS} \cdot N_{A15DVFS}) \cdot (N_{A7} \cdot N_{A15})^{N_{app}}$, where N_{A7} and N_{A15} are the numbers of A7 and A15 cores and N_{A7DVFS} and $N_{A15DVFS}$ are the numbers of DVFS points of the A7 and A15 power domains, for this type of platform. This NP complexity is sensitive to system heterogeneity, unlike our approach.

Overheads: We compared the time overheads (OH) of our method with the linear-regression (LR) method found in e.g. [128] and [4]. For each 500ms control cycle, our RTM, running at 200MHz, requires 10ms to complete for the trace in Figure 5.7. Over 90% of this time is spent on monitor information gathering. In comparison, LR requires 100ms to complete the same actions. It needs a much larger set of monitors. The computation, also much more complex, evenly divides its time in model building and decision making. In addition, a modelling control such as LR requires multiple control intervals to settle and the number of control intervals needed is combinatorial with N_{A7}, N_{A15}, N_{A7DVFS} and $N_{A15DVFS}$.

Scalability: Our RTM is scalable to any platform as it is

- agnostic to the number and type of application running concurrently, and
- independent of the number or type of cores in the platform, and their power domains.

This is because the complexity of the RTM only grows linearly with increased number of concurrent applications and cores.

Performance: Direct comparison is possible only with Chapter 4 [4], which studies the same set of benchmarks running on the same platform. As shown in Table 5.9, which does not take the OH into account for [4], this RTM compares favourably in terms of overall advantages over the Linux *ondemand* governor. These selected experiments cover single applications and various combinations of applications of different classes running concurrently.

Data collected from our large number of validation runs shows the RTM out-performing the Linux *ondemand* governor by considerable margins on IPS/Watt, as shown in Table 5.9. The method can be generalized to other optimization targets, such as performance and energy-delay product. It is also possible to switch targets at runtime. Table /reftab77 shows the actual performance of the system in IPS has been affected by the proposed algorithms.

Table 5.9: Percentage IPS/WATT improvements of the RTM ovber the LINUX *ONDEMAND* Governor.

Application scenarios	WLC (w/OH)	LR [4] (no OH)
fluidanimate	127.0%	127.0%
ferret + *fluidanimate*	68.6%	61.7%
ferret + *fluidanimate* + *bodytrack*	46.6%	29.3%
fluidanimate ×2	24.5%	19.8%
fluidanimate ×3	44.4%	36.4%
ferret ×2	31.0%	26.5%

Table 5.10: Comparison of performance in terms of IPS of the proposed RTM with the LINUX *ONDEMAND* Governor.

Applications	Ondemand Governor	Proposed RTM
ferret	8117913	5284962
fluidanimate	7120060	1716090
ferret + *fluidanimate*	7249059	2221383
ferret + *fluidanimate* + *bodytrack*	8313643	2181637
fluidanimate ×2	7705368	1964424
fluidanimate ×3	7598735	1834730
ferret ×2	7952582	3728804

5.6 SUMMARY AND CONCLUSION

A runtime management approach is proposed for multiple concurrent applications of diverse types running on heterogeneous multi-core platforms.

Based on workload classification, the approach is demonstrated by a governor aimed at improving system energy efficiency (IPS/Watt). This governor classifies workloads according to their CPU and memory signatures and makes decisions on core allocation and DVFS. Workload classification leads to very low RTM complexity (linear with the number of applications and cores) and cost (lookup tables of limited size), leading to high scalability. Experiments show the governor producing significant improvements. Detection of low-parallelizability improves the stability of the governor.

The approach is general in the sense of being agnostic to metrics, platforms, and workloads. It can be extended to the optimization of other performance metrics and different taxonomies of workload classification so long as the metrics in question are related to the classes of the taxonomies.

6 REDUCED-COMPLEXITY RUNTIME MANAGEMENT OF CONCURRENT WORKLOADS

In order to determine the optimal system configuration which includes the number and type of cores with their frequencies trailered to workload combinations, workload classification combined are used with model-based (multivariate linear regression) to maximize the energy-efficiency in terms of IPS/Watt during runtime. This runtime optimization approach is based on using workload classification technique to reduce the complexity and overhead of power and performance models that can be obtained during runtime by multivariate linear regression. This approach is extensively evaluated using the PARSEC-3.0 benchmark suite running on the Odroid-XU3 heterogeneous platform. In a way this chapter attempts to combine the techniques presented in the two preceding chapters for further energy efficiency improvements.

The rest of this chapter is organized as follows. The proposed methodology is described in Section 6.3. The system approach is described in Section 6.4. The runtime concurrent workload classification is described in Section 6.5. DVFS and core allocation controller is described in Section 6.6. The experimental results are presented in section 6.7. Section 6.8 concludes this chapter.

6.1 INTRODUCTION

The MLR runtime modelling approach presented in Chapter 4 achieves provable approach for energy efficiency for concurrent applications running on heterogeneous many-core systems. However it extracts a computation and time overhead and during the decision-making process optimality is not guaranteed. It also pertains to each application and therefore is blind to application phase changes. Chapter 5 attempts to address this with a low-cost per-control interval approach which improves overheads both in time and computation cost, and therefore the energy efficiency of the runtime itself. It also avoids per-application decision making thereby exposing application phase behaviour and reacts appropriately to it. However, it does not achieve any kind of provable optimality other than intuition obtained from offline characterization. In this chapter, the method used in Chapter 5 is used to create a much smaller space in which the MLR method presented in Chapter 4 will be used to further optimality.

The motivation is that this should reduce the time, computation and power cost for the optimization step, and even during unoptimized

time periods, e.g. during learning, the WLC approach on its own would still be effective. Given that this will have a higher overhead than WLC alone, the obvious question we need to answer is whether this approach will return overall better results, including overheads, than Chapter 5.

6.2 STATE SPACE ANALYSIS

The state space of the many-core heterogeneous system in which optimization methods must search includes all possible core allocations and all possible DVFS combinations. Table 6.1 shows the number of possible core allocations for N_{apps} applications running on a system with the number of A7 and A15 cores set as $N_{A7} = 3$ and $N_{A15} = 4$ respectively. The "brute force" value represents $(N_{A7}+1)^N apps (N_{A15}+1)^N apps$ combinations, not all of which are actually allowed considering the following rules:

- each application must have at least one thread and
- no more than one thread per core is allowed

However, there is no simple algorithm to iterate through only valid core allocations. The number of possible core allocations is then multiplied by the number of DVFS combinations, which is calculated as MA7 MA15, where MA7 is the number of DVFS points in the A7 domain, and MA15 is the number of DVFS points in the A15 domain. It is clear that some method of reducing this state space would help reduce the overhead of the optimization step.

Table 6.1: Nnmber of possible core allocations

N_{apps}	**brute force**	**valid**
1	20	19
2	400	111
3	8000	309
4	$1.6*10^5$	471
5	$3.2*10^6$	405
6	$6.4*10^7$	185
7	$1.28*10^9$	35

6.3 PROPOSED METHODOLOGY

Our method studies concurrent application workloads being executed on various hardware platforms with parallel processing facilities, and

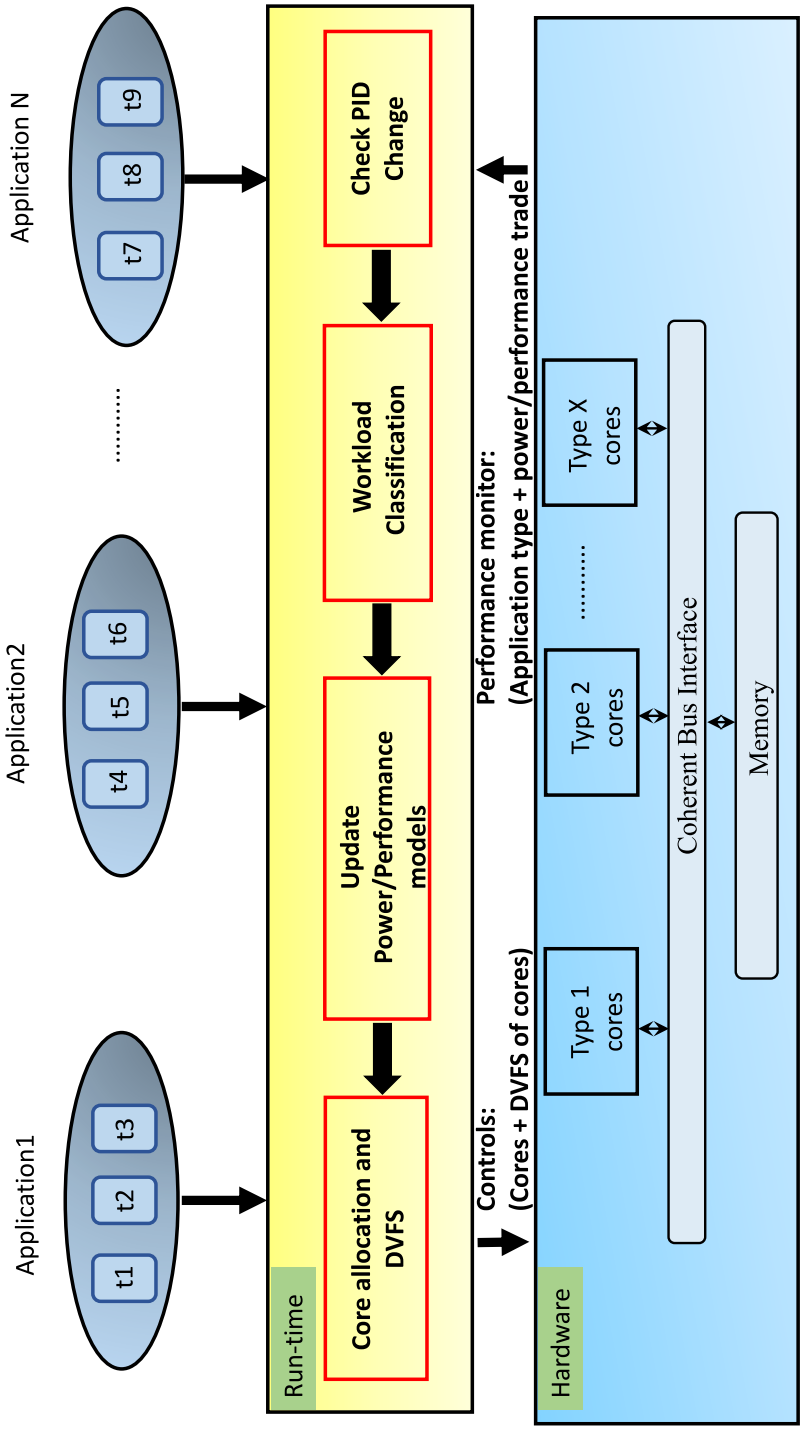


Figure 6.1: Block diagram for proposed runtime concurrent workloads power controller.

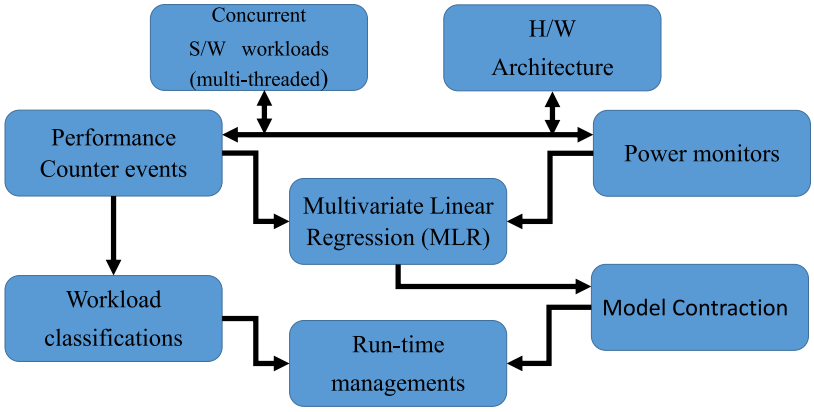


Figure 6.2: Simplified overview of proposed methodology.

derive optimal runtime management decisions from the results of this analysis.

Figure 6.1 gives the architecture of the proposed approach showing the interactions between the workload (multi-threaded), runtime managements, and hardware architecture and highlights our contributions in each section. Arrows indicate communications between layers. The applications are parallelizable workloads. The runtime includes both WLC and MLR modelling parts. The WLC restricts the space in which MLR operates. The workloads classification reduces the effort of model characterisation for combinations of concurrently running applications. The hardware architecture in this study is the Odroid XU3 described in Chapter 3, which is heterogeneous many-core supporting the running of multiple concurrent applications. The interacting functional steps are described in Figure 6.2.

Information of the workloads executing on platforms is collected by way of performance counters and power monitors, both available from the platforms. From this information two different actions are performed. The first is WLC (described in detail in Chapter 5), which only needs to use performance counter events to derive the type of a particular workload. The second is deriving the performance and power models through such methods as LR and MLR (described in detail in Section 6.4), which uses both performance counter and power monitor information. The workload classification result and the performance and power models from the MLR step are used to derive the optimal runtime management.

6.4 POWER AND PERFORMANCE RELATED MODELS

We develop runtime power and performance models using MLR. Based on these models a runtime adaptation is derived to determine the

most energy-efficient system configurations for different application scenarios. These models and their hypotheses are further detailed below.

Figure 6.3 shows the power and performance for ***ferret***, ***fluidanimate***, and three concurrent applications. We hypothesize that the relation between IPS and the numbers of group of cores in heterogeneous systems can be approximated by the following expression:

$$\text{IPS} = \sum_{i=1}^{K} \alpha_i f_i x_i + \epsilon_2(x), \tag{6.1}$$

where the first term ($\sum_{i=1}^{K} \alpha_i x_i$) represents the frequency-independent performance components that depend on architectural configuration, K is the total number of voltage islands (i.e. group of cores), and $\epsilon_2(x)$ is the error term. In the case of Exynos 5422 (i.e. Odroid big.LITTLE) can be expressed as:

$$\text{IPS} = \alpha_1 f_1 x_1 + \alpha_2 f_2 x_2 + \epsilon_2, \tag{6.2}$$

where α_1, α_2, and ϵ_2 are coefficients which need to be determined for each operating frequency by using MLR. x_1, f_1, x_2, f_2 are the numbers and frequencies of LITTLE and big cores respectively. All models have R-squared values greater than 0.95 showing the applicability of this model hypothesis.

6.5 RTM WORKLOAD CLASSIFICATIONS

The workload classes chosen for this work reflect the desire of differentiating CPU-intensive workloads from memory-intensive ones and differentiating high-activity workloads from low-activity ones as described in Chapter 5. Metrics and threshold values used to determine the application class are explained in Chapter 5.

6.6 LOW-COMPLEXITY RUNTIME

Figure 6.4 shows how to use the workload classification technique to reduce the complexity of applications and the complexity of systems hardware to strike the right tradeoff.

The first thing is to update the application queue - during the preceding period existing interval could have completed and a new interval could have started. If during the previous period, a new interval have started, Algorithm 5.1 is used to determine the application class of each application in this new interval, which has been explained in Chapter 5. Based on application class the state space has been

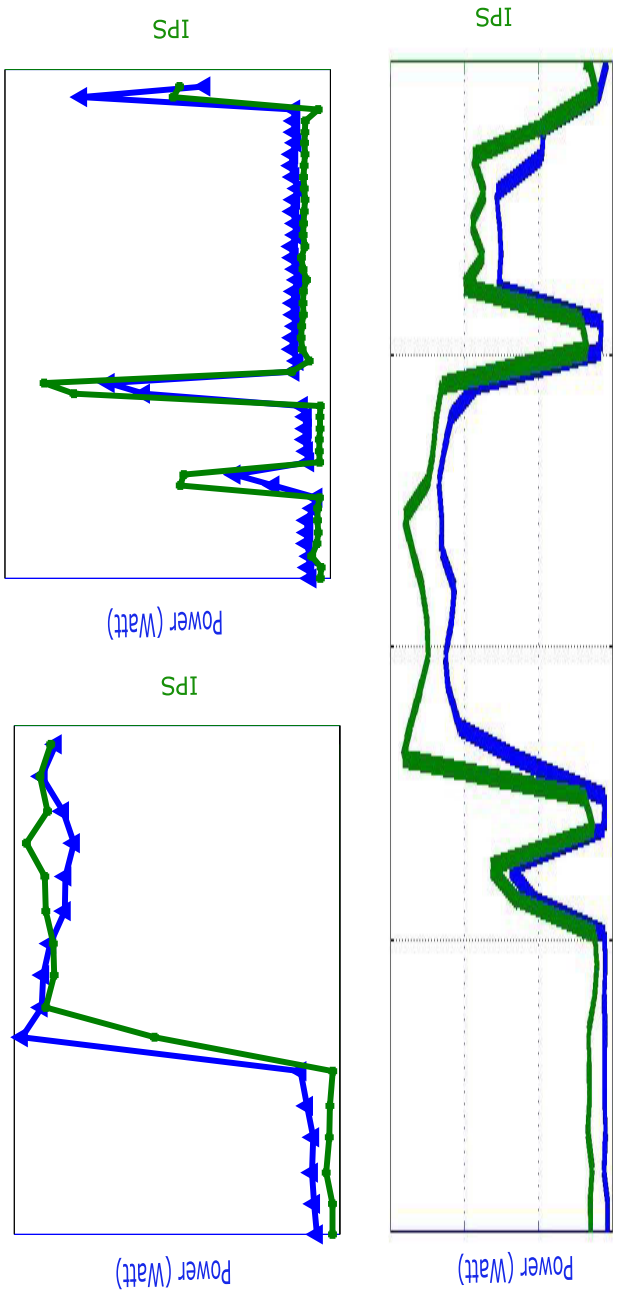


Figure 6.3: Power/performance characteristics for ***ferret***, ***fluidanimate***, and three different concurrent applications

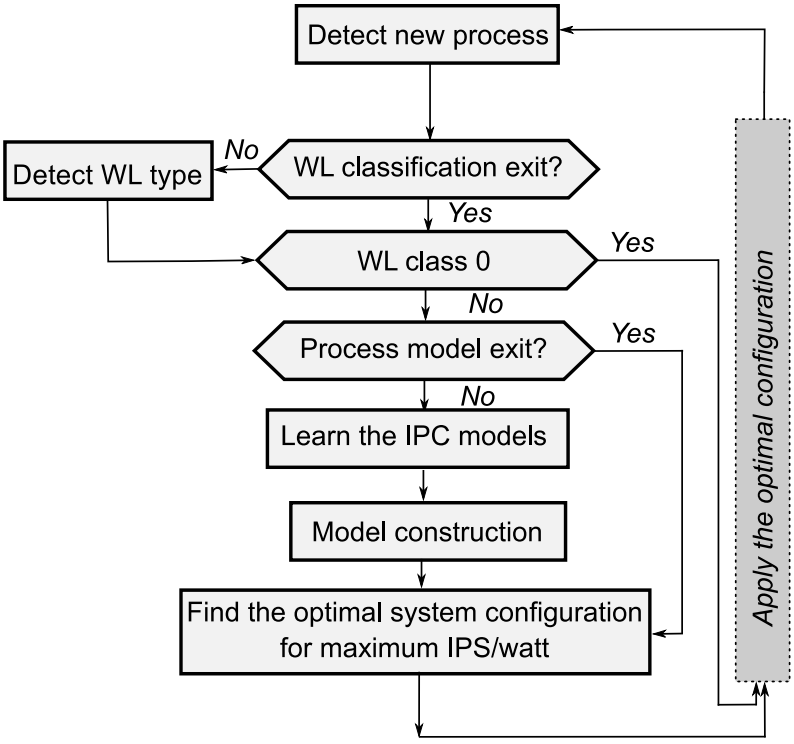


Figure 6.4: Simulation flow to obtain the results.

reduced. For example, in class 0 the search of optimal configuration for odroid XU-3 has been reduced from 4004 different frequency and core configurations (4*13*4*19 four A7 and four A15 can run at 13 and 19 different frequencies respectively) to one by using C0 and frequency=200 MHz as optimal configuration, in class 1 the search for optimal configuration has been reduced by more than 75% because we used the big cores and high frequencies for A15 (800-2000) MHz, and the state space has been reduced by more than 80% in class 3 because we used the big cores and high frequencies for A7 (800-1400) MHz. After that, MLR is used to determine the optimal frequency and core allocations for each class type.

6.7 PROPOSED RUNTIME RESULTS

Data collected from our large number of validation runs shows the RTM out-performing the Linux ondemand governor by considerable margins on IPS/Watt, as shown in Table 6.2. This table shows the comparison between three runtime optimization techniques model-based by using MLR, model-free by using workload classification, and low-complexity by combining workload classifications with MLR. Two observations can be made from these results. Firstly; the improvements

in IPS/Watt of the model-free technique is higher than model-based technique because the model-based optimization is per application while WLC optimization is per interval. Secondly, the improvements in IPS/Watt of the low-complexity technique is higher than model-free technique because of using workload classification with MLR determine optimal number of cores and a finer-grain control of their frequencies.

Table 6.2: Percentage IPS/WATT Improvements of the RTM over the LINUX Ondemand Governor

Application Scenarios	WLC	MLR	MLR+WLC
Fluidanimate alone	127.0%	127.0%	139.0%
Two different class applications	68.6%	61.7%	128.4%
Three different class applications	46.6%	29.3%	61.2%
Two Class 3 applications	24.5%	19.8%	40.3%
Three Class 3 applications	44.4%	36.4%	58.2%
Two Class 1 applications	31.0%	26.5%	41.7%

6.8 SUMMARY AND CONCLUSION

This chapter presents a novel runtime optimization approach demonstrated on heterogeneous systems, capable of workload classification and power-aware performance adaptation under sequential and concurrent application scenarios in heterogeneous many-core systems.

This approach is based on workload classification with performance model that can be obtained during runtime by multivariate linear regression based on low-complexity hypotheses of power and performance for a given operating frequency. The approach is extensively evaluated using PARSEC-3.0 benchmark suite running on the Odroid-XU3 heterogeneous platform.

A selection of experimental results was presented to illustrate the kinds of tradeoffs in a variety of concurrent application scenarios, core allocations, and DVFS points, highlighting an improvement of power normalized performance which produced IPS/Watt improvements between 42% and 139% for a range of applications. It is expected that modern embedded and high-performance system designers will benefit from the proposed approach in terms of a systematic power-aware performance optimization under variable workload and application scenarios.

7 CONCLUSIONS AND FUTURE WORK

The overall objective of this research is to develop methods for improving energy efficiency (in terms of IPS/Watt) for multi/many core heterogeneous systems running heterogeneous concurrent workloads. To meet this objectives, a novel runtime optimization approach is proposed for multiple concurrent applications of diverse workloads considering dynamic variation of workload and application scenarios running on heterogeneous multi-core platforms. Core to this approach is an empirical and data-driven method, which classifies applications based on their memory and CPU requirements. A low-complexity Linux power governor is implemented for the runtime control of DVFS and task mapping using information from system performance counters . A number of synthetic and real-world benchmark applications with different concurrent combinations are used to validate the approaches developed in this thesis. The remainder of this chapter is organised as follows. Section 7.1 gives a summary of thesis contribution and Section 7.2 presents a number of future related research opportunities opened up by this work.

7.1 SUMMARY AND CONCLUSION

Modern embedded systems execute multiple applications, both sequentially and concurrently. These applications may be executed on heterogeneous platforms generating varying power consumption and system workloads (CPU or memory intensive or both). This increasing system complexity in both hardware and software emphasizes a major challenge for computing systems, especially mobile and embedded systems, namely the performance-energy tradeoff. As a result, determining the most energy-efficient system configuration (i.e. the number of parallel threads, their core allocations and operating frequencies) tailored for each kind of workload and application scenario is extremely challenging. Underpinning these many core heterogeneous power/performance tradeoff design issues and challenges, the following contributions have been made in this thesis:

Platform exploration experiments: Chapter 3 present a comprehensive study that profiles the power consumption and performance characteristics of a multi-threaded many-core system, associating power consumption and performance with multiple applications. These applications are executed on an Odroid XU3 heterogeneous platform generating varying power consumption and system workloads (CPU or memory intensive or both). Core to this study is an investigation of

the tradeoffs between inter-application concurrency with performance and power consumption under different system configurations. These experiments present the basic method of characterization experimentation which is consistently used throughout the thesis in Chapters 4, 5, and 6.

Model-based runtime management of concurrent workloads A novel runtime optimization approach for single and concurrent applications is presented in Chapter 4. Real experimental measurements on an Odroid XU-3 heterogeneous platform with a number of PARSEC benchmark applications to profile real power consumption and performance measurement for single and concurrent applications. For the first time, this work reveals the impact of parallelism in different types of heterogeneous cores on performance, power consumption and power efficiency in terms of IPS/Watt. This approach is based on power and performance models that can be obtained during runtime by multivariate linear regression based on low-complexity hypotheses of power and performance for a given operating frequency. Using these models, this approach shows that CPU intensive applications show different gains in IPS/Watt compared to memory intensive applications in both sequential and concurrent application scenarios. Furthermore, this work demonstrates that it is possible to continuously adapt system configuration through a low-cost and linear-complexity runtime algorithm, which can improve the IPS/Watt by up to 125% compared to the existing approach.

Model-free runtime management of concurrent workloads: A runtime adaptation approach to improve the energy efficiency of a heterogeneous many-core system with concurrent workloads is presented in Chapter 5. Core to this approach is an empirical and data-driven method, which classifies applications based on their memory and CPU requirements. The aim is to derive DVFS and task mapping policies, tailored to the classified workloads without requiring any explicit modelling at runtime. Due to simplified runtime classification, this approach can significantly reduce overheads. Furthermore, the model-free classification based runtime enhances the scalability for any concurrent application mix, platform, and metric. It has linear complexity which is not affected by the system heterogeneity and the number of concurrent applications. Extensive experiments on an Odroid XU3 heterogeneous and i7 homogeneous platforms with synthetic and standard benchmark applications are used to develop the control policies and validate this approach. These experiments show that workload classification into CPU-intensive and memory-intensive types provides the foundation for scalable energy minimization with low complexity. Furthermore, the expermental results show that IPS/Watt can be improved by over 120% compared to existing approaches.

Reduced-complexity runtime management of concurrent workloads: Chapter 6 presents reduced-complexity runtime approach for

energy minimization of single and concurrent applications. This approach is demonstrated by a governor aimed at improving system energy efficiency (IPS/Watt). This governor combines the methods presented in Chapter 4 and Chapter 5, classifies applications according to their CPU and memory signatures, uses the classifier decision to reduce the complexity of state-space learning-based runtime from Chapter 4, and make decisions on core allocation and DVFS. Moreover, the experimental results of this approach show that it is possible to continuously adapt system configuration through a low-cost and linear-complexity runtime algorithm, which can improve the IPS/Watt by up to 136% compared to the existing approach.

7.2 FUTURE WORK

Runtime workload classification and management for energy-efficient many-core systems has many opportunities for future work.

Two worthy and interesting research recommendations for expanding and exploring the proposed work in this thesis are presented in this section.

The proposed model-based power-aware performance adaptation of concurrent applications in heterogeneous many-core systems can be expanded by taking into account the effect of idle power on the energy-efficiency by using additional effective runtime power management techniques such as CMOS power gating in addition to DVFS and task mapping for reducing the overall power consumption. Further exploration of other types of heterogeneity such as using GPU with CPU, and memory hierarchies is an interesting area of future research.

In terms of using model-free runtime Management of Concurrent Workloads for Energy-Efficient Many-Core Heterogeneous Systems, This work opens up opportunities for future runtime management research including the runtime tuning of such parameters as classification thresholds, control decisions, and RTM control cycles. Another promising direction is including investigation of the scalability of these approaches to more complex platforms and higher levels of concurrency.

Part II

Thesis Appendices

A RUNTIME MANGEMENT PROGRAMM

The source code of the runtime program is included below. It may be downloaded from http://async.org.uk/data/runtime-2019/.

```
---------------------------------------------------------------------------

#define DEBUG_INFO 1

#include "timeutils.h"

#include "affinity.h"

#include "monitors.h"

#include "exec.h"


// RTM --------------------------------

pthread_t rtm_thread_id = -1;

int req_stop_rtm = 0;


static void* rtm_proc(void *arg)

{

	int i, tnum, max, tmax;

	start_core_monitors();

	printf("rtm: started\n");


	#if READ_XU3POWER

		float mon_xu3power[NUM_XU3POWER_PARAMS];

		for(i=0; i<NUM_XU3POWER_PARAMS; i++)

			mon_xu3power[i] = 0.0;

	#endif


	for(i=0; req_stop_rtm==0; i++) {

		sleep_ms(MON_PERIOD);
```

```
		// collect monitor readings
		#if READ_XU3POWER
			if(read_xu3power_all(mon_xu3power)) {
				printf("XU3 power mon:
%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\n",
					mon_xu3power[0], mon_xu3power[1], mon_xu3power[2],
mon_xu3power[3],
					mon_xu3power[4], mon_xu3power[5], mon_xu3power[6],
mon_xu3power[7]);
			}
			else {
				printf("*error: read_xu3power_all\n");
			}
		#endif

		max = 0;
		for(tnum = 0; tnum < NUM_CORES; tnum++) {
			if(monitors_tinfo[tnum].mon_inst_retired>max) {
				max = monitors_tinfo[tnum].mon_inst_retired;
				tmax = tnum;
			}
		}

		// TODO: control
		printf("rtm[%d]: max %d @ C%d\n", i, max, tmax);

		next_unused_core = get_unused_core();
	}
	stop_core_monitors();
	printf("rtm: stopped\n");
}
```

```
void start_rtm()
{
	int err;
	pthread_attr_t attr;
	cpu_set_t cpus;

	// set affinity
	pthread_attr_init(&attr);
	CPU_ZERO(&cpus);
	CPU_SET(0, &cpus);
	err = pthread_attr_setaffinity_np(&attr, sizeof(cpu_set_t), &cpus);
	if(err)
		printf("*error(%d): pthread_attr_setaffinity_np, %d\n", __LINE__, err);

	// create_thread
	req_stop_rtm = 0;
	err = pthread_create(&rtm_thread_id, &attr, rtm_proc, NULL);
	if(err)
		printf("*error(%d): pthread_create, %d\n", __LINE__, err);
}

void stop_rtm()
{
	req_stop_rtm = 1;
	pthread_join(rtm_thread_id, NULL);
}

// MAIN ---------------------------------

int main()
```

```
{
        #if READ_XU3POWER
                init_xu3power();
        #endif
        init_affinity();
        start_rtm();
        run_exec();
        stop_rtm();
}
```

```
#include "affinity.h"

const int core_names[NUM_CORES] = {4, 5, 6, 7, 0, 1, 2, 3};

app_info apps_info[MAX_TASKS];
app_info* core_app_map[NUM_CORES];
int next_app_index = 0;
int next_unused_core = 0;

void init_affinity()
{
	int i;
	for(i=0; i<NUM_CORES; i++)
		core_app_map[i] = NULL;
	for(i=0; i<MAX_TASKS; i++)
		apps_info[i].pid = 0;
}

int get_unused_core()
{
	int i;
	for(i=0; i<NUM_CORES; i++) {
		if(core_app_map[i]==NULL)
			return i;
	}
	printf("*error: no available cores\n");
	return 0;
}

void set_app_affinity(app_info* app, int ncores, int cores[])
```

```
{
	int i, err;
	cpu_set_t cpus;
	CPU_ZERO(&cpus);
	app->ncores = ncores;
	#if DEBUG_INFO
		printf("app <%d> affinity: ", app->pid);
	#endif
	for(i=0; i<ncores; i++) {
		CPU_SET(core_names[cores[i]], &cpus);
		#if DEBUG_INFO
			printf("%d ", core_names[cores[i]]);
		#endif
		app->cores[i] = cores[i];
		core_app_map[cores[i]] = app;
	}
	#if DEBUG_INFO
		printf("\n");
	#endif
	err = sched_setaffinity(app->pid, sizeof(cpu_set_t), &cpus);
	if(err)
		printf("*error(%d): sched_setaffinity, %d\n", __LINE__, err);
}

void create_app_info(int task_id, pid_t pid)
{
	#if DEBUG_INFO
		printf("New app <%d> for task %d\n", pid, task_id);
	#endif
	int i = next_app_index++;
	apps_info[i].task_id = task_id;
```

```
        apps_info[i].pid = pid;
        apps_info[i].start_time = timestamp();
        int cores[] = {next_unused_core};
        set_app_affinity(&apps_info[i], 1, cores);
}
```

```
#include "exec.h"

char* tasks[][MAX_ARGS] = {
	{"./sqrt", "100000", NULL},
	{"ls", NULL},
	{"/root/work/parsec-3.0/pkgs/apps/fluidanimate/inst/arm-linux.gcc-pthreads/bin/fluidanimate",
		"1", "5",
		"/root/work/parsec-3.0/pkgs/apps/fluidanimate/inputs/in_300K.fluid",
		"/root/work/parsec-3.0/out.fluid", NULL}
};

int n_pid = 0;
pid_t pid_list[MAX_TASKS];

int start_task(int id)
{
	printf("Start task %d (%s)\n", id, tasks[id][0]);
	pid_t pid = fork();
	if(pid!=0) {
		pid_list[n_pid++] = pid;
		create_app_info(id, pid);
		return pid;
	}
	else {
		execvp(tasks[id][0], tasks[id]);
		perror("*error");
		exit(1);
		return 0;
```

```
        }
}

void wait_all()
{
        int status;
        while(n_pid>0) {
                waitpid(pid_list[--n_pid], &status, 0);
        }
}

void run_exec()
{
        int sleep, id;
        unsigned long tstart = timestamp();
        while(scanf("%d %d", &sleep, &id)==2) {
                sleep_ms(sleep);
                if(start_task(id)==0)
                        return;
        }
        wait_all();
        printf("Done exec, total time: %ld\n", timestamp()-tstart);
}
```

```
#include "monitors.h"

#if READ_XU3POWER
#define NUM_XU3POWER_FLAGS 4
const char* xu3power_flag_paths[NUM_XU3POWER_FLAGS] = {
        "/sys/bus/i2c/drivers/INA231/3-0045/enable",
        "/sys/bus/i2c/drivers/INA231/3-0040/enable",
        "/sys/bus/i2c/drivers/INA231/3-0041/enable",
        "/sys/bus/i2c/drivers/INA231/3-0044/enable",
};

const char* xu3power_param_paths[NUM_XU3POWER_PARAMS] = {
        "/sys/bus/i2c/drivers/INA231/3-0045/sensor_V", // A7 V
        "/sys/bus/i2c/drivers/INA231/3-0045/sensor_A",
        "/sys/bus/i2c/drivers/INA231/3-0045/sensor_W",
        "/sys/bus/i2c/drivers/INA231/3-0040/sensor_V", // A15 V
        "/sys/bus/i2c/drivers/INA231/3-0040/sensor_A",
        "/sys/bus/i2c/drivers/INA231/3-0040/sensor_W",
        "/sys/devices/system/cpu/cpu3/cpufreq/scaling_cur_freq",
        "/sys/devices/system/cpu/cpu7/cpufreq/scaling_cur_freq",
};
#endif

struct monitor_info monitors_tinfo[NUM_CORES];

// XU3 POWER ------------------------------

#if READ_XU3POWER

int read_xu3power(int param, const char* fmt, void* ptr)
```

```
{
	FILE* fp;
	fp = fopen(xu3power_param_paths[param], "r");
	if(fp==NULL) {
		return 0;
	}
	if(fscanf(fp, fmt, ptr)!=1) {
		fclose(fp);
		return 0;
	}
	else {
		fclose(fp);
		return 1;
	}
}

int read_xu3power_all(float* ptr)
{
	int i, res = 1;
	for(i=0; i<NUM_XU3POWER_PARAMS; i++) {
		res &= read_xu3power(i, "%f", &ptr[i]);
	}
}

void set_xu3power_flag(const char* path, const char* flag)
{
	FILE* fp;
	fp = fopen(path, "w");
	if(fp==NULL)
		return;
	fputs(flag, fp);
```

```
	fclose(fp);
}

void init_xu3power()
{
	int i;
	for(i=0; i<NUM_XU3POWER_FLAGS; i++) {
		set_xu3power_flag(xu3power_flag_paths[i], "1");
	}
	sleep_ms(2000);
}

#endif

// ARM PMU -------------------------------

#if READ_ARMPMU
#include "pmu.h"



#endif

// CORE MONITOR THREADS ------------------------------

static void* monitor_proc(void *arg)
{
	int i;
	int num = ((monitor_info*) arg)->num;
	FILE* log = open_log_core("mon", num);
	if(log) fprintf(log, "time\tunhalt_cycles\tinst_retired\tmem_access\n");
	printf("mon%d: started\n", num);
```

```
	cpu_set_t cpuset;
	CPU_ZERO(&cpuset);
	CPU_SET(num, &cpuset);
	sched_setaffinity(0, sizeof(cpuset), &cpuset);

	long mon_time, prev_time;
	unsigned int mon_unhalt_cycles, mon_inst_retired, mon_mem_access;
	unsigned int prev_unhalt_cycles, prev_inst_retired, prev_mem_access;
	int first = 1;

	#if READ_ARMPMU
		init_perf_start();
		init_perf(0, 0x08 /* INST_RETIRED */);
		init_perf(1, 0x13 /* MEM_ACCESS */);
		// init_perf(0x08 /* INST_RETIRED */, 0x13 /* MEM_ACCESS */, 2, 3, 0x08 /*
INST_RETIRED */, 0x13 /* MEM_ACCESS */);
	#endif

	for(i=0;; i++) {
		sleep_ms(MON_PERIOD);

		// collect monitor data
		mon_time = (long)timestamp();
		#if READ_ARMPMU
			mon_unhalt_cycles = (int)get_cyclecnt();
			get_evt(0, &mon_inst_retired);
			get_evt(1, &mon_mem_access);
			// get_evt(&mon_inst_retired, &mon_mem_access, &mon_2, &mon_3,
&mon_4, &mon_5);
		#else
			mon_unhalt_cycles = 0;
```

```
                        mon_inst_retired = 0;
                        mon_mem_access = 0;
                #endif

                if(!first) {
                        ((monitor_info*) arg)->mon_time = mon_time - prev_time;
                        ((monitor_info*) arg)->mon_unhalt_cycles = mon_unhalt_cycles -
prev_unhalt_cycles;
                        ((monitor_info*) arg)->mon_inst_retired = mon_inst_retired -
prev_inst_retired;
                        ((monitor_info*) arg)->mon_mem_access = mon_mem_access -
prev_mem_access;

                        // log data
                        if(log) fprintf(log, "%ld\t%d\t%d\t%d\n",
                                        ((monitor_info*) arg)->mon_time,
                                        ((monitor_info*) arg)->mon_unhalt_cycles,
                                        ((monitor_info*) arg)->mon_inst_retired,
                                        ((monitor_info*) arg)->mon_mem_access
                                );
                }

                prev_time = mon_time;
                prev_unhalt_cycles = mon_unhalt_cycles;
                prev_inst_retired = mon_inst_retired;
                prev_mem_access = mon_mem_access;
                first = 0;
        }
}

void start_core_monitors()
{
```

```
	int tnum, err;
	pthread_attr_t attr;
	cpu_set_t cpus;

	pthread_attr_init(&attr);
	for(tnum = 0; tnum < NUM_CORES; tnum++) {
		monitors_tinfo[tnum].num = tnum;

		// set affinity
		CPU_ZERO(&cpus);
		CPU_SET(tnum, &cpus);
		err = pthread_attr_setaffinity_np(&attr, sizeof(cpu_set_t), &cpus);
		if(err)
			printf("*error(%d): pthread_attr_setaffinity_np, %d\n", __LINE__, err);

		// start thread
		err = pthread_create(&monitors_tinfo[tnum].thread_id, &attr, monitor_proc,
&monitors_tinfo[tnum]);
		if(err)
			printf("*error(%d): pthread_create, %d\n", __LINE__, err);
	}
}

void stop_core_monitors()
{
	int tnum;
	for(tnum = 0; tnum < NUM_CORES; tnum++) {
		pthread_cancel(monitors_tinfo[tnum].thread_id);
	}
}
```

```
#include "timeutils.h"

void sleep_ms(int ms)
{
	struct timespec t;
	if(ms>0) {
		t.tv_sec  = ms/1000;
		t.tv_nsec = (ms%1000)*1000000L;
		nanosleep(&t, NULL);
	}
}

unsigned long timestamp()
{
	struct timespec t;
	clock_gettime(CLOCK_REALTIME, &t);
	return (t.tv_sec)*1000L + (t.tv_nsec)/1000000L;
}
```

Part III

Thesis Bibliography

BIBLIOGRAPHY

[1] ARM, big.LITTLE Technology,. http://www.arm.com/products/processors/technologies/biglittleprocessing.php.

[2] Odroid XU3. http://www.hardkernel.com/main/products, .

[3] Parsec 3.0. http://parsec.cs.princeton.edu/parsec3-doc.htm, .

[4] Ali Aalsaud, Rishad Shafik, Ashur Rafiev, Fie Xia, Sheng Yang, and Alex Yakovlev. 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*, pages 368–373. ACM, 2016.

[5] Ali Aalsaud, Haider Alrudainv, Rishad Shafik, Fei Xia, and Alex Yakovlev. MEMS-Based Runtime Idle Energy Minimization for Bursty Workloads in Heterogeneous Many-Core Systems. In *2018 28th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, pages 198–205. IEEE, 2018.

[6] Ali Aalsaud, Ashur Rafiev, Fei Xia, Rishad Shafik, and Alex Yakovlev. Model-Free Runtime Management of Concurrent Workloads for Energy-Efficient Many-Core Heterogeneous Systems. In *2018 28th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, pages 206–213. IEEE, 2018.

[7] Behnam Amelifard and Massoud Pedram. Design of an efficient power delivery network in an SoC to enable dynamic power management. In *Proceedings of the 2007 international symposium on Low power electronics and design*, pages 328–333. ACM, 2007.

[8] Rabie Ben Atitallah, Eric Senn, Daniel Chillet, Mickael Lanoe, and Dominique Blouin. An efficient framework for power-aware design of heterogeneous MPSoC. *IEEE Transactions on Industrial Informatics*, 9(1):487–501, 2013.

[9] Luca Benini, Alessandro Bogliolo, and Giovanni De Micheli. A survey of design techniques for system-level dynamic power management. *IEEE transactions on very large scale integration (VLSI) systems*, 8(3):299–316, 2000.

[10] Christian Bienia and Kai Li. Parsec 2.0: A new benchmark suite for chip-multiprocessors. In *Proceedings of the 5th Annual Workshop on Modeling, Benchmarking and Simulation*, volume 2011, 2009.

[11] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 39(2):1–7, 2011.

[12] Partha Biswas, Sudarshan Banerjee, Nikil Dutt, Paolo Ienne, and Laura Pozzi. Performance and energy benefits of instruction set extensions in an FPGA soft core. In *VLSI Design, 2006. Held jointly with 5th International Conference on Embedded Systems and Design., 19th International Conference on*, pages 6–pp. IEEE, 2006.

[13] Ramazan Bitirgen, Engin Ipek, and Jose F Martinez. Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach. In *Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture*, pages 318–329. IEEE Computer Society, 2008.

[14] Shekhar Borkar. Design challenges of technology scaling. *IEEE micro*, (4):23–29, 1999.

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

[16] Bishop Brock and Karthick Rajamani. Dynamic power management for embedded systems [soc design]. In *SOC Conference, 2003. Proceedings. IEEE International [Systems-on-Chip]*, pages 416–419. IEEE, 2003.

[17] Aaron Carroll, Gernot Heiser, et al. An analysis of power consumption in a smartphone. In *USENIX annual technical conference*, volume 14, pages 21–21. Boston, MA, 2010.

[18] Koushik Chakraborty, Philip M Wells, and Gurindar S Sohi. A case for an over-provisioned multicore system: Energy efficient processing of multithreaded programs. *Department of Computer Sciences, University of Wisconsin-Madiso, Tech. Rep*, 2007.

[19] Anantha P Chandrakasan and Robert W Brodersen. Minimizing power consumption in digital CMOS circuits. *Proceedings of the IEEE*, 83(4):498–523, 1995.

[20] Zhuo Chen and Diana Marculescu. Distributed reinforcement learning for power limited many-core system performance optimization. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, pages 1521–1526. EDA Consortium, 2015.

[21] Kihwan Choi, Ramakrishna Soma, and Massoud Pedram. Dynamic voltage and frequency scaling based on workload decomposition. In *Proceedings of the 2004 international symposium on Low power electronics and design*, pages 174–179. ACM, 2004.

[22] Kihwan Choi, Ramakrishna Soma, and Massoud Pedram. Fine-grained dynamic voltage and frequency scaling for precise energy and performance tradeoff based on the ratio of off-chip access to on-chip computation times. *IEEE transactions on computer-aided design of integrated circuits and systems*, 24(1):18–28, 2005.

[23] Hongsuk Chung, Munsik Kang, and Hyun-Duk Cho. Heterogeneous Multi-Processing Solution of Exynos 5 Octa with ARM® big. LITTLE Technology. *Samsung White Paper*, 2012.

[24] Ryan Cochran, Can Hankendi, Ayse K Coskun, and Sherief Reda. Pack & Cap: adaptive DVFS and thread packing under power caps. In *Microarchitecture (MICRO), 2011 44th Annual IEEE/ACM International Symposium on*, pages 175–185. IEEE, 2011.

[25] Matthew Curtis-Maury, Ankur Shah, Filip Blagojevic, Dimitrios S Nikolopoulos, Bronis R De Supinski, and Martin Schulz. Prediction models for multi-dimensional power-performance optimization on many cores. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pages 250–259. ACM, 2008.

[26] Matthew Curtis-Maury, Ankur Shah, Filip Blagojevic, Dimitrios S Nikolopoulos, Bronis R De Supinski, and Martin Schulz. Prediction models for multi-dimensional power-performance optimization on many cores. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pages 250–259. ACM, 2008.

[27] Gaurav Dhiman and Tajana Simunic Rosing. System-level power management using online learning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(5): 676–689, 2009.

[28] Bryan Donyanavard, Tiago Mück, Santanu Sarma, and Nikil Dutt. Sparta: Runtime task allocation for energy efficient heterogeneous manycores. In *Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2016 International Conference on*, pages 1–10. IEEE, 2016.

[29] Farshad Firouzi, Mostafa E Salehi, Fan Wang, Sied Mehdi Fakhraie, and Saeed Safari. Reliability-aware dynamic voltage and frequency scaling. In *VLSI (ISVLSI), 2010 IEEE Computer Society Annual Symposium on*, pages 304–309. IEEE, 2010.

[30] Rem Gensh, Ali Aalsaud, Ashur Rafiev, Fei Xia, Alexei Iliasov, Alexander Romanovsky, and Alex Yakovlev. *Experiments with odroid-xu3 board*. Newcastle University, Computing Science, 2015.

[31] Marco ET Gerards, Johann L Hurink, Philip KF Hölzenspies, Jan Kuper, and Gerard JM Smit. Analytic clock frequency selection for global DVFS. In *Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on*, pages 512–519. IEEE, 2014.

[32] Lee Kee Goh, Bharadwaj Veeravalli, and Sivakumar Viswanathan. Design of fast and efficient energy-aware gradient-based scheduling algorithms heterogeneous embedded multiprocessor systems. *IEEE Transactions on Parallel and Distributed Systems*, 20(1):1–12, 2009.

[33] Michel Goraczko, Jie Liu, Dimitrios Lymberopoulos, Slobodan Matic, Bodhi Priyantha, and Feng Zhao. Energy-optimal software partitioning in heterogeneous multiprocessor embedded systems. In *Proceedings of the 45th annual design automation conference*, pages 191–196. ACM, 2008.

[34] Ujjwal Gupta. *Power-Performance Modeling and Adaptive Management of Heterogeneous Mobile Platforms*. PhD thesis, Arizona State University, 2018.

[35] Ujjwal Gupta, Spurthi Korrapati, Navyasree Matturu, and Umit Y Ogras. A generic energy optimization framework for heterogeneous platforms using scaling models. *Microprocessors and Microsystems*, 40:74–87, 2016.

[36] Ujjwal Gupta, Chetan Arvind Patil, Ganapati Bhat, Prabhat Mishra, and Umit Y Ogras. Dypo: Dynamic pareto-optimal configuration selection for heterogeneous mpsocs. *ACM Transactions on Embedded Computing Systems (TECS)*, 16(5s):123, 2017.

[37] Mohammad-Hashem Haghbayan, Amir-Mohammad Rahmani, Pasi Liljeberg, Juha Plosila, and Hannu Tenhunen. Online testing of many-core systems in the dark silicon era. In *Design and Diagnostics of Electronic Circuits & Systems, 17th International Symposium on*, pages 141–146. IEEE, 2014.

[38] Can Hankendi and Ayse K Coskun. Adaptive power and resource management techniques for multi-threaded workloads. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International*, pages 2302–2305. IEEE, 2013.

[39] Sebastian Herbert and Diana Marculescu. Analysis of dynamic voltage/frequency scaling in chip-multiprocessors. In *Low Power*

Electronics and Design (ISLPED), 2007 ACM/IEEE International Symposium on, pages 38–43. IEEE, 2007.

[40] Arliones Stevert Hoeller, Lucas Francisco Wanner, and Antônio Augusto Fröhlich. A hierarchical approach for power management on mobile embedded systems. In *From Model-Driven Design to Resource Management for Distributed Embedded Systems*, pages 265–274. Springer, 2006.

[41] Chung-hsing Hsu and Wu-chun Feng. A power-aware run-time system for high-performance computing. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 1. IEEE Computer Society, 2005.

[42] Shaoxiong Hua and Gang Qu. Approaching the maximum energy saving on embedded systems with multiple voltages. In *Proceedings of the 2003 IEEE/ACM international conference on Computer-aided design*, page 26. IEEE Computer Society, 2003.

[43] Quentin JM Huys, Anthony Cruickshank, and Peggy Seriès. Reward-based learning, model-based and model-free. In *Encyclopedia of Computational Neuroscience*, pages 2634–2641. Springer, 2015.

[44] Canturk Isci, Alper Buyuktosunoglu, Chen-Yong Cher, Pradip Bose, and Margaret Martonosi. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In *Proceedings of the 39th annual IEEE/ACM international symposium on microarchitecture*, pages 347–358. IEEE Computer Society, 2006.

[45] Jaeseok Jeon. Advanced relay design and technology for energy-efficient electronics. Technical report, California University Berkeley department of electrical engineering and computer science, 2011.

[46] Bojan Jovanović and Milun Jevtić. Static and dynamic power consumption of arithmetic circuits in modern technologies. In *55th Conference for Electronics, Telecommunications, Computers, Automation, and Nuclear Engineering*, volume 55, pages 1–4, 2011.

[47] Da-Cheng Juan and Diana Marculescu. Power-aware performance increase via core/uncore reinforcement control for chip-multiprocessors. In *Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design*, pages 97–102. ACM, 2012.

[48] Da-Cheng Juan, Siddharth Garg, Jinpyo Park, and Diana Marculescu. Learning the optimal operating point for many-core systems with extended range voltage/frequency scaling. In

Proceedings of the Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, page 8. IEEE Press, 2013.

[49] Rakhee Kallimani and Krupa Rasane. A survey of techniques for power management in embedded systems. pages 461–462, Volume=14, Issue = 2, April 2015.

[50] James T Kao, Masayuki Miyazaki, and AR Chandrakasan. A 175-mv multiply-accumulate unit using an adaptive supply voltage and body bias architecture. *IEEE journal of solid-state circuits*, 37 (11):1545–1554, 2002.

[51] Steven P Kerrison. *Energy modelling of multi-threaded, multi-core software for embedded systems.* PhD thesis, University of Bristol, 2015.

[52] Nam Sung Kim, Todd Austin, David Baauw, Trevor Mudge, Krisztián Flautner, Jie S Hu, Mary Jane Irwin, Mahmut Kandemir, and Vijaykrishnan Narayanan. Leakage current: Moore's law meets static power. *computer*, 36(12):68–75, 2003.

[53] Shin-gyu Kim, Hyeonsang Eom, Heon Y Yeom, and Sang Lyul Min. Energy-centric DVFS controlling method for multi-core platformswonyoung. *Computing*, 96(12):1163–1177, 2014.

[54] Wonyoung Kim, Meeta S Gupta, Gu-Yeon Wei, and David Brooks. System level analysis of fast, per-core DVFS using on-chip switching regulators. In *High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on*, pages 123–134. IEEE, 2008.

[55] Sumeet S Kumar, Mitzi Tjin A Djie, and Rene Van Leuken. Low overhead message passing for high performance many-core processors. In *2013 First International Symposium on Computing and Networking-Across Practical Development and Theoretical Research (CANDAR)*, pages 345–351. IEEE, 2013.

[56] Christos Kyrkou, Christos-Savvas Bouganis, Theocharis Theocharides, and Marios M Polycarpou. Embedded hardware-efficient real-time classification with cascade support vector machines. *IEEE transactions on neural networks and learning systems*, 27(1):99–112, 2016.

[57] Etienne Le Sueur and Gernot Heiser. Dynamic voltage and frequency scaling: The laws of diminishing returns. In *Proceedings of the 2010 international conference on Power aware computing and systems*, pages 1–8, 2010.

[58] Suk-Bok Lee, Sai-Wang Tam, Ioannis Pefkianakis, Songwu Lu, M. Frank Chang, Chuanxiong Guo, Glenn Reinman, Chunyi

Peng, Mishali Naik, Lixia Zhang, and Jason Cong. A Scalable Micro Wireless Interconnect Structure for CMPs. In *Proceedings of the 15th Annual International Conference on Mobile Computing and Networking*, MobiCom '09, pages 217–228, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-702-8. doi: 10.1145/1614320.1614345. URL http://doi.acm.org/10.1145/1614320.1614345.

[59] Charles Leech, Charan Kumar, Amit Acharyya, Sheng Yang, Geoff V Merrett, and Bashir M Al-Hashimi. Runtime performance and power optimization of parallel disparity estimation on many-core platforms. *ACM Transactions on Embedded Computing Systems (TECS)*, 17(2):41, 2018.

[60] Oded Lempel. 2nd Generation Intel® Core Processor Family: Intel® Core i7, i5 and i3. In *Hot Chips 23 Symposium (HCS), 2011 IEEE*, pages 1–48. IEEE, 2011.

[61] Yves Lhuillier, Maroun Ojail, Alexandre Guerre, Jean-Marc Philippe, Karim Ben Chehida, Farhat Thabet, Caaliph Andriamisaina, Chafic Jaber, and Raphaël David. Hars: A hardware-assisted runtime software for embedded many-core architectures. *ACM Transactions on Embedded Computing Systems (TECS)*, 13(3s):102, 2014.

[62] Jian Li and Jose F Martinez. Dynamic power-performance adaptation of parallel computation on chip multiprocessors. In *High-Performance Computer Architecture, 2006. The Twelfth International Symposium on*, pages 77–87. IEEE, 2006.

[63] Sheng Li, Jung Ho Ahn, Richard D Strong, Jay B Brockman, Dean M Tullsen, and Norman P Jouppi. Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 469–480. ACM, 2009.

[64] Wei Liu, Ying Tan, and Qinru Qiu. Enhanced Q-learning algorithm for dynamic power management with performance constraint. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 602–605. European Design and Automation Association, 2010.

[65] Daniel Llamocca, Cesar Carranza, and Marios Pattichis. Separable FIR filtering in FPGA and GPU implementations: Energy, Performance, and Accuracy considerations. In *Field Programmable Logic and Applications (FPL), 2011 International Conference on*, pages 363–368. IEEE, 2011.

[66] Jiong Luo and Niraj K Jha. Power-efficient scheduling for heterogeneous distributed real-time embedded systems. *IEEE Transac-*

tions on Computer-Aided Design of Integrated Circuits and Systems, 26(6):1161–1170, 2007.

[67] Jun Ma, Guihai Yan, Yinhe Han, and Xiaowei Li. An analytical framework for estimating scale-out and scale-up power efficiency of heterogeneous manycores. *IEEE Transactions on Computers*, (1):1–1, 2016.

[68] Kai Ma, Xue Li, Ming Chen, and Xiaorui Wang. Scalable power control for many-core architectures running multi-threaded applications. In *ACM SIGARCH Computer Architecture News*, volume 39, pages 449–460. ACM, 2011.

[69] Mahesh Mamidipaka and Nikil Dutt. On-chip stack based memory organization for low power embedded architectures. In *Proceedings of the conference on Design, Automation and Test in Europe-Volume 1*, page 11082. IEEE Computer Society, 2003.

[70] Grant Martin. Overview of the mpsoc design challenge. In *Proceedings of the 43rd annual Design Automation Conference*, pages 274–279. ACM, 2006.

[71] Sparsh Mittal. A survey of techniques for improving energy efficiency in embedded computing systems. *International Journal of Computer Aided Engineering and Technology*, 6(4):440–459, 2014.

[72] Sparsh Mittal and Zhao Zhang. EnCache: Improving cache energy efficiency using a software-controlled profiling cache. *IEEE EIT*, 2012.

[73] Gordon E Moore. Gramming more components onto integrated circuits. *Electronics*, 38:8, 1965.

[74] Atukem Nabina and Jose Luis Nunez-Yanez. Adaptive voltage scaling in a dynamically reconfigurable FPGA-based platform. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 5(4):20, 2012.

[75] Venkatesh Pallipadi and Alexey Starikovskiy. The ondemand governor. In *Proceedings of the Linux Symposium*, volume 2, pages 215–230. sn, 2006.

[76] Gung-Yu Pan, Jing-Yang Jou, and Bo-Cheng Lai. Scalable power management using multilevel reinforcement learning for multiprocessors. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 19(4):33, 2014.

[77] Jun Cheol Park, Vincent Mooney, and Sudarshan K Srinivasan. Combining data remapping and voltage/frequency scaling of second level memory for energy reduction in embedded systems. *Microelectronics journal*, 34(11):1019–1024, 2003.

[78] Vinicius Petrucci, Orlando Loques, and Daniel Mossé. Lucky scheduling for energy-efficient heterogeneous multi-core systems. In *HotPower*, 2012.

[79] Roman Plyaskin, Alejandro Masrur, Martin Geier, Samarjit Chakraborty, and Andreas Herkersdorf. High-level timing analysis of concurrent applications on MPSoC platforms using memory-aware trace-driven simulations. In *VLSI System on Chip Conference (VLSI-SoC), 2010 18th IEEE/IFIP*, pages 229–234. IEEE, 2010.

[80] Viswanathan Lakshmi Prabha and Elwin Chandra Monie. Hardware architecture of reinforcement learning scheme for dynamic power management in embedded systems. *EURASIP Journal on Embedded Systems*, 2007(1):065478, 2007.

[81] Alok Prakash, Siqi Wang, Alexandru Eugen Irimiea, and Tulika Mitra. Energy-efficient execution of data-parallel applications on heterogeneous mobile platforms. In *Computer Design (ICCD), 2015 33rd IEEE International Conference on*, pages 208–215. IEEE, 2015.

[82] Ashur Rafiev, Alexei Iliasov, Alexandre Romanovsky, Andrey Mokhov, Fei Xia, and Alex Yakovlev. Studying the Interplay of Concurrency, Performance, Energy and Reliability with ArchOn–An Architecture-Open Resource-Driven Cross-Layer Modelling Framework. In *Application of Concurrency to System Design (ACSD), 2014 14th International Conference on*, pages 122–131. IEEE, 2014.

[83] Ashur Rafiev, F Xia, A Iliasov, R Gensh, A Aalsaud, A Romanovsky, and A Yakovlev. *Power-proportional modelling fidelity*. Computing Science, Newcastle University, 2015.

[84] Ashur Rafiev, F Xia, A Iliasov, R Gensh, A Aalsaud, A Romanovsky, and A Yakovlev. *Power-proportional modelling fidelity*. Computing Science, Newcastle University, 2015.

[85] Ashur Rafiev, Andrey Mokhov, Fei Xia, Alexei Iliasov, Rem Gensh, Ali Aalsaud, Alexander Romanovsky, and Alex Yakovlev. Resource-driven modelling for managing model fidelity. In *Model-Implementation Fidelity in Cyber Physical System Design*, pages 25–55. Springer, 2017.

[86] Basireddy Karunakar Reddy, Amit Kumar Singh, Dwaipayan Biswas, Geoff V Merrett, and Bashir M Al-Hashimi. Inter-cluster Thread-to-core Mapping and DVFS on Heterogeneous Multicores. *IEEE Transactions on Multi-Scale Computing Systems*, 2017.

[87] Basireddy Karunakar Reddy, Geoff V Merrett, Bashir M Al-Hashimi, and Amit Kumar Singh. Online concurrent workload classification for multi-core energy management. In *Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 621–624, 2018.

[88] Phil Rogers and AC Fellow. Heterogeneous system architecture overview. In *Hot Chips*, volume 25, 2013.

[89] Hiroshi Sasaki, Satoshi Imamura, and Koji Inoue. Coordinated power-performance optimization in manycores. In *Proceedings of the 22nd international conference on Parallel architectures and compilation techniques*, pages 51–62. IEEE Press, 2013.

[90] Marcus Schmitz, Bashir Al-Hashimi, and Petru Eles. Energy-efficient mapping and scheduling for DVS enabled distributed embedded systems. In *Proceedings of the conference on Design, automation and test in Europe*, page 514. IEEE Computer Society, 2002.

[91] Semiconductor Industry Association. ITRS: International Technology Roadmap for Semiconductors . http://www.itrs.net/reports.html [online], 2005.

[92] Semiconductor Industry Association. ITRS: International Technology Roadmap for Semiconductors . http://www.itrs.net/reports.html [online], 2006.

[93] Rishad A Shafik, Anup Das, Sheng Yang, Geoff Merrett, and Bashir M Al-Hashimi. Adaptive energy minimization of openmp parallel applications on many-core systems. In *Proceedings of the 6th Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures*, pages 19–24. ACM, 2015.

[94] Rishad Ahmed Shafik. Investigation into low power and reliable system-on-chip design. 2010.

[95] Rishad Ahmed Shafik, Bashir M Al-Hashimi, Sandip Kundu, and Alireza Ejlali. Soft error-aware voltage scaling technique for power minimization in application-specific multiprocessor system-on-chip. *Journal of Low Power Electronics*, 5(2):145–156, 2009.

[96] Rishad Ahmed Shafik, Anup K Das, Luis Alfonso Maeda-Nunez, Sheng Yang, Geoff V Merrett, and Bashir Al-Hashimi. Learning transfer-based adaptive energy minimization in embedded systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(6):877–890, 2016.

[97] Hao Shen, Jun Lu, and Qinru Qiu. Learning based DVFS for simultaneous temperature, performance and energy management. In *Quality Electronic Design (ISQED), 2012 13th International Symposium on*, pages 747–754. IEEE, 2012.

[98] Hao Shen, Ying Tan, Jun Lu, Qing Wu, and Qinru Qiu. Achieving autonomous power management using reinforcement learning. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 18(2):24, 2013.

[99] Simon McIntosh-Smith. Trends in heterogeneous systems architectures. http://www.people.cs.bris.ac.uk [online], 2013.

[100] Amit Kumar Singh, Charles Leech, Basireddy Karunakar Reddy, Bashir M Al-Hashimi, and Geoff V Merrett. Learning-based run-time power and energy management of multi/many-core systems: current and future trends. *Journal of Low Power Electronics*, 13(3):310–325, 2017.

[101] Sam Skalicky, Sonia Lopez, Marcin Lukowiak, and Andrew G Schmidt. A parallelizing matlab compiler framework and run time for heterogeneous systems. In *High Performance Computing and Communications (HPCC), 2015 IEEE 7th International Symposium on Cyberspace Safety and Security (CSS), 2015 IEEE 12th International Conferen on Embedded Software and Systems (ICESS), 2015 IEEE 17th International Conference on*, pages 232–237. IEEE, 2015.

[102] E Del Sozzo, Gianluca C Durelli, EMG Trainiti, Antonio Miele, Marco D Santambrogio, and Cristiana Bolchini. Workload-aware power optimization strategy for asymmetric multiprocessors. In *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*, pages 531–534. EDA Consortium, 2016.

[103] Srinath Sridharan, Gagan Gupta, and Gurindar S Sohi. Adaptive, efficient, parallel execution of parallel programs. *ACM SIGPLAN Notices*, 49(6):169–180, 2014.

[104] Stefan Steinke, Nils Grunwald, Lars Wehmeyer, Rajeshwari Banakar, Mahesh Balakrishnan, and Peter Marwedel. Reducing energy consumption by dynamic copying of instructions onto onchip memory. In *System Synthesis, 2002. 15th International Symposium on*, pages 213–218. IEEE, 2002.

[105] Richard S Sutton and Andrew G Barto. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[106] Ibrahim Takouna, Wesam Dawoud, and Christoph Meinel. Accurate mutlicore processor power models for power-aware resource management. In *Dependable, Autonomic and Secure Computing*

(DASC), 2011 IEEE Ninth International Conference on, pages 419–426. IEEE, 2011.

[107] Ying Tan, Wei Liu, and Qinru Qiu. Adaptive power management using reinforcement learning. In *Proceedings of the 2009 International Conference on Computer-Aided Design*, pages 461–467. ACM, 2009.

[108] Gerald Tesauro. Online resource allocation using decompositional reinforcement learning. In *AAAI*, volume 5, pages 886–891, 2005.

[109] Constantin Timm, Andrej Gelenberg, F Weichert, and P Marwedel. Reducing the energy consumption of embedded systems by integrating general purpose GPUs. *TU, Dep. of Computer Science*, 2010.

[110] A Torrey, J Cleman, and P Miller. Comparing interactive scheduling in Linux. *Software-Practices & Experience*, 34(4):347–364, 2007.

[111] Matthew Travers, Rishad Shafik, and Fei Xia. Power-normalized performance optimization of concurrent many-core applications. In *2016 16th International Conference on Application of Concurrency to System Design (ACSD)*, pages 94–103. IEEE, 2016.

[112] Kunio Uchiyama. Power-efficient heteregoneous parallelism for digital convergence. In *VLSI Circuits, 2008 IEEE Symposium on*, pages 6–9. IEEE, 2008.

[113] Kenzo Van Craeynest, Aamer Jaleel, Lieven Eeckhout, Paolo Narvaez, and Joel Emer. Scheduling heterogeneous multicores through performance impact estimation (PIE). In *ACM SIGARCH Computer Architecture News*, volume 40, pages 213–224. IEEE Computer Society, 2012.

[114] Matthew J Walker, Stephan Diestelhorst, Andreas Hansson, Anup K Das, Sheng Yang, Bashir M Al-Hashimi, and Geoff V Merrett. Accurate and stable run-time power modeling for mobile and embedded CPUs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36(1):106–119, 2017.

[115] Alice Wang and Anantha Chandrakasan. A 180-mv subthreshold fft processor using a minimum energy design methodology. *IEEE JSSC*, 40(1):310–319, 2005.

[116] Huaide Wang, Meng-Hsiung Hung, Yu-Ching Yeh, and Jri Lee. A 60-GHz FSK transceiver with automatically-calibrated demodulator in 90-nm CMOS. In *VLSI Circuits (VLSIC), 2010 IEEE Symposium on*, pages 95–96, June 2010. doi: 10.1109/VLSIC.2010.5560338.

[117] Yanzhi Wang and Massoud Pedram. Model-free reinforcement learning and bayesian classification in system-level power management. *IEEE Transactions on Computers*, 65(12):3713–3726, 2016.

[118] Yanzhi Wang, Qing Xie, Ahmed Ammari, and Massoud Pedram. Deriving a near-optimal power management policy using model-free reinforcement learning and bayesian classification. In *Proceedings of the 48th Design Automation Conference*, pages 41–46. ACM, 2011.

[119] Yanzhi Wang, Maryam Triki, Xue Lin, Ahmed C Ammari, and Massoud Pedram. Hierarchical dynamic power management using model-free reinforcement learning. In *Quality Electronic Design (ISQED), 2013 14th International Symposium on*, pages 170–177. IEEE, 2013.

[120] Yi-Chu Wang and Kwang-Ting Cheng. Energy and performance characterization of mobile heterogeneous computing. In *Signal Processing Systems (SiPS), 2012 IEEE Workshop on*, pages 312–317. IEEE, 2012.

[121] Vincent M Weaver. Linux perf_event features and overhead. In *The 2nd International Workshop on Performance Analysis of Workload Optimized Systems, FastPath*, volume 13, 2013.

[122] Yuan Wen, Zheng Wang, and Michael FP O'boyle. Smart multi-task scheduling for OpenCL programs on CPU/GPU heterogeneous platforms. In *High Performance Computing (HiPC), 2014 21st International Conference on*, pages 1–10. IEEE, 2014.

[123] Henry Wong and Tor M Aamodt. The performance potential for single application heterogeneous systems. In *8th Workshop on Duplicating, Deconstructing, and Debunking*, 2009.

[124] Yun Wu, Dimitrios S Nikolopoulos, and Roger Woods. Runtime support for adaptive power capping on heterogeneous socs. In *Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS), 2016 International Conference on*, pages 71–78. IEEE, 2016.

[125] Fei Xia, Ashur Rafiev, Ali Aalsaud, Mohammed Al-Hayanni, James Davis, Joshua Levine, Andrey Mokhov, Alexander Romanovsky, Rishad Shafik, and Alex Yakovlev. Voltage, throughput, power, reliability, and multicore scaling. *Computer*, 50(8): 34–45, 2017.

[126] Changjiu Xian, Yung-Hsiang Lu, and Zhiyuan Li. Energy-aware scheduling for real-time multiprocessor systems with uncertain task execution time. In *Design Automation Conference, 2007. DAC'07. 44th ACM/IEEE*, pages 664–669. IEEE, 2007.

[127] Zichen Xu, Yi-Cheng Tu, and Xiaorui Wang. Exploring power-performance tradeoffs in database systems. In *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*, pages 485–496. IEEE, 2010.

[128] Sheng Yang, Rishad A Shafik, Geoff V Merrett, Edward Stott, Joshua M Levine, James Davis, and Bashir M Al-Hashimi. Adaptive energy minimization of embedded heterogeneous systems using regression-based learning. In *Power and Timing Modeling, Optimization and Simulation (PATMOS), 2015 25th International Workshop on*, pages 103–110. IEEE, 2015.

[129] Rong Ye and Qiang Xu. Learning-based power management for multicore processors via idle period manipulation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(7):1043–1055, 2014.

[130] Lin Yuan and Gang Qu. Analysis of energy reduction on dynamic voltage scaling-enabled systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(12): 1827–1837, 2005.

[131] Yuhao Zhu and Vijay Janapa Reddi. High-performance and energy-efficient mobile web browsing on big/little systems. In *High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on*, pages 13–24. IEEE, 2013.

[132] Sergey Zhuravlev, Juan Carlos Saez, Sergey Blagodurov, Alexandra Fedorova, and Manuel Prieto. Survey of scheduling techniques for addressing shared resources in multicore processors. *ACM Computing Surveys (CSUR)*, 45(1):4, 2012.