INVESTIGATION INTO SCALABLE ENERGY AND PERFORMANCE MODELS FOR MANY-CORE SYSTEMS

Mohammed A. Noaman Al-hayanni

A Thesis Submitted for the Degree of

Doctor of Philosophy at Newcastle University

School of Engineering

Faculty of Science, Agriculture and Engineering

May 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 - May 2019

Mohammed A. Noaman
Al-hayanni

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 at the School of Engineering / Newcastle University for the degree of PhD in Electrical and Electronic Engineering / Computer Engineering.

Alex Yakovlev

Rishad Shafik

Fei Xia

To my lovely family.
— Mohammed Al-hayanni

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my supervisors Prof. Alex Yakovlev, Dr. Rishad Shafik and Dr. Fei Xia for their support and guidance through my Ph.D. journey. They have always been a source of motivation and my inspirational model as a researcher.

I am grateful to my sponsor Higher Committee of Education Development (HCED) in Iraq for funding my Ph.D. study through their scholarship program. Furthermore, I am thankful to the University of Technology-Iraq for giving the opportunity to join the scholarship and also would like to express my sincere gratitude to the Iraqi Cultural Attache in London.

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 synthetic benchmark programs.

I am also grateful to Dr. Waleed Fawwaz for his advice and suggestions for preparing some Matlab codes.

I would also like to express my gratefulness and appreciation to my colleagues and friends in the School of Engineering, particularly those in μSystems Research Group at Newcastle University for their assistance through my study.

I would like to offer my special regards to all the staff of the Electrical and Electronic Engineering in the School of Engineering at 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.

Recite in the name of your Lord who created - Created man from a clinging substance. Recite, and your Lord is the most Generous - Who taught by the pen -Taught man that which he knew not.

The holy Quran - Chapter 30 - Verse 96 - (1-5).

ABSTRACT

It is likely that many-core processor systems will continue to penetrate emerging embedded and high-performance applications. Scalable energy and performance models are two critical aspects that provide insights into the conflicting trade-offs between them with growing hardware and software complexity. Traditional performance models, such as Amdahl's Law, Gustafson's and Sun-Ni's, have helped the research community and industry to better understand the system performance bounds with given processing resources, which is otherwise known as speedup. However, these models and their existing extensions have limited applicability for energy and/or performance-driven system optimization in practical systems. For instance, these are typically based on software characteristics, assuming ideal and homogeneous hardware platforms or limited forms of processor heterogeneity. In addition, the measurement of speedup and parallelization factors of an application running on a specific hardware platform require instrumenting the original software codes. Indeed, practical speedup and parallelizability models of application workloads running on modern heterogeneous hardware are critical for energy and performance models, as they can be used to inform design and control decisions with an aim to improve system throughput and energy efficiency.

This thesis addresses the limitations by firstly developing novel and scalable speedup and energy consumption models based on a more general representation of heterogeneity, referred to as the normal form heterogeneity. A method is developed whereby standard performance counters found in modern many-core platforms can be used to derive speedup, and therefore the parallelizability of the software, without instrumenting applications. This extends the usability of the new models to scenarios where the parallelizability of software is unknown, leading to potentially Run-Time Management (RTM) speedup and/or energy efficiency optimization. The models and optimization methods presented in this thesis are validated through extensive experimentation, by running a number of different applications in wide-ranging concurrency scenarios on a number of different homogeneous and heterogeneous Multi/Many Core Processor (M/MCP) systems. These include homogeneous and heterogeneous architectures and

range from existing off-the-shelf platforms to potential future system extensions. The practical use of these models and methods is demonstrated through real examples such as studying the effectiveness of the system load balancer.

The models and methodologies proposed in this thesis provide guidance to a new opportunities for improving the energy efficiency of M/MCP systems.

STATEMENT OF ORIGINALITY

The contributions in this thesis have supported by diverse publications which include a large amount of work during the journey of my study. They are published in different papers including journal, magazine, conferences and technical reports. The materials are covered in separate chapters. The publications which include the ***contributions*** of this thesis are listed as follows:

Journals and Magazines Publications:

1. Ashur Rafiev; **Mohammed A. N. Al-hayanni**; Fei Xia; Rishad Shafik; Alexander Romanovsky; Alex Yakovlev, *Speedup and Power Scaling Models for Heterogeneous Many-Core Systems*, Transactions on Multi-Scale Computing Systems (TMSCS), 12 January 2018, Page(s): 436 - 449, DOI: 10.1109/TMSCS.2018.2791531, Publisher: IEEE, Electronic ISSN: 2332-7766.

   The underpinning studies of this paper can be found in Chapter 3 and Chapter 4. It includes the essential ***contributions*** of this literary work which represented by extending the assumption of system heterogeneity to cover such modern Multi/Many-Core Processors (M/MCP) configurations. It presents the attempt to extend the classical speedup models (Amdahl, Gustafson, and Sun-Ni) to estimate power and energy normalized speedup metrics. Thus, it studies the power/performance trade-offs of the extended models for energy-efficient computing in diverse M/MCP configurations. Further, This literature considers the effect of Dynamic Voltage Frequency Scaling (DVFS) techniques on all power and energy models. Moreover, it clarifies the limitations of the Amdahl-like heterogeneous models and outlining further challenges of heterogeneous speedup and power modeling.

   On the other hand, this paper performs intensive validations experiments. It validates the extended models on real heterogeneous platforms on the Odroid XU3 platform and dual-GPU laptop under a set of carefully controlled model parameters. Furthermore, it uses these

models to evaluate the efficiency of the Linux scheduler's load balancing while running realistic workloads.

In addition, the synthetic benchmark used in this paper has been included in Appendix A, the source code for Open Computing Language (OpenCL) version is also available in [1], and the full data set have included in Appendix B.

Conference Publications:

1. **Mohammed A. N. Al-hayanni**; Ashur Rafiev; Rishad Shafik; Fei Xia , *Power and Energy Normalized Speedup Models for Heterogeneous Many-Core Computing*, 16th International Conference on Application of Concurrency to System Design (ACSD), 19-24 June 2016, pp 84 - 93, DOI: 10.1109/ACSD.2016.16, Publisher: IEEE.

   The underpinning studies of this paper can be found in Chapters 3, 4 and 5. This paper demonstrates the first attempt to extend the classical speedup models (Amdahl, Gustafson, and Sun-Ni) for energy efficient computing. It demonstrates the theoretical calculations for all extended speedup models. In addition, it explains the first set of experimental work.

2. **Mohammed A. N. Al-hayanni**; Rishad Shafik; Ashur Rafiev; Fei Xia; Alex Yakovlev, *Speedup and Parallelization Models for Energy-Efficient Many-Core Systems Using Performance Counters*, International Conference on High-Performance Computing and Simulation (HPCS), 17-21 July 2017, PP: 410 - 417, DOI: 10.1109/HPCS.2017.68, Publisher: IEEE.

   The underpinning studies of this paper can be found in Chapter 5. It extends the Amdahl's speedup model considering applications and system software related overhead separately. Moreover, It proposes a novel method to model the parallelization ratio for executed applications. This paper performs intensive validations experiments to validate the proposed models. Furthermore, it demonstrates the effectiveness of our method for identifying parallelization-aware energy-efficient system configurations using power/energy metrics.

Technical Reports and Memos:

1. **Mohammed A. N. Al-hayanni**; Ashur Rafiev; Rishad Shafik; Fei Xia; Alex Yakovlev, *Extended Power and Energy Normalized Performance Models for Many-Core Systems*, Technical Report Series, NCL-EEE-MICRO-TR-2016-198, Year: 2016, Newcastle University, μSystems Research Group, School of Engineering. Available at http://async.org.uk/tech-reports/NCL-EEE-MICRO-TR-2016-198.pdf.

   The underpinning studies of this paper can be found in Chapters 3, 4 and 5. It attempts to extend the assumption of heterogeneity for classical speedup models. It includes the theoretical modeling and calculations of all performance and power models. Also, it contains the first set of experimental works.

2. **Mohammed A. N. Al-hayanni**; Rishad Shafik; Ashur Rafiev; Fei Xia; Alex Yakovlev, *Speedup and Parallelization Models for Energy-Efficient Many-Core Systems Using Performance Counters*, Technical Report Series, NCL-EEE-MICRO-TR-2017-205, Year: 2017, Newcastle University, μSystems Research Group, School of Engineering UK. Available at http://async.org.uk/tech-reports/NCL-EEE-MICRO-TR-2017-205.pdf.

   The underpinning studies of this paper can be found in Chapter 5. It attempts to extend the Amdahl speedup model and system software separately. It uses hardware performance counter to avoid the need for instrumenting applications. In addition, This paper proposes a new method to model the parallelization ratio for different applications. Furthermore, it performs extensive synthetic and real benchmarks experiments to validate all models. The full data set of performance and power of the Princeton Application Repository for Shared-Memory Computers (PARSEC) benchmarks of this paper included in AppendixC

Other Contributions:

I also contributed in the following work:

1. Fei Xia; Ashur Rafiev; Ali Aalsaud; **Mohammed A. N. Al-hayanni**; James Davis; Joshua Levine; Andrey Mokhov; Alexander Romanovsky;

Rishad Shafik; Alex Yakovlev; Sheng Yang, *Voltage, Throughput, Power, Reliability, and Multicore Scaling*, Journal : Computer, Year: 2017, Volume : 50, Issue: 8, pp: 34 - 45, DOI: 10.1109/MC.2017.3001246, ISSN: 0018-9162.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

ALU Arithmetic Logic Unit

AMCP Asymmetric Multi-Core Processor

BCE Base Core Equivalent

BCEs Base Core Equivalents

CMOS Complementary Metal-Oxide-Semiconductor

CPU Central Processing Unit

CPU – GPU Central Processing Unit-Graphics Processing Unit

CUDA Compute Unified Device Architecture

DMCP Dynamic Multi-Core Processor

DVFS Dynamic Voltage Frequency Scaling

EDP Energy-Delay Product

ENP Energy Normalized Performance

EPI Energy per Instruction

FPGA Field-Programmable Gate Array

GPU Graphics Processing Unit

HeMCP Heterogeneous Multi-Core Processor

HoMCP Homogeneous Multi-Core Processor

HPC High Performance Computing

IPC Instructions per Clock

IPS Instructions per Second

ISA Instruction Set Architecture

ISSCC International Solid-State Circuits Conference

ITRS International Technology Roadmap for Semiconductors

M/MCP Multi/Many Core Processor

MIDs Mobile Internet Devices

MSR Model-Specific Register

MOSFET Metal Oxide Semiconductor Field Effect Transistor

NoC Networks-on-Chip

OpenMP Open Multi-Processing

OpenCL Open Computing Language

OS Operating System

PNP Power Normalized Performance

PARSEC Princeton Application Respository for Shared-Memory Computers

POSIX Portable Operating System Interface for Unix

RTM Run-Time Management

SoC System-on-Chip

SIMD Single Instruction, Multiple Data

SMCP Symmetric Multi-Core Processor

TSC Time Stamp Counter

LIST OF SYMBOLS

α	performance factor for the core
α_i	performance factor for the core type i
α_s	performance factor of sequential execution
$\overline{\alpha}$	vector of the core performance factors
β	power factor of core
β_i	power factors of core type i
β_s	power factor of sequential execution
$\overline{\beta}$	vector of core power factors
θ	base core equivalent performance
$\Theta(n)$	homogeneous system performance on number of cores
$\Theta(r)$	relative performance of Hill-Marty assumption
$\mathcal{C}$	number of clock cycle
ΔI	system software instructions
a	activity factor
A_0	product of activity factor by capacitance for idle power consumption
A_n	product of activity factor by capacitance for effective power
C	switch capacitance
$\overline{c}$	set of core allocations
$D_w(\overline{n})$	power distribution characteristic function
$D_w(n)$	power distribution function for homogeneous system
e	effective energy
E	total energy consumption

E_0 idle energy

F clock frequency

$g(\overline{n})$ parallel workload scaling factor

$g(n)$ workload/memory scaling

$h(\overline{n})$ proportional workload scaling factor

I unscaled workload size

I' scaled workload size

I_{Amd} total number of instructions for Amdahl's model

I_{Gus} total number of instructions for Gustafson's model

n number of cores

n_i number of cores of type i

$\overline{n}$ vector of core numbers

N total number of heterogeneous cores

N_α performance-equivalent number of Base Core Equivalents

N_β power-equivalent number of Base Core Equivalents

p parallelization factor

p_{Amd} Amdahl's parallelization factor

p_{Gus} Gustafson's parallelization factor

q quality metric of load balancimg algorithm

s type of core executing sequential workload

$S(\overline{n})$ heterogeneous speedup

$S(n)$ homogeneous speedup

$S_{low}(\overline{n})$ lower speedup limit

$S_{high}(\overline{n})$ higher speedup limit

$t(\overline{n})$ unscaled workload execution time

$t'(\overline{n})$ scaled workload execution time

$t(1)$ workload execution time in a single core

$t(n)$ workload execution time in the number of cores

$t'(n)$ extended workload execution time

$t'_p(\overline{n})$ parallel execution time

$t'_s(\overline{n})$ sequential execution time

$t'_p(n)$ speedup-dependent parallel execution time

$t'_s(n)$ speedup-dependent sequential execution time

V supply voltage

w_0 idle power of a core

w effective power of Base Core Equivalent

w_a active power of a core

W_0 total background power

$W(\overline{n})$ total effective power

$W(n)$ total effective power of homogeneous system

W_{total} total power of the system

w_s sequential effective execution power

w_p parallel effective execution power

W_{Amd} Amdahl's total power consumption

W_{Gus} Gustafson's total power consumption

W_l leakage power

x clusters (types) of homogeneous cores

Part I

Thesis Chapters

1

INTRODUCTION

1.1 MOTIVATION AND CHALLENGES

Micro/nanoelectronic scaling technology has facilitated significant performance improvements with reduced power consumption in microprocessor system design through increased operating frequency and smaller device geometries [5]. Furthermore, the number of transistors per unit area is also increasing substantially, which also conforms to Moore's [6] and Koomey's laws [7]. It is further stated that performance per watt is growing exponentially and the number of electronic components is doubling every 1.5 years [6].

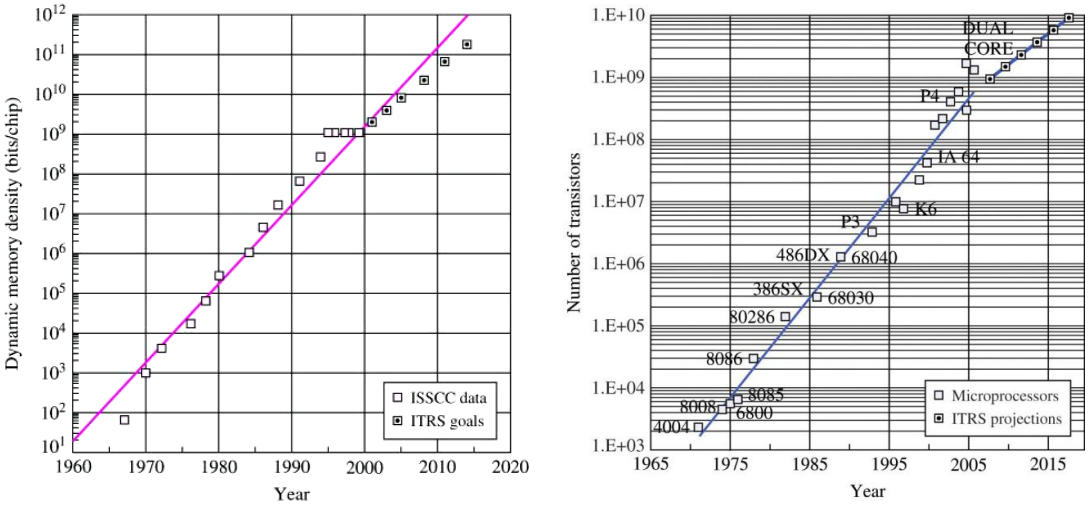


Figure 1.1: Micro/nanoelectronic scaling over time [2]

In order to predict the future of component scalability, some additional studies and research groups have proposed the future roadmaps, such as the International Technology Roadmap for Semiconductors (ITRS) [8] and the International Solid-State Circuits Conference (ISSCC) [9]. Figure 1.1 explains the goals and projections of the ITRS [8] and ISSCC [9] for the growth in

the number of microelectronics components over time for dynamic memory density in Figure 1.1(a) and the number of transistors in chips in Figure 1.1(b).

Over the years, significant research has been carried out to understand the trends of growth in performance in single and many interconnected cores. An example of these models is Pollack's Rule, which suggests that an increase in performance are approximately proportional to the square root of complexity [10]. Following this rule, a doubling of the number of components in a double processor will provide twice the performance in contrast to a single processor [5]. Therefore, several-core systems will deliver further improvements in throughput and latency for the same die area. Many studies have demonstrated the necessity of the evolution from single to Multi/Many Core Processor (M/MCP) to achieve performance improvements in the whole system alongside reducing in power consumption [5].

The calculation of performance is essential to the growth of M/MCP. One of the most appropriate methods to discover the suitable performance in M/MCP is *speedup*. Several recent studies have explained the ideas behind speedup calculations; however, the most familiar classical speedup models are considered by most of these studies. The first scalable and familiar model in relation to M/MCP is explained in Amdahl's law [11]. Amdahl assumed that the speedup model for a fixed workload can be calculated by comparing the performance of the workload executed in a single core with the performance of the same workload executed in the number of cores (n). The model shows the limitations in speedup with increasing n. In 1988, Gustafson introduced the principle of scalable computing in M/MCP pertaining to a fixed time model. Gustafson proposed a linear speedup model where increases in workload are proportional to increasing machine scalability, while execution time remains fixed [12]. In 1990, Sun and Ni suggested a new model which includes extended workload calculations by increasing the capability of memory. It is important to note that the executed workload and time should change based on the capability of the system, whereas the performance calculations appear to be super-linear within the cores increasing [13, 14].

On the other hand, power consumption management is a significant issue which should be considered in calculations of scalable systems and M/MCP. Several techniques have been designed to manage power consumption. For instance, Dynamic Voltage Frequency Scaling (DVFS), clock, and power gating techniques and fine grain power management. [15, 16, 17, 18] are some of the scaling techniques used to decrease power consumption.

Currently, most consumer devices and embedded systems make use of the computational performance and power of M/MCP. The number of cores is growing constantly, and hence speedup models remain very important. Amdahl's law and models derived from it do not require complex modeling and the simulation of individual inter-process communication. Instead, they operate according to average platform and application characteristics and provide simple analytical solutions that project the system's capabilities in a clear and understandable way. However, it is crucial to keep the models up to date to make sure that they remain relevant and correctly represent novel aspects of platform design.

The classical speedup models of Amdahl, Gustafson, and Sun-Ni have been investigated in several studies to gain an understanding of M/MCP modeling. Hill and Marty extended Amdahl's model to simple homogeneous and heterogeneous configurations [3]. Other studies [19, 20] have extended all three major speedup models following the principle of the Hill-Marty models. Meanwhile, researchers in [21, 22] have extended Hill-Marty's models to consider the problem of energy efficiency in simple homogeneous and heterogeneous configurations.

The concept of heterogeneity has emerged within the study of the increase in system complexity and integration. Basically, this phenomenon appears in specialized accelerator forms such as Graphics Processing Unit (GPU). Recently, several types of Central Processing Unit (CPU) cores integrated into a single processor have also been made popular, such as the ARM big.LITTLE processor which has found wide use in mobile devices [23]. Heterogeneous systems have added more research and engineering challenges. From the perspective of the trade-offs between performance and power, the goal is to achieve better performance with the available power. From the perspective of load balancing and scheduling, the goal is to improve the core utilization to more efficiently use the performance available.

The Hill-Marty models and related studies have considered simple heterogeneous configurations consisting of a single big core and many smaller ones of exactly the same type [3, 19, 20], which relates to the Central Processing Unit-Graphics Processing Unit (CPU – GPU) type of heterogeneity. In addition, the problem of energy efficiency has been addressed in [21] for simple homogeneous and heterogeneous Hill-Marty models.

Parallel programming and its effects on power/energy management techniques are one of the vital issues in M/MCP. It may be facing some obstacles within the developments in hardware and software complexity [24].

Parallel programming is represented in many recent studies by workload parallelism in a different level of software design [25, 26, 26]. The classical and related speedup models [11, 12, 13, 14] consider workload parallelism by a fixed controlled parameter called parallelization factor (p). On the other hand, the parallelization factor has also been modeled based on the average of a program's parallelism and its variance in parallelism [27]. In addition, the implications of the parallelization factor modeling according to Amdahl's law in M/MCP has been considered [28] describing the level of parallelism achieved in different parts of the calculation. The effects of p on calculations for performance, power and energy metrics has described by [21] for a simple different architecture of M/MCP. This leads to focus more on the study of parallelization in Run-Time Management (RTM)[29, 30].

1.2 AIM AND OBJECTIVES

This thesis describes research which seeks to extend existing classical speedup models in the context of contemporary M/MCP architectures, which are not covered by existing models. In addition, it includes power and energy models in the same context. The investigations extend to making these models more practically applicable for such goals as the RTM optimization of M/MCP systems. Experimental investigations using real off the shelf systems support the research at all points. The objectives of this thesis can be summarized as follows:

1. This thesis seeks to extend the classical theoretical M/MCP speedup models to cover modern system homogeneity and heterogeneity. The models can capture the interplay between energy and performance. Furthermore, these models are expected to be used for analyzing workload balancing methods.

2. This thesis sets out to propose a novel method to model p based on the extended speedup models. It should explain the effect of different application parallelizability on energy efficient computation for M/MCP.

3. This thesis seeks to validate all models by extensive experiments on different homogeneous and heterogeneous M/MCP by using designed synthetic and real Princeton Application Respository for Shared-Memory Computers (PARSEC) benchmarks.

4. This thesis seeks to establish M/MCP power , energy, and other efficiency models considering full-domain DVFS and per-core DVFS.

1.3 THESIS ORGANIZATION AND KEY FINDINGS

This thesis is organized into six chapter. The major contributions of this thesis are described in 2 parts. The extended speedup, power models and the validations experiments of M/MCP are covered in Chapter 3 and Chapter 4. The p modeling, full-domain DVFS and per-core DVFS power modeling with all their experimental work are coverd in Chapter 5. The chapters are summarized as follows:

Chapter 1 "*Introduction*". Introduces the motivations and challenges, aim and objectives, and the layout of the thesis.

Chapter 2 "*Background and Literature Review*". Provides a background literature survey establishing the environment and baseline of this research. The survey takes an in-depth look at the trend of integration of digital electronics, the development and significance of parallelization in M/MCP systems, and the modeling of speedup, power and energy in parallel processing systems. The chapter summarizes existing related work to illustrate the gaps this research sets out to fill and gaps for potential future work to fill.

Chapter 3 "*Speedup and Power Scaling Models*". Extends the assumption of system core heterogeneity in order to cover modern configurations such as big.LITTLE. The assumption establishes by extending the classical speedup models (Amdahl, Gustafson, and Sun-Ni) [11, 12, 13] in order to cover modern system heterogeneity. Furthermore, It extend the speedup models to estimate power and energy normalized speedup metrics. It incorporates the representations of the power and energy optimization effects techniques such as DVFS in the extended power models. In addition, it incorporates the calculations of workload balancing methods within extended performance and power modeling.

Chapter 4 "*Experimental Validation of Speedup and Power Scaling Models*". Validates the extended models from Chapter 3 on real heterogeneous

platforms including big.LITTLE and CPU – GPU through extensive experimentation. Moreover, this chapter uses these models to evaluate the efficiency of Linux scheduler's load balancing while running realistic workloads in a heterogeneous system.

Chapter 5 "*Speedup and Parallelization Models Using Performance Counters*". Presents a new method of p modeling from the extended speedup models. Furthermore, extensive experiments and analysis of synthetic and real PARSEC benchmarks have been applied. In addition, it presents novel full-domain DVFS and per-core DVFS power models for M/MCP.

Chapter 6 "*Conclusions and Future Work*". Summarizes the work and contributions of this thesis and identifies promising directions for future work.

2

BACKGROUND AND LITERATURE REVIEW

2.1 INTRODUCTION

There is a growing body of literature that recognizes the importance of micro/nano electronic in computer system development. The principles of electronic scaling permit to double the electronic components in the same die area every two years [6, 7]. Furthermore, the trend toward processors fabrication from single core to Multi/Many Core Processor (M/MCP) became necessary to improving performance and produce energy-efficient processors [5]. In general, M/MCP may be configured into Homogeneous Multi-Core Processor (HoMCP), where all the cores have a similar architecture such as Intel core i7-4820k and Intel Xeon Phi 7120X, or Heterogeneous Multi-Core Processor (HeMCP), where the cores have different architectures such as ARM big.LITTLE [31, 32, 3]. The HeMCP may incorporate diverse architectures of processing units such as, Central Processing Unit (CPU)s, Graphics Processing Unit (GPU) and embedded Field-Programmable Gate Array (FPGA) [33, 34].

Along the exploitation of micro/nano electronics component scaling to produce high performance and energy efficient M/MCP, additional techniques have been developed for dealing with same issues. For instance, Dynamic Voltage Frequency Scaling (DVFS) is a technique has developed to manage power consumption within processor clusters or individual cores [35, 36], the CPU affinity allows a task to core mapping, which may be used to reduce energy consumption. Other techniques targeting the same concerns include the energy efficient load balancing, task migration, and scheduling have invented for the same issues [37, 38, 39, 40].

This chapter reviews the literature on M/MCP, particularly those related to speedup models such as Amdahl [11], Gustafson [12] and Sun-Ni [13, 14]. It studies the background of the speedup models. Extended speedup models for M/MCP for improving performance and energy will also be discussed. The survey will also deal with theoretical models and practical validation methods found in the literature. In addition, it discusses the overheads that can affect performance and power consumption, explain the parallelization calculation,

and consider dark silicon calculation in extended speedup models. Finally, it summarises the discussed studies by the Table 2.1. The ***contributions*** of this chapter can be addressed as follows:

1. Provides basic concepts and theory of micro/nano electronic scalability, the trend from single to M/MCP, M/MCP architecture, the methods of performance improvements and power/energy consumption reduction in M/MCP (Section 2.2 to 2.5).
2. Gives a brief survey including the concepts of classical speedup models Amdahl, Gustafson, and Sun-Ni and related speedup models such as, (Downey), the extended speedup models in M/MCP such as Hill-Marty (Section 2.6).
3. Provides an extensive survey of the overhead calculations in speedup models, the parallelization factor, the useful utilization of speedup models in networking, Run-Time Management (RTM), dark silicon, and power/energy normalized extended models (Section 2.6).
4. Summarizes all literature in this chapter in Table 2.1 to clarify the contributions of each study.

2.2 MICRO/NANO ELECTRONIC TECHNOLOGY SCALING

The evolution of electronics components has been continuing since the transistor was invented. Technology scaling, traditionally a good method of improving energy efficiency, has been facing challenges [41]. Dennard presented the Metal Oxide Semiconductor Field Effect Transistor (MOSFET) scaling law. The study considers the design, fabrication, and characterization of very small MOSFET switching devices appropriate for digital integrated circuits [42]. Consequently, The Dennard law states that approximately, if transistors get smaller their power density stays constant. Thus, the power calculations remain in proportion to the die area. Dennard scaling relates to Moore's Law [6] which explains that a reduction in the transistor's size leading to more transistors per chip at the same cost. Furthermore, Moore's law states that the performance per watt is growing exponentially at the same rate. This claim is related to Koomey's law [7] which elucidates that electrical efficiency *performance* has double roughly every 1.5 years. This leads to the efficiency improvements that enables many devices creations, such as mobile smartphone and wireless sensors.

Recently, Dennard scaling starts facing challenges. The study in [43] explains the collapse of Dennard scaling even though Moore's law continued. The essential reason is that at a small size, current leakage presents greater challenges related to power increases and a threat of thermal dissipation. Thus, the energy cost should increase. Based on these challenges, numerous researchers and architects have switched to focusing on developing microprocessors fabrication to M/MCP scaling [5].

The Pollack rule simplifies the explanation of the scaling trends [10]. It demonstrates the changes with each technology generation, frequency increases by 50%, transistor density doubles, and the voltage is lowered. Moreover, the study explains the improvement in manufacturing technology that allowed to increase die size without cost increases. Generally, the die size still limited by power, because of the increase in power dissipation which leads to several new challenges.

In the same field of Complementary Metal-Oxide-Semiconductor (CMOS) scalability, the study in [44] explains the next generations of CMOS scaling by using different data of Intel microprocessors. However, the analysis correspondingly applicable to other types of logic designs that meet the goals forecast by scaling theory introduced in this study [44]. The main consequences of scaling theory analyzed microprocessor performance by increasing the operating frequency and reducing gate delay, double the transistor density, and reduce power consumption.

The scaling aims to find the solutions for all challenges and predict the future of the next generation of technology. For instance, the study in [45] describes the possibility of exascale performance via voltage scaling for logic and memory, managing hierarchical interconnects concurrency and system level resilience. Generally, International Technology Roadmap for Semiconductors (ITRS) provides a set of up-to-date reference documents defining the requirements of the semiconductor technology advancements which takes into consideration the past, present and future related to both industry and academic fields [8].

2.3 FROM SINGLE-CORE TO MULTI/MANY-CORE (M/MCP)

The continuity of microelectronics scaling gives a chance to integrate billions of transistors in a single chip, and this has been expected to double every 18 months [6, 7].

The demand for increased performance in the field of a computer is continuing with every new improvement in processors, which leads to a higher level of requirements from users and businesses. More recently, the performance progress no longer relates to speed improvement solely, it does include smaller portable devices with higher efficiency, long battery life and better price per performance for one watt and lowest cooling costs [46, 23].

Formerly, performance improvements of the processors were achieved by increasing microelectronics technology scaling. Increasing clock speeds and power dissipation. Recently, the clock speeds have stagnated, and the power dissipation remained flat [47]. Thus, the principle of M/MCP has established as one of the new methods to improve processors performance. To explain this principle clearly, consider a logic block with an operating voltage of 1 unit, a frequency of 1, a throughput of 1, and a power of 1. If the designer reduced the voltage to 0.7, then the frequency must also reduce to 0.7 by considering DVFS technique. Thus, the throughput reduces to 0.7, and the power reduces to 0.35. The power can be calculated by 2.1 [48] with a hypothesis that constant C equal to 1, V is voltage, F is operating frequency and P_{Dyn} is the dynamic power.

$$P_{Dyn} = C \cdot V^2 \cdot F. \tag{2.1}$$

If the logic block is then replicated and two blocks operate in parallel. The total power for the two blocks is 0.7, and the throughput is 1.4. This illustrates how parallel processing can increase performance while cutting power consumption.

Continuing the process of doubling the logic in a single processor core is not the unique method to improve processor performance within the same power envelope. This leads to using additional techniques, such as M/MCP and Multi-thread technology. Pollack assumed that a single thread processor would provide a diminishing return in performance versus power [10].

On the other hand, M/MCP has several benefits related to power consumption reduction [5] such as:

1. The cores can be individually turned on or off, thus saving power when a core is not needed.

2. Each processor core may have its own supply voltage and frequency, providing flexibility.

3. Easier distribute heat across the die with core work matching.

4. It can potentially produce lower die temperatures, improving reliability and leakage.

Figure 2.1 demonstrates the simple architecture of M/MCP.

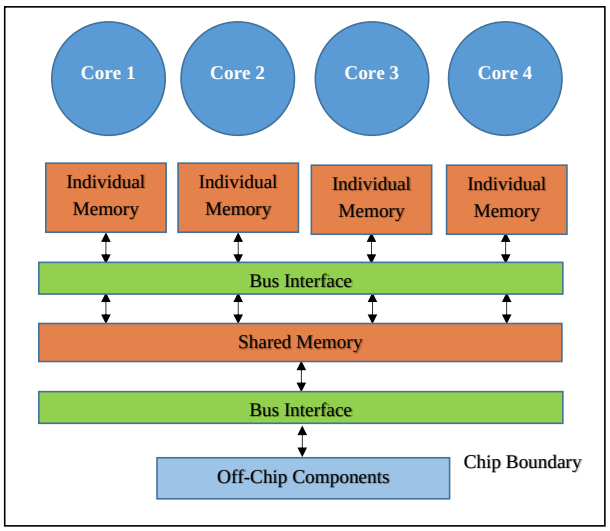


Figure 2.1: M/MCP architecture.

2.4 M/MCP ARCHITECTURE

The continued increase in the number of microelectronic components allowed the development of M/MCP in the number of architectures. In general, it has been proposed to classify the M/MCP architectures into homogeneous, heterogeneous and dynamic as shown in Figure 2.2 [32, 3, 49].

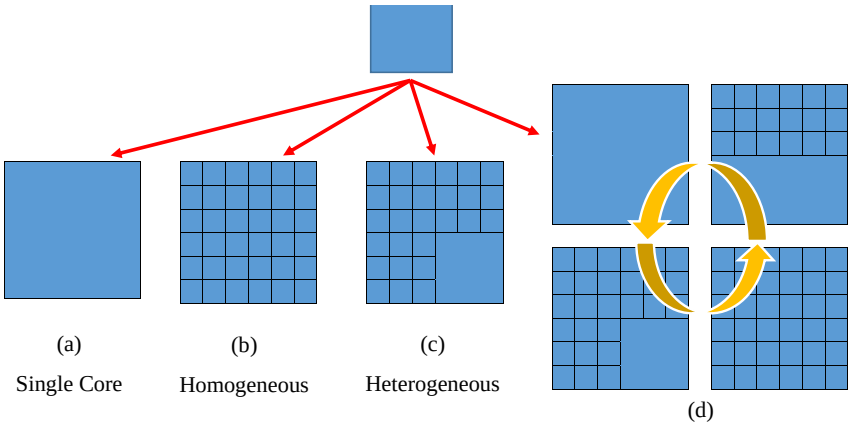


Figure 2.2: Diversity of M/MCP.

2.4.1 *Homogeneous M/MCP*

HoMCP integrates M/MCP that have the same architecture of Symmetric Multi-Core Processor (SMCP) as shown in Figure 2.2 (a) [31, 32, 3]. In this type all the cores have identical performance and Instruction Set Architecture (ISA) [50, 51].

HoMCP has several benefits such as the flexibility to run different processes simultaneously. They may also execute independent threads spawned from a single process to improve the performance of a single application.

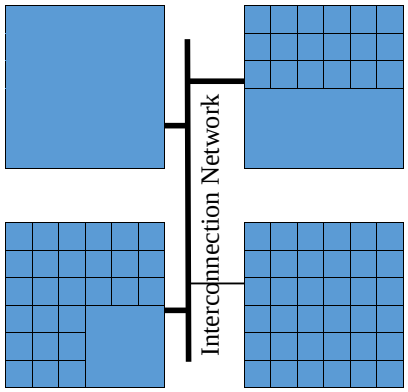
One of the specific types of HoMCP is GPU which are designed as special purpose processors for visual processing [52]. Modern GPUs may incorporate hundreds of cores to achieve parallel processing by handling thousands of threads simultaneously [53]. Generally, GPU is similar to CPU. However, it includes many smaller cores in comparison with multi bigger cores in CPU and has some differences such as CPU may be designed for speedup including (latency/throughput) improvements while GPU tend to be designed generally to improve throughput [54].

2.4.2 *Heterogeneous M/MCP*

HeMCP incorporates the large number of different computational units that may have different architecture. These include full-blown latency oriented cores for sequential processing, massively parallel Single Instruction, Multiple Data (SIMD) accelerators such as GPU, embedded FPGA and media accelerators [33, 34].

The Asymmetric Multi-Core Processor (AMCP) is a special case of heterogeneity when the processor includes two types of different cores) as shown in Figure 2.2 (b) [31, 32, 3]. In this type, not all the cores have the same performance and may have a single ISA [50, 55] or more than one ISA [56].

The HeMCP suffer from challenges brought by the flexibility to run different processes simultaneously. However, one advantage of heterogeneity is the high ability to manage performance/power trade-off to improve system performance and reduce power consumption. For instance, the big.LITTLE technology from ARM is HeMCP incorporating a cluster of 'big' cores for high performance and a cluster of 'LITTLE' cores for low power consumption. Further, some studies interested to study the software model design to manipulate the executed workload in HeMCP [23, 57].



(a)

Distributed Multi-Core Chips

Figure 2.3: Distributed M/MCP.

M/MCP architectures may feature in single chips or form distributed structures with multiple cores connected through communications facilities such as networks [58]. Figure 2.3 shows a simple architecture of distributed M/MCP.

HeMCP could include CPUs or Central Processing Unit-Graphics Processing Unit (CPU – GPU) in single chip as shown in Figure 2.4. The CPU – GPU integration in single chip offer performance imrovements [59, 60]. Further advantages include reduced communication overheads and costs, specially designed shared memory for avoiding explicit data copying [61]. They may also deliver more power and energy efficient computations [62, 59, 60].

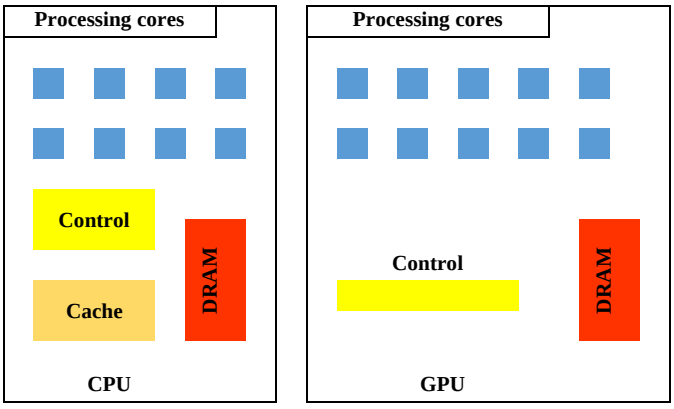


Figure 2.4: Architecture of a CPU – GPU chip.

Recently, it has been claimed in the literature that optimization targeting certain applications have resulted in performance speedup from 25x to 100x or more utilizing GPU instead of CPU [63]. The main reason for this comes

from the differences in the architecture of each unit. CPU and GPU are designed in order to execute different types of applications [64, 65]. These differences cause CPU achieve better performance on latency-sensitive applications which need to respond rapidly to specific events and partially parallel applications [66, 64, 65]. On the other hand, GPU achieves better performance with latency-tolerant applications, and the processor utilization may be high due to multithreading [67, 68], highly parallel applications and independent applications [63]. Thus, the overall performance on CPU and GPU depends on the application characteristics. Moreover, other works have focused on heterogeneous architecture to combine unconventional cores such as custom logic or FPGA in the traditional M/MCP to achieve superior energy efficiency and performance improvements [59, 65]. This new paradigm considers the relationships between a conventional processor and a various set of unconventional cores. It forecasts future architecture from scaling developments predicted by ITRS [8].

2.4.3 *Dynamic M/MCP*

The Dynamic Multi-Core Processor (DMCP) has the ability to configure cores dynamically to adjust their computational resources through execution. Thus, it can provide superior system performance and lowest power consumption. It tackles the problem of performance/power trade-off which faces M/MCP architecture [32]. Figure 2.2(d) shows the simple idea of DMCP architecture, it can dynamically configure the number and the size of cores. For instance, if the executing program is easy to parallelize, the system may be configured into an increasing the number of cores. Whereas, it can combine several cores into one large one to improve the M/MCP performance for the sequential portion of a program. The overall aim of this type of dynamic configuration is to improve system performance.

2.5 THE METHODS OF ENERGY EFFICIENCY

In parallel with the development from single to M/MCP, methods have been developed to improve these issues. This section provides an overview of some important techniques which relate to this study.

2.5.1 *Dynamic Voltage Frequency Scaling (DVFS)*

DVFS is a technique designed to manage power and energy consumption in M/MCP [35]. Essentially, a controller can scale the voltage and frequency up and down in order to save the power. This technique can be found in a wide range of commercial M/MCP from embedded systems and mobile devices to servers [16]. [36, 69, 5] claim that the modern M/MCP have supported per-core DVFS, for instance, Intel and ARM's big.LITTLE. [5] suggests fine grain power management design, particularly with the power management for idle cores. The study considers two frequency operations for each Microprocessor core, with maximum frequency for fast processing and 1/2 of maximum frequency for idle cores. When a core operates at f/2, it uses lower voltage, subsequently consuming only a few of the maximum dynamic power. Furthermore, when there is support for DVFS for each core individually, each core will have specific performance and power consumption calculations [17, 18, 70]. It is possible to control technique by use dynamic sleep transistor and body biasing in conjunction with clock gating in CMOS technology to reduce power for idle cores [15]. Certain systems even provide the capability to scale voltage and frequency up-and-down independently [36]. Recently, M/MCP systems extended mechanisms to control cores and uncore components which include the components out of cores but connected to the core closely, such as Arithmetic Logic Unit (ALU), L1 and L2 cache [71, 72, 73, 74].

2.5.2 *Thread to Core Affinity Managements*

Modern processors lose a significant amount of time and energy when moving data. With the increase of core numbers, the importance of energy expenditure will increase with time. Thread-to-core affinity may also be used as a technique to enhance performance and/or reduce power.

CPU affinity or pinning is a technique that can enable the binding and unbind of tasks (thread or process) to specific cores in M/MCP. As a result, the tasks execute on the particular core or cores rather than any random core [75]. Generally, there are two types of CPU affinity, software, and hardware. The software affinity is used by an Operating System (OS) or specific application's scheduler to keep tasks on a CPU. Hardware itself may also provide affinity capabilities which makes particular threads execute on specific hardware.

The affinity technique has several trends to utilize the tasks in an efficient method. One area where affinity may help improve energy efficiency is in cache memory use. If shared cache between different cores needs to be used, all cores involved must validate the data, which is a costly procedure. Bouncing the same thread between different cores can also cause increased cache misses and increasing demands on cache validation [76].

Further, the performance of multi-thread applications can be improved when they are accessing the same data. It is better to bind them to the same CPU or core in M/MCP. Thus, the threads do not contend over data and increase cache misses [76]. Besides performance, thread-to-core affinity can also improve the performance to power consumption ratio. For instance, High Performance Computing (HPC) employ different thread to core affinity strategies to maximize the performance-power ratio. [75] demonstrates these improvements through extensive experimenting with HPC benchmarks.

The management of real-time and time-sensitive applications can also be improved based on this technique. Usually, the processors spend a significant amount of time, power and energy to move data. The system processes can be bound to a subset of cores in M/MCP, while other applications can be bound to the remaining cores [76]. Moreover, the core affinity relates to scheduling of real-time tasks on M/MCP. Thus, the unscheduled tasks can be pinned to a single core while the scheduled tasks can be pinned to multiple cores [77]. Furthermore, the impact of threads and data mapping mechanism on the communication traffic of Networks-on-Chip (NoC) is an important issue in relation to execution time in M/MCP. Following a mechanism to approach the lowest communication traffic through threads affinity with data results in significant communication traffic reduction and energy saving [78].

Usually, each application has a parallelization ratio, splitting sequential and parallel sections to schedule the parallel code to run on multiple cores can improve performance and decrease power consumption [79].

2.5.3 *Energy Efficient Load Balancing, Task Migration and Task Scheduling Over M/MCP*

Workload task manipulation is a significant and necessary consideration when aiming to improve M/MCP and embedded system performance and energy-efficient computations trough resources optimization, maximize throughput

and minimize response time. The techniques of task manipulation include load balancing, task migration and task scheduling [37, 38, 39, 40].

Load balancing is a technique used to improve the distribution of workload across multiple cores [37, 38, 39]. Diverse types of load balancing algorithms may be employed to improve performance and reduce energy consumption, particularly during task migration [39]. The algorithms should consider the task behavior and M/MCP architecture to achieve system optimization.

Task migration is the transfer technique of partially executed tasks from a core or cluster of cores to another core or cluster of cores in M/MCP [37, 38, 40, 39]. Task migration may be used to migrate a task from heavily loaded cores to lightly loaded or idle cores in M/MCP particularly HeMCP to balance the load across all cores. Thus, the average response time will be reduced to improve performance and decrease energy consumption [37, 40].

Task scheduling is the technique used to assign the start and end times to a set of tasks [80, 40]. The main challenges in real-time systems are how to satisfy better performance and minimize energy consumption [81], particularly in embedded real-time system and mobile devices. Task scheduling can be implemented by task scheduler software or OS process schedulers [80].

2.6 SPEEDUP MODELS

Speedup is a number comparing the performance of using multiple computation units (cores) to solve a particular problem to the performance of solving the same problem with a single unit (core). One of the most important speedup models was presented in 1967 by Amdahl [11]. Amdahl's law pertains to fixed workloads. It compares the latency/performance of a fixed workload executed on a single core with the performance of the same workload executed in a number of cores (n).

It assumes that any specific workload consists of parallelizable and non-parallelizable parts, with a parallelization factor (p) ($0 \leqslant p \leqslant 1$). This means that with no scheduling overhead, a portion p of the entire quantity of computation for the fixed workload is entirely parallelizable, while the remaining (1 - p) portion must be executed entirely sequentially. Figure 2.5(a) shows the simple speedup calculations of Amdahl's speedup model in M/MCP. This is an idealized model that incorporates issues such as communication overheads and data access to a constant non-parallelizable part. Workloads with different p values that follow this model behave

differently on M/MCP platforms. Speedup is below linear when $p < 1$. The smaller p is, the worse the speedup.

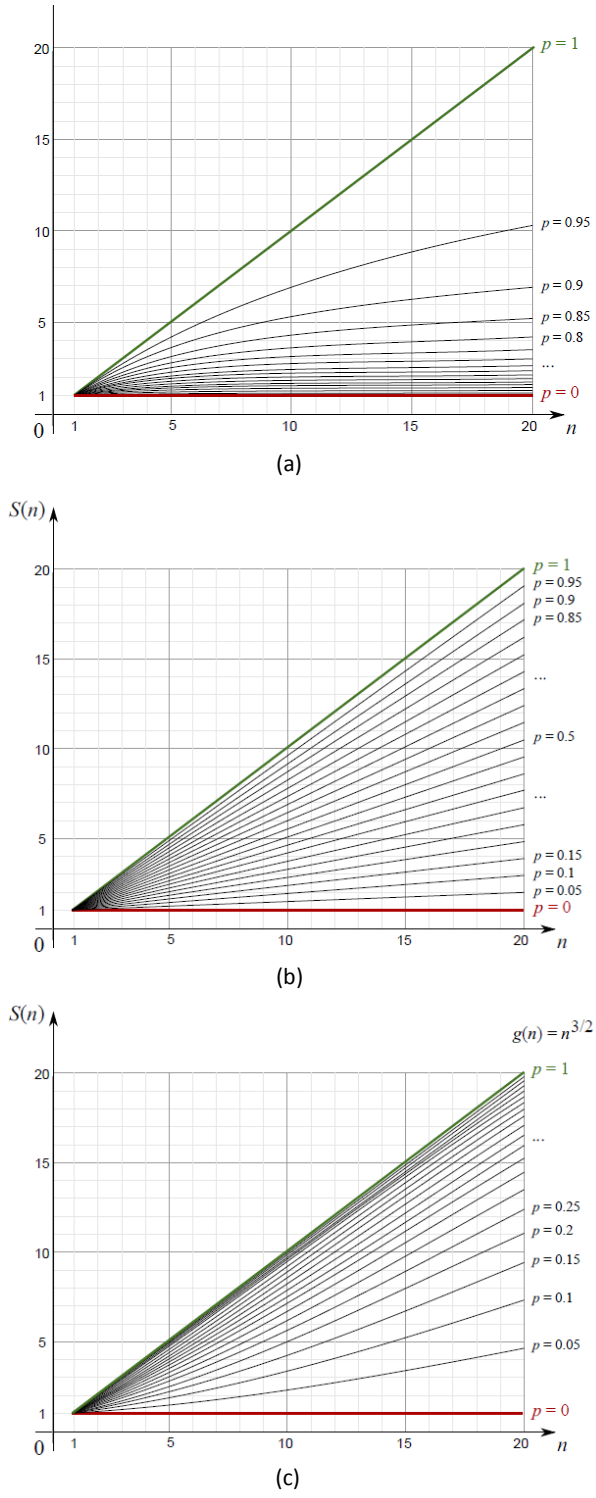


Figure 2.5: S(n) vs n for (a) Amdahl's law (b) Gustafson's model and (c) Sun-Ni's model

In 1988, Gustafson re-evaluated Amdahl's law to introduce the principle of scalable computing in M/MCP related to fixed time workload calculations [12]. It proposes a linear speedup model for workloads that can scale up their parallel part according to the number of computation units available after the sequential part has been completed. As a result, the executed workload during a fixed amount of time grows linearly with the number of available cores. The speedup is calculated according to how much larger the workload is in M/MCP in comparison with the workload in a single core. Workloads that follow this model do not have the non-linear scaling problem which Amdahl's type workloads suffer. Then M/MCP performance improvement is linear even with low p as shown in Figure 2.5 (b).

In 1990, Sun and Ni proposed a superlinear speedup model which extends parallel workload calculation based on the theoretical computation of memory capability [13, 14]. In this model the executed workload and execution time change based on memory capacity. The speedup calculations of this model appear a super linear outcomes with increasing cores. This is an artifact of the memory-bound nature of workloads following this model.

As cores are not the limiting factor of speedup, throughput can scale superlinearly to n in certain situations as shown in Figure 2.5 (c).

There are other types of speedup models, such as Downey's model [27] which are similar to the described models. They calculate speedup using different formulas based on the same principles.

2.6.1 *Extended Speedup Models in M/MCP*

A considerable amount of literature has been published on extending speedup models in the context of reasoning about the performance and energy trade-off in M/MCP. The literature of speedup models may be divided into the number of categories related to the performance improvements calculations, energy-efficient computing, parallelization factor computations, and networking. Further, some others include memory wall and communication overheads calculations.

2.6.1.1 *Extended Speedup Models for Performance Calculations in M/MCP (Hill-Marty Models)*

Hill and Marty attempted to enhance Amdahl's law [3]. They assume that M/MCP consists of units of the same type, called Base Core Equivalents (BCEs),

and multiple BCEs can be reconfigured into larger cores each of which is an integer multiple of BCEs [3], see Figure 2.6.

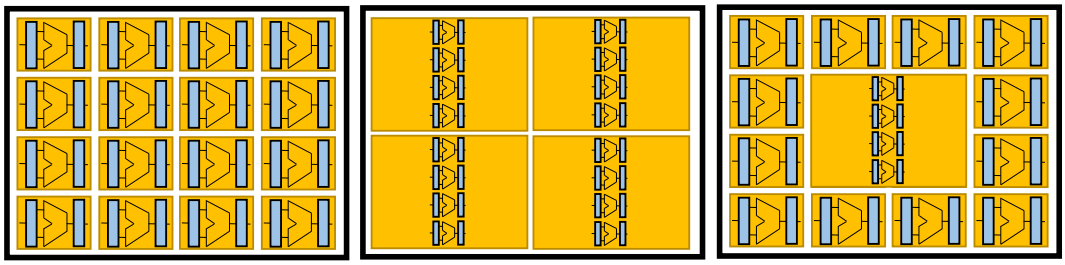


Figure 2.6: M/MCP diversity. (a) SMCP with 16 one-BCE cores, (b) AMCP with 4 four-BCE cores, and (c) AMCP with 1 four-BCE core and 12 one-BCE cores. These figures eliminate important structures such as memory interfaces, shared caches, and interconnects. They assume that area, not power, is a chip's limiting resource. [3]

The chip configuration can consist of n BCEs to behave as HoMCP as shown in Figure 2.6 (a) where (n equal to 16 BCEs). It can also incorporate larger cores of the same size to behave as cores to utilize as HoMCP as shown in Figure 2.6 (b) where (each core includes 4 BCEs, thus the chip configures into 4 large cores. On the other hand, a system can be configured into different-sized cores to form HeMCP as shown in Figure 2.6 (c) where a large single core incorporating 4 BCEs work with 12 small BCEs).

The configurations can cover three M/MCP types which are (SMCP with multi identical big/small multi/many cores, AMCP with single large core and many small cores and DMCP which configure cores dynamically as mentioned in 2.4.3. This was based on their then prediction of what system hardware would become capable of in the future, which has not entirely been proven correct. The Hill-Marty model omits essential issues such as memory wall and communication overheads. Investigations in the literature show that DMCP may hold performance advantages over AMCP which may in turn hold advantages over SMCP. In addition, [82] offers quantitative comprehension of Hill-Marty's study for computer architects in M/MCP scalability. This work also contains additional investigations on finding optimal performance in M/MCP within technology developments according to Moore's law. It suggests a future work direction of focusing on improving the parallelizability of programs and developing more efficient cores to target sequential processing.

Several studies follow the Hill-Marty's principle, [49] extends the performance calculations of Hill-Marty's model to help designers find the optimal performance of Amdahl's law. This study focuses on the theoretical

analysis of M/MCP scalability by adding a performance index into SMCP, AMCP and DMCP configurations. The main principle of the performance index is to improve speedup models by considering the effect of the parallelization factor, the number of the total budget of cores n and the performance of individual configure cores in the total performance. Moreover, [83] addresses the limitations of Amdahl's and Gustafson's speedup models extended by Hill-Marty assumptions by using a (Generalized Scaled Speedup Equation) which incorporates Amdahl's and Gustafson's models via exchanging the appropriate application scaling function. The scaling function is proportional to the executed parallel, unlike constant as in Amdahl's law, but it does not scale linearly as in Gustafson's model either. Further, [20] extends the assumptions of the extended Hill-Marty model for Amdahl's, Gustafson's and Sun-Ni's speedup models in HoMCP and HeMCP. It complements existing studies and provides a clear understanding of M/MCP architecture design.

The studies described in this section do not include other structural issues such as communication overheads or memory wall, these will be covered in section 2.6.1.2.

2.6.1.2 *M/MCP Overheads*

In recent years, there has been an increasing amount of studies on overheads calculation. Overheads are considered to be one of the greatest challenges of performance improvements in M/MCP. They are a combination of extra computation time required by networking, memory bandwidth or other additional parts of processors system that are essential to achieving a specific task. Overheads can be classified into subcategories such as memory wall, communication, and synchronization overheads.

The memory wall characterizes the effects of the memory-processor performance gap on the entire system. The insufficiency in memory latency and bandwidth, for instance, limits the processor's capability of accessing instructions and data. In this case, the processor will stall waiting on memory to continue computation. This issue becomes more complicated in M/MCP and shared memory [84, 85, 86].

The communication overheads describe the effect of communication on the total performance of M/MCP [87, 88, 89].

The synchronization overheads describe the effects on performance of the joining and handshaking of multiple processes and data in M/MCP [87, 90, 89].

The studies in this field may include one or more types of overheads. The work in [91] extend Amdahl's, Gustafson's and Sun-Ni's speedup models to focus on the speedup models from the scalable computing point of view. These models combine a new dimension of scalable computing represented by communication overheads and data access *memory wall* calculations to satisfy better scalability. The calculations of new dimensions are demonstrated via some case studies. This work explains the ability to improve the M/MCP scalability in the approaches of industry and academia. Further, [31] evaluates HoMCP and HeMCP architectures in order to achieve highest performance for a given power budget. This study incorporates synchronization and communication overheads into Amdahl's model to calculate theoretical upper bound performance. The outcomes appear that HeMCP achieves better performance than HoMCP for the same power budget.

Moreover, some studies extend Hill-Marty model to consider the overheads calculations. The studies in [92, 93] extend the Hill-Marty model by considering (Overhead of Data Preparation) of M/MCP. The Overhead of Data Preparation includes memory access, on-chip communications and synchronization among cores. These papers extend theoretical work within three core configurations HoMCP, CPU – GPU HeMCP and CPU – GPU DMCP. They illustrate the importance of reducing Overhead of Data Preparation in M/MCP particularly HoMCP. Moreover, [94] introduces a simple theoretical Hill-Marty model to account for uncore components such as communication overheads. The uncore components on a chip are not considered as cores, they consider other components such as last level caches, on-chip interconnections. The study explains the importance of continuity to design uncore components to scale sub-linearly to sustain the M/MCP scalability. Further, [95] extends Hill-Marty model in diverse topology (HoMCP, HeMCP and DMCP) considering CPU and GPU like organizations to project upper bounds on performance. The speedup models in this study are given as a function of single core performance, micro-architectural features, application parameters, chip organization, and M/MCP topology. They incorporate first order effects exposing more bottlenecks than Amdahl's law.

On the other hand, the study in [19] enhances the principle of Hill-Marty's and Sun-Chen's models [3, 19] in HoMCP. It satisfies the ability of extensive scalability in M/MCP and considers a theoretical study of memory wall in speedup models.

2.6.1.3 *Parallelization Factor (p)*

Certain existing research into speedup has focused on the calculations of average parallelism of a program. The study in [27] proposes a new speedup model calculation for parallel applications based on the average parallelism of a program and its variance in parallelism. It suggests two main theoretical parallelization variance models *low variance model and high variance model* which cover software parallelization diversity. These models have methods to calculate the characteristics of the program. These methods are then applied to speedup curves from real applications in a number of different architectures. The study uses Stanford Parallel Applications for Shared-Memory-2 [96] benchmarks and achieved good speedup model validations. Furthermore, [28] considers the implications of Amdahl's law in a HeMCP in which each processor is a HoMCP. It presents a parallelism function which includes flexible workload distributions, including uneven distributions, onto multiple cores. Some calculations of parallelism function explain the improvements in performance in HeMCP over the performance in HoMCP. The investigations in [27, 28] avoid M/MCP system obstacles such as the effects of synchronization overheads, communication delays and memory coherency, but [28] suggests the ability to include those challenges in the calculations of parallelism function.

2.6.1.4 *Extended Speedup Models in Networks*

The speedup models can be extended to cover distributed CPUs across a network or many cores on a chip across NoC [97]. In this case, the network should be considered as additional parameters to determine the communications overhead. The study in [98] has started to tackle the problem of a real-time system which should ensure the time response requirements in the M/MCP. It extends Amdahl's law by including the communication overheads to analyze the quantitive relationship between speedup and communication based on the ratio of the communication time to the computation time. Thus, it can estimate the lower and upper bounds on speedup calculations. Further, [99] extends the communication overheads for Amdahl's speedup model by considering the effects of communication/computation ratio, network topology, traffic mode, and network size. This study includes experiments using real data of parallel applications on M/MCP NoC platforms.

Moreover, the studies in [100, 58] include the theoretical calculations of communications overheads caused by network limitations in distributed M/MCP. They extend Hill-Marty's [3] and Sun-Chen's models [19] to focus on the impact of communications on latency and throughput. The studies explain the accuracy of new models in comparison to Amdahl's law. The study [101] also considers the communication overheads through additional theoretical calculations into Hill-Marty's [3] and Sun-Chen's models [19]. It models communication time to divide execution into four parts including the sequential and parallel workload of computation and communication. Firstly, the study evaluates the effects of interconnects on overall speedup models. Secondly, it proposes a hardware cost model which includes the area of cores and interconnects. Finally, it applies the hardware cost model to speedup models leading to extended speedup models under area constraints. The results of [101] support the explicit inclusion of inter-core communication in M/MCP speedup models.

2.6.1.5 *Extended Speedup Models in Run-Time Management (RTM) System*

Debate continues about the best strategies for the management of applications in RTM. The speedup models have been extended to cover this area. The performance and power of most applications change during RTM. It is, therefore, necessary to establish a technique to monitor and control important events in any application. Hardware performance counters can serve as monitors for this purpose. [102] examines the use of hardware performance counters and finds that they can be useful in providing power allocation predictions. Furthermore, the approach of application-aware power management has been presented. In [103] the study combines continuous monitoring of critical workload indicators for performance-based speedup usage, online power, and timely state control. This study shows two new power management solutions by using Standard Performance Evaluation Corporation benchmarks on an Intel Pentium M platform during RTM. *Performance maximizer* can find the best performance under specific power constraints and *power saver* can save energy while keeping performance specified requirements. Furthermore, the study in [104] includes the dynamic effects of a wide range of DVFS on the performance-based speedup, power and energy calculations in real time. It proposes an online performance, power, and energy prediction framework that proactively searches the DVFS

space. The study verified its results on modern AMD processors with 152 benchmarks at 5 distinct DVFS states.

[29] proposes a system called Varuna which is able to dynamically, continuously, transparently and rapidly adapt to a program's parallelism. The adaptation matches the instantaneous capabilities of the hardware resources while satisfying different efficiency metrics. Varuna was designed to be applicable for both task-based and multi-threaded programs. It can be inserted into the OS or an application without changing the source code of either.

[105] demonstrates this principle through experiments with multi-thread applications in M/MCP. It develops a framework called *Thread Reinforcer* to dynamically monitor the execution of multi-threaded applications. Thus, It can determine the number of threads that can produce optimal speedup.

The design of an efficient RTM system for M/MCP is presented in [106] which satisfies system requirements by cooperating multi-threaded programs and DVFS in tandem. The proposed system which is called (Performance-Per-Power Optimization) optimizes the system performance under a power constraint. It is based on a heuristic algorithm which uses hardware performance monitoring units aiming for the prediction of power and performance. The algorithm controls the threads to be executed within specific cores at specific operating frequencies to distribute the power budget to each program.

2.6.1.6 *Extended Speedup Models for Energy Efficiency in M/MCP*

The traditional methods of performance improvements such as extracting more parallelism from applications or increasing clock frequencies are losing their effect because of the power wall. Essentially, the speedup models have proven to be significantly useful for performance insights to parallel applications in M/MCP. However, the power-aware model can generalize familiar speedup models such as Amdahl's model [107] by considering the effects of active power-management strategies. A wide set of studies have extended speedup models particularly Amdahl's model for energy efficient computing. The Study in [31] evaluates HoMCP and HeMCP architectures to achieve highest performance for a given power budget. This study incorporates synchronization and communication overheads into Amdahl's model calculate the theoretical upper bound for performance. The results show that HeMCP achieves better performance than HoMCP for the same

power budget. [108] illustrates the interaction between the workload's parallelization and energy consumption in a parallelizable application. It derives the optimal operating frequencies allocated to the serial and parallel portions' regions in the specific application to minimize the total energy consumption, while the execution time is preserved. The paper shows a number of points, firstly the function of dynamic energy improvement due to parallelization rising faster with increasing the number of cores than the speedup improvement function of Amdahl's law. Secondly, it establishes the relationship between parallelization, speedup, and energy consumption. In sum, these points can be used in energy-aware M/MCP resource management. In addition, [109] presents a study of energy efficient improvements based on hardware parallelization rather than performance increase. It shows the ability to know the number of parallel cores which permits a system to achieve maximum energy improvements for a given throughput. It derives the number of formulas based on Amdahl's law for HoMCP and presents the serial and parallel ratios of parallel application. It gives the ability to find the optimum frequency, voltage, and energy improvements while preserving execution time. Furthermore, [110] analyzes the performance, power and power normalized performance of Mobile Internet Devices (MIDs) which became very important for industry, academia, and customers. This study presents prediction models and analysis models based on Amdahl's law. Moreover, [60] investigates the effects of power constraints on energy efficiency and scalability of HoMCP, HeMCP and hybrid CPU – GPU HeMCP. It presents analytical models which extend Amdahl's law by accounting for energy limitations. It suggests important improvements in hardware and software approaches. For hardware, it suggests that the future of M/MCP should include one or a few large cores along many small cores to support energy efficient hardware platforms. For software, it suggests that increasing the parallelism of applications should not be the exception, and it can create energy-efficient applications suitable for the future of M/MCP architectures.

Other literature also extends Hill-Marty's model for theoretical energy-efficient computing in M/MCP. [21] consider the principle of Hill-Marty with a view to estimate HoMCP and HeMCP performance and power consumption. Thus, the calculation of Power Normalized Performance (PNP) can be achieved by dividing performance, which is calculated from speedup, over power. PNP represents the performance achievable at the same cooling capacity. This metric is reciprocal to energy based on the definition of performance as the reciprocal of execution time. Additionally, It calculates the Energy Normalized

Performance (ENP) for evaluating the performance achievable in the same battery life cycle which is equivalent to the reciprocal of Energy-Delay Product (EDP).

2.6.1.7 *Dark Silicon*

There is a rich research literature focusing on the phenomenon of dark silicon. Dark silicon refers to the amount of circuitry of a chip that can not be powered on at the nominal operating voltage for a given (Thermal Design Power) constraint. This phenomenon presents new challenges and opportunities for CPU designers. It becomes more interesting particularly in the interaction with thermal management, Thermal Design Power diversity and reliability trade-offs and the leveraging and exploiting of variability concern. A simple theoretical study of dark silicon is presented by [111]. It uses extended Amdahl's law by considering Hill-Marty principle and using DVFS to decrease the amount of dark silicon and improve total performance. The paper presents a HeMCP model for diverse workload parallelism at different power budgets. The theoretical contributions make it possible to determine the improvement in calculated speedup and energy efficient system by using DVFS for a high parallel workload.

Dark silicon remains a major and significant challenge for the future of M/MCP scaling. [112, 113] demonstrate the understanding of the sustainability of M/MCP scaling in the historical exponential performance growth in the energy limited era. They bring together transistor technology, processors core and application models (HoMCP, HeMCP and DMCP Amdahl based models) to support this understanding. With the increase of the number of cores, the power constraints may prevent powering all cores at their highest speed. In this case, a portion of cores has to power off at all times to be *dark*. The models in these papers explain that the ratio of dark cores may be as much as 50% within three process generations based on ITRS projection. Thus, the effect of dark silicon may prevent the scaling to the high number of cores. The outcomes from this study show that realistic assumptions of M/MCP provide less performance gain over next three generations than parallel workload M/MCP scaling, and much less when parallel workload is unavailable. In summary, these papers suggest other directions to continue the historical rate of performance improvement.

Further, [114, 115] investigate the dark silicon phenomenon and provide experimental evidence for the (Electronic Design Automation) community by

running benchmarks on a M/MCP. One of the important sides of these experiments is the effects of p of an application on the calculated performance which calculated by Amdahl's law at the RTM. Further, [116] presents a technique called DsRem to managing thermally constrained resources of M/MCP. The technique selects the number of active cores with their voltage frequency scaling levels to maximize the overall performance by considering the (Instruction Level Parallelism) or (Thread Level Parallelism) nature of different Princeton Application Respository for Shared-Memory Computers (PARSEC) applications. These studies were followed by [117], which presents a new power budget concept called (Thermal Safe Power) which is a technique capable of providing safe power and power density constraints as a function of active cores. Thermal Safe Power has the ability to support M/MCP in many ways. If the cores are executing at power consumption below Thermal Safe Power, the (Dynamic Thermal Management) will not trigger. This method can inform task partition and core mapping decisions.

2.6.2 *Multi-Amdahl Model*

Multi-Amdahl is an analytic optimization technique aiming to help HeMCP design. It considers the diverse range of applications, the performance of each computational unit, and the total available resources [33, 118, 34]. [33, 34] establish Multi-Amdahl framework to model the performance of each computational unit as a function of available resources such as unit area, power or energy. Based on this framework, the analytical optimization for resource sharing among heterogeneous units may be obtained using the *Lagrange multiplier* method.

Hill-Marty's model has considered the impact of accelerating one portion of the workload on the overall execution as described in section 2.6.1.1 [3]. The Multi-Amdahl models [33, 34] generalize Amdahl's law from two types of execution environments *serial and parallel* to n types of different execution segments, by modeling different design constraints and accounting for their impacts. Thus, the new model is suitable for different resource constraints and efficiency models. However, the Multi-Amdahl models assume that only one of the computational units (accelerators) is active at any time, which is not the case for current and future real-life platforms and applications.

2.7 DISCUSSIONS AND CONCLUSIONS

This chapter includes a comprehensive survey of speedup model related literature. An existing survey [119] covers some of the fields but is not geared toward the direction of work covered by this thesis. This new survey helps establish the baseline for the research work reported in this thesis, which can be summarized into the following points. None of these points have been addressed by existing work. Table 2.1 compares existing work and this thesis in an organized manner.

1. This thesis extends the assumption of classical speedup models Amdahl, Gustafson and Sun-Ni to cover contemporary commercial system core homogeneity and heterogeneity. The models cover modern architectures such as Intel homogeneous M/MCP, ARM big.LITTLE heterogeneous cores, CPU – GPU configuration, FPGA-based acceleration schemes and complex structures with many types of cores and complex System-on-Chip (SoC).

2. This thesis extends speedup models to cover other non-functional properties including power and energy. These models provide the basis for studying the power/performance trade-offs for energy-efficient computing.

3. This thesis makes use of these models to evaluate the efficiency of the OS scheduler load balancing while running realistic application, particularly in a heterogeneous system.

4. This thesis proposes a new method of estimating the parallelization factor to study the effect of this factor on energy-efficient computing.

5. This thesis generates a new full-domain DVFS and per-core DVFS power models for M/MCP.

Table 2.1: Literature summary

Literature	Amdahl	Gustafson	Sun-Ni	Hill-Marty	Homogeneity	Heterogeneity	Dynamically	Distributed	CPU-GPU	Performance	Energy, Power	calculations Overhead	Factor Parallelization	modelling Theoretical	application Real	DVFS	Core affinity	Dark Silicon	Management Run Time
[11]	Yes	No	No	No	Yes	No	No	No	No	Yes	No	No	Controlled	Yes	No	No	No	No	No
[12]	Yes	Yes	No	No	Yes	No	No	No	No	Yes	No	No	Controlled	Yes	No	No	No	No	No
[13, 14]	Yes	Yes	Yes	No	Yes	No	No	No	No	Yes	No	No	Controlled	Yes	No	No	No	No	No
[27]	similar	No	No	No	Yes	No	No	No	No	Yes	No	No	Calculated	Yes	Yes	No	No	No	No
[3, 82]	Yes	No	No	Yes	Yes	Yes	Yes	No	No	Yes	No	No	Controlled	Yes	No	No	No	No	No
[49]	Yes	No	No	Yes	Yes	Yes	Yes	No	No	Yes	No	No	Controlled	Yes	No	No	No	No	No
[83]	Yes	Yes	No	Yes	Yes	Yes	Yes	No	No	Yes	No	No	Controlled	Yes	No	No	No	No	No
[20]	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	Yes	No	No	Controlled	Yes	No	No	No	No	No
[84]	No	No	No	No	No	No	No	No	No	No	No	Memory	No	Yes	No	No	No	No	No
[85]	No	No	No	No	No	Yes	No	No	No	Yes	No	Memory	No	Yes	No	No	No	No	No
[87]	No	No	No	No	No	No	No	No	No	Yes	Yes	Communication	No	No	Yes	No	No	No	No
[89]	No	No	No	No	No	No	No	No	No	No	No	Synchronization Communication	No	No	Yes	No	No	No	No
[88]	No	No	No	No	Yes	Yes	No	No	No	No	No	Communication	No	No	Yes	No	No	No	No
[91]	Yes	Yes	Yes	Yes	No	Yes	No	No	No	Yes	No	Memory	Controlled	Yes	No	No	No	No	No
[19]	Yes	Yes	Yes	Yes	Yes	No	No	No	No	Yes	No	Memory	Controlled	Yes	No	No	No	No	No
[31]	Yes	No	No	Yes	Yes	No	No	No	No	Yes	Yes	Synchronization Communication	Controlled	Yes	No	No	No	No	No
[92, 93]	Yes	No	No	No	Yes	Yes	No	Yes	Yes	Yes	No	Synchronization Communication	Controlled	Yes	Yes	No	No	No	No
[94]	Yes	No	No	Yes	Yes	No	No	No	No	Yes	Yes	Memory Communication	Controlled Calculated	Yes	Yes	No	No	No	No
[95]	Yes	No	No	Yes	Yes	Yes	No	No	Yes	Yes	No	Memory	Controlled	Yes	Yes	No	No	No	No
[28]	Yes	No	No	No	Yes	Yes	No	No	No	Yes	No	No	Calculated	Yes	No	No	No	No	No
[120]	Yes	No	No	No	Yes	No	No	No	No	Yes	No	No	Calculated	Yes	No	No	No	No	No

Table 2.1 Continued: Literature summary

Literature	Amdahl	Gustafson	Sun-Ni	Hill-Marty	Homogeneity	Heterogeneity	Dynamically	Distributed	CPU-GPU	Performance	Energy - Power	calculations Overhead	Factor Parallelization	modelling Theoretical	application Real	DVFS	Core affinity	Dark Silicon	Run Time Management
[98]	Yes	No	No	No	No	No	No	No	No	Yes	No	Communication	Controlled	Yes	No	No	No	No	Yes
[99]	Yes	No	No	No	Yes	No	No	No	No	Yes	No	Communication	Controlled	Yes	Yes	No	No	No	No
[100, 58]	Yes	Yes	No	Yes	Yes	No	No	Yes	No	Yes	No	Communication	Controlled	Yes	Yes	No	No	No	No
[101]	Yes	Yes	No	Yes	Yes	No	No	No	No	Yes	No	Communication	Evaluated	Yes	Yes	No	No	No	No
[102]	No	No	No	No	No	No	No	No	No	No	Yes	No	No	No	Yes	No	No	Yes	No
[103]	No	No	No	No	No	No	No	No	No	Yes	Yes	No	No	No	Yes	No	No	Yes	No
[104]	No	No	No	No	Yes	No	No	No	No	Yes	Yes	Time	No	Yes	Yes	Yes	Yes	Yes	No
[29]	Yes	No	No	No	Yes	No	No	No	No	Yes	Yes	Load balancing Scheduling	Program parallelism	Yes	Yes	No	No	Yes	No
[105]	No	No	No	No	Yes	No	No	No	No	Yes	No	Time	No	No	Yes	No	Yes	No	No
[106]	No	No	No	No	Yes	Yes	No	No	No	Yes	Yes	Time	No	No	Yes	Yes	Yes	Yes	No
[107]	Yes	No	No	No	No	No	No	No	No	Yes	Yes	Time	No	No	No	No	No	No	No
[21]	Yes	No	No	Yes	Yes	Yes	No	No	No	Yes	Yes	No	controlled	Yes	No	No	No	No	No
[60]	Yes	No	No	No	Yes	Yes	No	No	Yes	Yes	No	No	controlled	Yes	No	No	No	No	No
[108]	Yes	No	No	No	Yes	No	No	No	No	No	Yes	No	controlled	Yes	No	No	No	No	No
[109]	Yes	No	No	No	Yes	No	No	No	No	Yes	Yes	Communication	Controlled	Yes	Yes	No	No	No	No
[110]	Yes	No	No	No	Yes	No	No	No	No	Yes	Yes	No	No	Yes	Yes	No	No	No	No
[111]	Yes	No	No	Yes	Yes	No	No	No	No	Yes	Yes	No	Controlled	Yes	No	Yes	No	No	Yes
[112, 113]	Yes	No	No	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Memory	Controlled	Yes	Yes	No	No	No	Yes
[114]	Yes	No	No	No	Yes	No	No	No	No	No	Yes	No	No	Yes	Yes	No	Yes	Yes	Yes
[115]	No	No	No	No	Yes	No	No	No	No	No	Yes	No	No	Yes	Yes	No	Yes	Yes	Yes
[116]	No	No	No	No	Yes	No	No	No	No	Yes	Yes	No	No	Yes	Yes	No	Yes	Yes	Yes
[117]	No	No	No	No	Yes	Yes	No	No	No	Yes	Yes	No	No	Yes	Yes	Yes	Yes	Yes	Yes
[33, 118, 34]	Yes	No	No	Yes	Yes	Yes	No	No	Yes	Yes	No	No	Controlled	Yes	No	No	No	No	No

3

SPEEDUP AND POWER SCALING MODELS

3.1 INTRODUCTION

From the early days of computing systems, there has been a persistent engineering effort to improve computation speed by distributing the work across multiple devices. Predicting the system's gain in performance, called the *speedup*, has been a major focus in this area of the research. Amdahl's law has been known since 1967 [11]. It assumes that a fixed workload is executed in n processors and compares the performance with the same workload executed in a single processor. The model shows that the speedup will quickly saturate with increasing n if the workload requires synchronization. In 1988, Li and Malek explicitly considered inter-processor communications in this model [98]. In the same year, Gustafson introduced the principle of workload scaling pertaining to the fixed time model [12]. This model proposes to extend the workload proportionally to the scalability of systems with the result of having a linear increase in the speedup. In 1990, Sun and Ni suggested a new model, which included extended workload calculations by considering the capability of the memory [13, 14].

Over the years, technology scaling has facilitated significant performance improvement at reduced power consumption through increased operating frequency and smaller device geometries [5]. The number of transistors per unit of area have increased substantially conforming to Moore's [6] and Koomey's laws [7], and Pollack's rule suggests that performance is increasing approximately proportional to the square root of the complexity [10].

As a result, nowadays almost every consumer device or embedded system uses the computational power of M/MCP. The number of cores in a device is constantly growing, hence the speedup scaling models remain of high importance. They provide a valuable insight into system scalability and have become pivotal for multi-scale systems research. However, it is still important to keep the models up to date, to make sure they stay relevant and correctly represent novel aspects of platform design.

With increasing system complexity and integration, the concept of heterogeneous computation has emerged. Initially, the heterogeneity

appeared in a form of specialized accelerators, like GPU. In recent years, multiple types of CPU cores in a single device have also been made popular. For instance, the ARM big.LITTLE processor has found a wide use in mobile devices [23]. Heterogeneous systems pose additional engineering and research challenges. In the area of scheduling and load balancing, the aim is to improve core utilization for more efficient use of the available performance. Operating systems traditionally implement SMCP scheduling algorithms designed for homogeneous systems, and ARM have done dedicated work on modifying the Linux kernel to make load balancing suitable for their big.LITTLE architecture [121].

In addition to performance concerns, power dissipation management is also a significant issue in scalable systems: according to Dennard's CMOS scaling law [42] despite smaller geometries the power density of devices remains constant.

Hill-Marty extended Amdahl's speedup model to cover simple heterogeneous configurations consisting of a single big core and many smaller ones of exactly the same type [3], which relates to the CPU – GPU type of heterogeneity. The studies in [19, 20] extended Hill-Marty analysis to all three major speedup models. The problem of energy efficiency has been addressed in [21, 22] for the homogeneous and simple heterogeneous Amdahl's model. This overview covers only the most relevant publications, an extensive survey can be found in chapter 2 and summarized in Table 2.1.

In order to be relevant to more general and emerging types of heterogeneous systems, new types of models need to be developed. This chapter extends the classical speedup models to a so-called normal form representation of heterogeneity, which describes core performances as a vector. This representation can fit a wider range of systems, including the big.LITTLE processor, homogeneous processors with multiple DVFS islands, as well as multi-GPU heterogeneous systems.

The initial work on this topic includes the derivation of the speedup models as well as a set of power models for this extended representation of heterogeneity [122, 123]. In addition to fixed-workload Amdahl's law, workload scaling from the works of Gustafson and Sun-Ni have also been considered. In addition, we expand the work by addressing the effects of workload distribution and load balancing, and also explore additional modes of workload scaling relevant only to heterogeneous systems. We discover that the presented models inherit certain limitations from the Amdahl's law,

which may be significant for heterogeneous modeling and need to be taken into account. The work provides a concise discussion on the matter.

Table 3.1: Summary of the existing speedup models and the proposed model

	homogen.	heterogen.	power	memory	intercon.	Amdahl	Gustafson	Sun-Ni
[11, 124]	yes	no	no	no	no	yes	no	no
[98]	yes	no	no	no	yes	yes	no	no
[12]	yes	no	no	no	no	yes	yes	no
[13]	yes	no	no	yes	no	yes	yes	yes
[27]	yes	no	no	no	no	similar	no	no
[29]	yes	no	yes	no	no	yes	no	no
[94]	yes	simple	no	no	yes	yes	no	no
[3, 49]	yes	simple	no	no	no	yes	no	no
[19, 20]	yes	simple	no	yes	no	yes	yes	yes
[21]	yes	simple	yes	no	no	yes	no	no
proposed models	yes	normal form	yes	no	no	yes	yes	yes

The major ***contributions*** of this chapter are:

1. Extends the classical speedup models to normal form heterogeneity in order to represent modern examples of heterogeneous systems.
2. Extends models to include power estimation.
3. Clarifies the limitations of the Amdahl-like heterogeneous models and outlining further challenges of heterogeneous speedup and power modeling.

This work lays the foundation for newer types of heterogeneous models, and does not yet cover all system aspects. The effects of memory and interconnects are planned as future model extensions. Table 3.1 compares this chapter's contributions to the range of related research publications.

The experimental work which validating this chapter's models will be present in Chapter 4.

3.2 EXISTING SPEEDUP MODELS

In homogeneous systems, all cores are identical in terms of performance, power, and workload execution.

Homogeneous system considers system consisting of n cores, each core having a base core equivalent performance (θ) = unscaled workload size (I)/workload execution time in a single core ($t(1)$), where I is the given workload and $t(1)$ is the time needed to execute the workload on the core. This section describes various existing models for determining the $S(n)$ in relation to a single core, which can be used to find the homogeneous system performance on number of cores ($\Theta(n)$):

$$\Theta(n) = \theta S(n). \tag{3.1}$$

Amdahl-like speedup models are built around the parallelizability factor p, $0 \leqslant p \leqslant 1$, which reflects the application's capability of performing parallel computation. The fundamental assumption is that any application has a sequential part, which is not parallelizable at all, and a parallel part, which is infinitely parallelizable (i.e. it can be split into up to an infinite number of parallel threads). This leads to the following statement regarding the parallelizability factor: Given a total workload of I, the parallel part of a workload is pI and the sequential part is $(1-p)I$.

3.2.1 *Amdahl's Law (Fixed Workload)*

The general idea of this model is to compare execution time for some fixed workload I on a single core with the execution time for the same workload on the entire n-core system [11].

Time to execute workload I on a single core is $t(1)$, whereas workload execution time in the number of cores ($t(n)$) adds up the sequential execution time on one core at the performance θ and the parallel execution time on all n cores at the performance $n\theta$:

$$t(1) = \frac{I}{\theta}, \quad t(n) = \frac{(1-p)I}{\theta} + \frac{pI}{n\theta}, \tag{3.2}$$

thus the speedup can be found as follows:

$$S(n) = \frac{t(1)}{t(n)} = \frac{1}{(1-p) + \frac{p}{n}}. \tag{3.3}$$

This speedup in relation to n for different values of p is shown in Figure 2.5(a).

3.2.2 *Gustafson's Model (Fixed Time)*

Gustafson re-evaluated the fixed workload speedup model to derive a new fixed time model [12]. In this model, the workload increases with the number of cores, while the execution time is fixed. An important note is that the workload scales asymmetrically: the parallel part is scaled to the number of cores, whilst the sequential part is not increased. A classical example is the manufacture of copies of data. The reading of data is the sequential part and the writing of data can be infinitely parallelized. This kind of workload generally follows Gustafson's model because the writing part of the workload scales with the availability of writing hardware (e.g. cores). Hence this is not a fixed workload, but the speedup due to parallelization can still be studied by analyzing what happens during a fixed time period.

Let's denote the initial workload as I and extended workload as scaled workload size (I'). The time to execute initial workload is $t(n)$ and extended workload execution time ($t'(n)$). The workload scaling ratio can be found from:

$$t(1) = \frac{I}{\theta}, \quad t(n) = \frac{(1-p)\,I}{\theta} + \frac{pI'}{n\theta}, \tag{3.4}$$

and, since $t(1) = t(n)$, the extended workload can be found as $I' = nI$. The time that would take to execute I' on a single core is:

$$t'(1) = \frac{(1-p)\,I}{\theta} + \frac{pnI}{\theta}, \tag{3.5}$$

which means that the achieved speedup equals to:

$$S(n) = \frac{t'(1)}{t(1)} = (1-p) + pn. \tag{3.6}$$

Not all applications can provide workload extension only in the parallelizable part without changing the sequential part, hence there is a limitation on the applicability of the extended workload models. The main contribution of Gustafson's model, however, is to show that it is possible to build an application that scales to multiple cores without suffering saturation. Figure 2.5(b) shows Gustafson's model of the speedup in relation to n.

3.2.3 *Sun-Ni's Model (Memory Bounded)*

Sun and Ni took into account the previous two speedup models by considering the memory bounded constraints [13, 14]. In this model the execution time and the workload change according to the memory capability. The parameter workload/memory scaling ($g(n)$) reflects the scaling of the workload in relation to scaling the memory with the number of cores:

$$I' = g(n)\, I. \tag{3.7}$$

A typical example $g(n)$ is given for an $m \times m$ matrix multiplication, which has the memory requirement of $O(m^2)$ and the computation cost (workload) of $O(m^3)$. In this case, $g(n) = n^{\frac{3}{2}}$.

The model calculates the speedup as follows:

$$S(n) = \frac{t'(1)}{t'(n)} = \frac{(1-p) + pg(n)}{(1-p) + \frac{pg(n)}{n}}. \tag{3.8}$$

Because the workload is scaled by $g(n)$ according to (3.7), one of the important properties of this model is that for $g(n) = 1$ Sun-Ni's model (3.8) transforms into Amdahl's law (3.3), and for $g(n) = n$ it becomes Gustafson's law (3.6). Figure 2.5(c) shows the speedup for $g(n) = n^{\frac{3}{2}}$. Further in this chapter, we do not specifically relate $g(n)$ to the memory access or any other property of the system and consider it as a given or determined parameter pertaining to a general case of workload scaling.

3.2.4 *Hill-Marty's Heterogeneous Models*

Hill-Marty extended speedup laws for heterogeneous systems were mainly focused on a single high-performance core and many smaller cores of the same type [3].

Core performances are related to some BCE, which is considered to have $\theta = 1$. This model studies a system with one big core and $(n-r)$ little cores. The big core has relative performance of Hill-Marty assumption ($\Theta(r)$) while little cores have BCE performances, as shown in Figure 3.1(b). The sequential workload is executed on the faster core, while the parallel part exercises all cores simultaneously. This transforms Amdahl's law (3.3) as follows:

$$S(n, r) = \frac{1}{\frac{(1-p)}{\Theta(r)} + \frac{p}{\Theta(r)+(n-r)}}. \tag{3.9}$$

A reconfigurable version of this model can be extended to cover multi-GPU systems, but still implies only one active accelerator at a time [34, 22]. This chapter aims to cover more diverse cases of heterogeneity pertaining to such modern architectures as ARM big.LITTLE [125] and multi-GPU platforms with simultaneous heterogeneous execution, which are not directly covered by Hill-Marty's models.

3.3 HETEROGENEOUS SYSTEM

Homogeneous models are used to compare the speedup between different numbers of cores. Similarly, heterogeneous models should compare the speedup between core configurations, where each configuration defines the number of cores in each available core type. This section discusses the problems of modeling consistency across different core types and provides the foundation for all heterogeneous models presented later in this chapter.

3.3.1 *The Challenges of Heterogeneous Modeling*

Heterogeneous models must capture the performance and other characteristics across different types of cores in a comparable way. Such a comparison is not always straightforward, and in many ways similar to cross-platform comparison. This section discusses the assumptions behind Amdahl's law and

similar models under the scope of heterogeneous modeling and outlines the limitations they may cause.

3.3.1.1 *Hardware-dependent parallelizability*

In the models presented in Section 3.2, there is a clear time-separation of the synchronous and parallel executions of the entire workload. In other words, all threads synchronize at the same time. These models do not explore complex interactions between the processes, hence they do not provide exact timing predictions and should not be used for time-critical analyses like real-time systems research. Solving for process interactions is possible with Petri Net simulations [126] or process algebra [127]. Amdahl-like models, in contrast, focus on generic analytical solutions that give approximate envelopes for platform capabilities.

The parameter p is a workload property, assuming that the workload is running on ideal parallel hardware. Realistic hardware affects the p value of any workload running on it. From the standpoint of heterogeneous modeling, the potential differences in parallelizability between core types or cache islands will cause the overall p to change between core configurations. In this chapter, we do not attempt to solve this challenge. As demonstrated further in this chapter, it is still possible to build heterogeneous models around a constant p and use a range of possible values to determine the system's minimum and maximum speedup capabilities.

3.3.1.2 *Workload equivalence and performance comparison*

The workload is a model parameter that links performance with the execution time. In many cases, a popular metric for performance is Instructions per Second (IPS), in which case a workload is characterized by its number of instructions. IPS is convenient as it is an application-independent property of the platform; it is also used for deriving power optimization metrics such as Energy per Instruction (EPI).

In heterogeneous models, it is important to have a consistent metric across all core types. For devices of different architecture types, the same computation may be compiled into different numbers of instructions. In this case, the total number of instructions can no longer meaningfully represent the same workload, and IPS cannot be universally used for cross-platform performance comparison. This is particularly clear when comparing CPU and GPU devices.

In order to build a valid cross-platform performance comparison model, there is a need to reason about the workload as a meaningful computation, and two workloads are considered equivalent as long as they perform the same task. In this chapter, workload is measured in so-called "workload items", which can be defined on a case by case basis depending on the practical application. Respectively, instead of energy per instruction, energy per workload item will be used.

Hill-Marty's model, presented in Section 3.2.4, describes the performance difference between the core types as $\Theta(r)$. In real life, this relation is application dependent, as will be demonstrated in Sections 4.2 and 4.3. Differences in hardware, such as pipeline depth and cache sizes, cause performance differences on a per-instruction basis [128]. As a result, even within the same instruction set, core type i may execute workload A faster than core type j, but core type j may execute workload B faster than core type i. Hence, A and B must use different performance ratios to describe the same heterogeneous platform.

3.3.2 *Platform Assumptions*

We build our models under the assumptions listed below. These assumptions put limitations on the models as discussed earlier in this section. The same assumptions are used in the classical Amdahl's law and similar models, hence there is no further reduction in generality.

- The models and model parameters are both application and hardware specific.
- The relation between performances of cores of different types can be approximated to a constant ratio.
- The parallelizability factor p can be approximated by a constant and is known or can be determined (exactly or within a range).
- Environmental factors, such as temperature, are not considered.

Memory and communication-related effects [98, 99, 19] are not explicitly considered in this work and are the subject of future work outlined in Chapter 6.

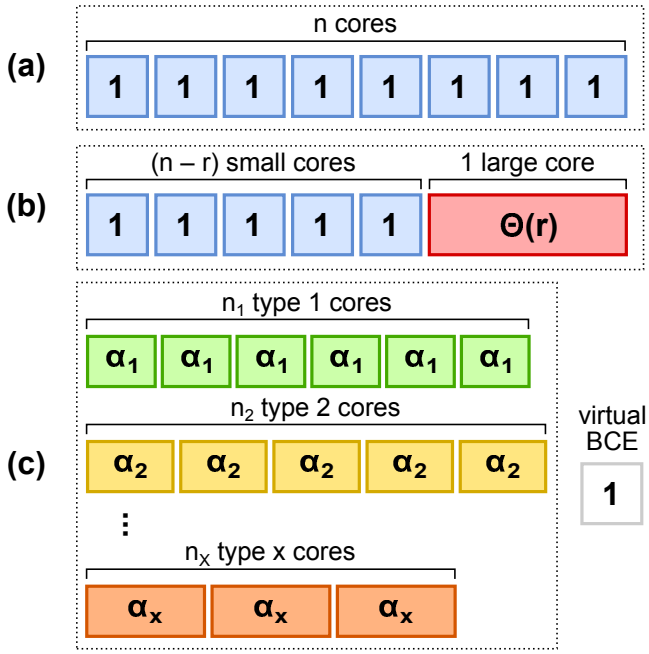


Figure 3.1: The proposed extended structure of a heterogeneous system (c) compared to a homogeneous system (a) and the previous assumption [3] on heterogeneity (b). The numbers in the core boxes denote the equivalent number of BCEs.

3.3.3 *Normal Form Representation of Heterogeneity*

Performance-wise, the models presented in subsequent sections describe heterogeneity using the following normal form representation.

The normal form of heterogeneous system configuration considered in this chapter consists of clusters (types) of homogeneous cores (x) with the numbers of cores defined as a vector of core numbers ($\overline{n}$) = ($n_1, \ldots, n_x$). The total number of cores in the system is denoted as total number of heterogeneous cores (N), which equal to summation of number of cores of type i (n_i), $N = \sum_{i=1}^{x} n_i$. The vector of the core performance factors ($\overline{\alpha}$) includes the performance factor for the core (α) of x clusters, viz $\overline{\alpha}$ = ($\alpha_1, \ldots, \alpha_x$) defines the performance of each core by cluster (type) in relation to some BCE, such that for all $1 \leqslant i \leqslant x$ we have $\theta_i = \alpha_i \theta$. As discussed earlier, the parameter $\overline{\alpha}$ is application- and platform-dependent. The structure is shown in Figure 3.1(c).

3.4 PROPOSED HETEROGENEOUS SPEEDUP MODELS

This section extends homogeneous speedup models for determining the heterogeneous speedup ($S(\overline{n})$) of a heterogeneous system in relation to a

single BCE, which can then be used to find the performance of the system using (3.1).

3.4.1 *Workload Distribution*

Homogeneous models distinguish two states of performance: the parallel execution exercises all cores, and the sequential execution exercises only one core while others are idle. The cores in such systems are considered identical, hence they all execute equal shares of the parallelizable part of the workload and finish at the same time. Consequently, the combined performance of the cores working in parallel is θn. In heterogeneous systems this is not as straightforward: each type of cores works at a different performance rate, hence the execution time depends greatly on the workload distribution between the cores. Imperfect distribution causes some cores to finish early and become idle, even when the parallelizable part of the workload has not been completed.

In real systems, the scheduler is assisted by a load balancer, whose task is to redistribute the workload during run-time from busy cores to idle cores. However, its efficiency is not guaranteed to be optimal [129]. The actual algorithm behind the load balancer may vary between different operating systems, and the load balancer typically has access to run-time only information such as CPU time of individual processes and the sizes of waiting queues. Hence it is virtually impossible to accurately describe the behavior of the load balancer as an analytical formula. This section addresses the problem by studying two boundary cases, which may provide a range of minimum and maximum parallel performances.

By definition, the *total execution time* for the workload I' is a sum of sequential execution time ($t'_s(\overline{n})$) and parallel execution time ($t'_p(\overline{n})$), and it represents the time interval between the first instruction in I' starting and the last instruction in I' finishing. This means that during a parallel execution, only the longest running core has an effect on the total execution time. In other words:

To be analogous to the homogeneous models and to simplify our equations, we also define the system's parallel performance via the *performance-equivalent number* of BCEs denoted as performance-equivalent number of Base Core Equivalents (N_α):

3.4.1.1 *Equal-share workload distribution*

In homogeneous systems, the parallelizable workload is equally split between all cores. As a results, many legacy applications, developed with the homogeneous system architecture in mind, would also equally split the workload by the total number of cores (threads). This leads to a very inefficient execution in heterogeneous systems, where everyone is waiting for the slowest core (thread), as illustrated in Figure 3.2(a). In this case, N_α is calculated from the minimum of $\overline{\alpha}$:

$$N_\alpha = N \cdot \min_{i=1}^{x} \alpha_i. \tag{3.10}$$

The above equation implies that the workload cannot be moved between the cores. If the system load balancer is allowed to re-distribute the work, then the real N_α may be greater than that calculated by (3.10). This equation can be used to define a lower performance bound corresponding to naïve scheduling policy with no balancing.

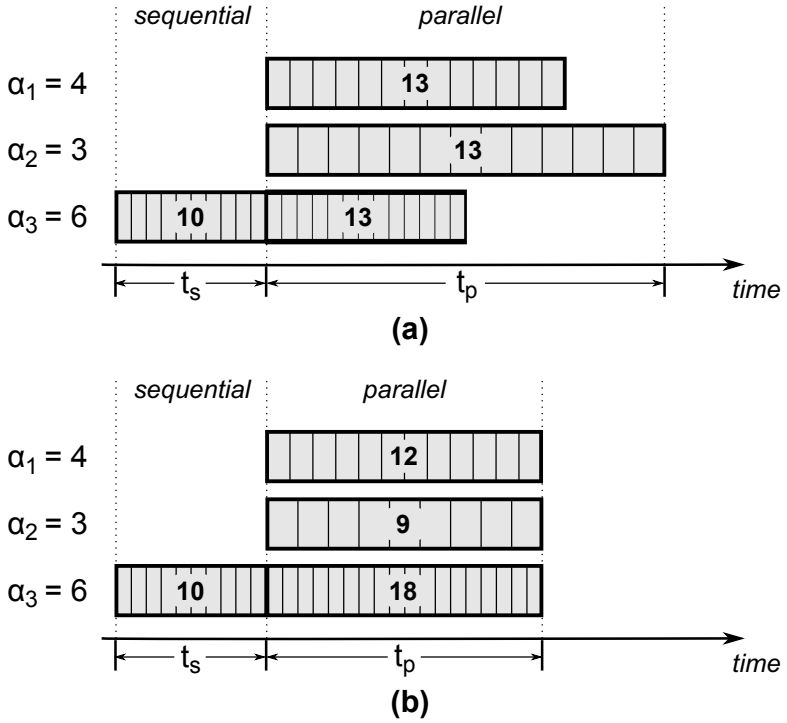


Figure 3.2: Workload distribution examples following (a) equal-share model and (b) balanced model.

3.4.1.2 *Balanced workload distribution*

Figure 3.2(b) shows the ideal case of workload balancing, which implies zero waiting time, hence all cores should theoretically finish at the same time. N_α for optimal workload distribution is as follows:

$$N_\alpha = \sum_{i=1}^{x} \alpha_i n_i. \tag{3.11}$$

$N_\alpha \theta$ represents the system's performance during the parallel execution, hence N_α values from (3.10) and (3.11) define the range for heterogeneous system parallel performances. A load balancer that violates the lower bound (3.10) is deemed to be worse than naïve. The upper bound (3.11) represents the theoretical maximum, which cannot be exceeded.

3.4.2 *Heterogeneous Amdahl's Law*

We assume that the sequential part is executed on a single core in the cluster type of core executing sequential workload (s), hence the system's performance during sequential execution is equal to multiplying the performance factor of sequential execution (α_s) by θ. In Section 3.4.1 parallel performance has defined as $N_\alpha \theta$. Hence, the time to execute the fixed workload I on the given heterogeneous system is:

$$t(\overline{n}) = t_s(\overline{n}) + t_p(\overline{n}) = \frac{(1-p)\,I}{\alpha_s \theta} + \frac{pI}{N_\alpha \theta}. \tag{3.12}$$

The speedup in relation to a single BCE is:

$$S(\overline{n}) = \frac{t(1)}{t(\overline{n})} = \frac{1}{\frac{(1-p)}{\alpha_s} + \frac{p}{N_\alpha}}. \tag{3.13}$$

One can verify that this equation also covers Hill-Marty's model (3.9), in which case $\overline{n} = (n-r, 1)$, $\overline{\alpha} = (1, \Theta(r))$, $\alpha_s = \Theta(r)$, and N_α is calculated for the balanced workload distribution (3.11).

3.4.3 *Workload Scaling*

As in the homogeneous case, Amdahl's law works with a fixed workload, while Gustafson and Sun-Ni allow changing the workload with respect to the system's capabilities. In this section we consider a general assumption on workload scaling, which defines the extended workload using characteristic functions parallel workload scaling factor ($g(\overline{n})$) and proportional workload scaling factor ($h(\overline{n})$) as follows:

$$I' = h(\overline{n}) \cdot ((1-p)\, I + p g(\overline{n})\, I), \tag{3.14}$$

where $h(\overline{n})$ represents the symmetric scaling of the entire workload, and $g(\overline{n})$ represents the scaling of the parallelizable part only.

The sequential and parallel execution times are respectively:

$$t'_s(\overline{n}) = \frac{(1-p)\, I \cdot h(\overline{n})}{\alpha_s \theta}, \quad t'_p(\overline{n}) = \frac{p g(\overline{n})\, I \cdot h(\overline{n})}{N_\alpha \theta}. \tag{3.15}$$

Hence, in the general case, for given workload scaling functions $g(\overline{n})$ and $h(\overline{n})$, the speedup is calculated as follows:

$$S(\overline{n}) = \frac{I'}{(t'_s + t'_p)\,\theta} = \frac{(1-p) + p g(\overline{n})}{\frac{(1-p)}{\alpha_s} + \frac{p g(\overline{n})}{N_\alpha}}. \tag{3.16}$$

Note that the speedup does not depend on the symmetric scaling $h(\overline{n})$. Indeed, the execution time proportionally increases with the workload, and the performance ratio (i.e. the speedup) remains constant. However, changing the execution time is important for the fixed-time Gustafson's model.

3.4.4 *Heterogeneous Gustafson's Model*

In the Gustafson's model, the workload is extended to achieve equal time execution: scaled workload execution time ($t'(\overline{n})$)= $t(1)$. For homogeneous Gustafson's model: $g(\overline{n}) = n$ and $h(\overline{n}) = 1$. For a heterogeneous system, there are more than one way to achieve equal time execution.

3.4.4.1 *Purely parallel scaling mode*

The maximum speedup for equal time execution is achieved by scaling only the parallel part, i.e. $h(\overline{n}) = 1$. We know that Gustafson's model requires equal execution time, hence:

$$t'_s(\overline{n}) + t'_p(\overline{n}) = t(1), \tag{3.17}$$

which leads to:

$$\frac{p\,g(\overline{n})}{N_\alpha} = 1 - \frac{(1-p)}{\alpha_s}. \tag{3.18}$$

From this, we can find that:

$$g(\overline{n}) = \left(1 - \frac{(1-p)}{\alpha_s}\right)\frac{N_\alpha}{p}, \tag{3.19}$$

however, this equation puts the number of restrictions on the system. Firstly, it doesn't work for $p = 0$, because it is not possible to achieve equal time execution for a purely sequential program if $\alpha_s \neq 1$ and only the parallel workload scaling is allowed. Secondly, a negative $g(\overline{n})$ does not make sense, hence the relation $\alpha_s > (1-p)$ must hold true. This means that the sequential core performance must be high enough to overcome the lack of parallelization. Another drawback of this mode is that it requires the knowledge of p to properly scale the workload.

In this scenario, the speedup is calculated from 3.16 and 3.19 as:

$$S(\overline{n}) = (1-p) + \left(1 - \frac{(1-p)}{\alpha_s}\right)N_\alpha. \tag{3.20}$$

3.4.4.2 *Classical scaling mode*

In order to remove the restrictions of the purely parallel scaling mode, and to provide a model generalizable to $p = 0$, we need to allow scaling of the sequential execution. However, since this mode potentially increases the sequential execution time, it exercises the cores less efficiently than the

previous mode and leads to lower speedup. In this case, (3.17) can be updated to:

$$h(\overline{n}) \cdot \left(t'_s(\overline{n}) + t'_p(\overline{n})\right) = t(1). \tag{3.21}$$

From this, in the case of $p = 0$, we find that $h(\overline{n}) = \alpha_s$. And for the case of $p > 0$ and $h(\overline{n}) = \alpha_s$:

$$g(\overline{n}) = \left(\frac{1}{h(\overline{n})} - \frac{(1-p)}{\alpha_s}\right)\frac{N_\alpha}{p} = \frac{N_\alpha}{\alpha_s}. \tag{3.22}$$

This scaling mode relates to the classical homogeneous Gustafson's model, which requires $g(n)$ to be proportional to the ratio between the system performances of the parallel and sequential executions. In the homogeneous case, if the sequential performance is θ, the parallel performance would be $n\theta$, leading to $g(n) = n$.

For the heterogeneous Gustafson's model in classical scaling mode, the speedup is calculated from 3.16 and 3.22 as:

$$S(\overline{n}) = \alpha_s(1-p) + pN_\alpha. \tag{3.23}$$

3.5 PROPOSED HETEROGENEOUS POWER MODELS

We base our power models on the concept of power state modeling, in which a device has the number of distinct power states, and the average power over an execution is calculated from the time the system spent in each state.

For each core in the system, we consider two power states: active and idle. Lower power states such as sleeping and shutting down the cores are not included in the presented models. However, it is possible to extend the models to cater to these effects. Let's denote the active power of a core in a homogeneous system as active power of a core (w_a) and the idle power of a core (w_0). Active power can also be expressed as a sum of idle power w_0 and effective power of Base Core Equivalent (w) that is spent on workload computation, $w_a = w_0 + w$. In this view, the idle component is no longer dependent on the system's activity and can be expressed as a system-wide

constant term total background power (W_0), called *background power*. The total power of the system (W_{total}) is:

$$W_{total} = W_0 + W(\overline{n}), \tag{3.24}$$

The total effective power ($W(\overline{n})$) is the total effective power of active cores – this is the focus of our models. The constant term of background power W_0 can be studied separately.

3.5.1 *Power Modeling Basics*

In the normal from representation of a heterogeneous system (Section 3.3), the difference between power dissipations of the cores is expressed by the vector of core power factors ($\overline{\beta}$) which includes power factor of core (β), viz $\overline{\beta} = (\beta_1, \ldots, \beta_x)$, which defines the effective power in relation to a BCE's effective power, such that for all $1 \leqslant i \leqslant x$ we have effective power w_i = power factors of core type i (β_i) by w. The effective power model can be found as a time-weighted average of the sequential effective execution power (w_s) and parallel effective execution power (w_p) of the system:

$$W(\overline{n}) = \frac{w_s t'_s(n) + w_p t'_p(n)}{t'_s(n) + t'_p(n)}, \tag{3.25}$$

where speedup-dependent sequential execution time ($t'_s(n)$) and speedup-dependent parallel execution time ($t'_p(n)$). They required to execute sequential and parallel parts of the extended workload respectively.

In a homogeneous system:

$$w_s = w \quad , w_p = nw. \tag{3.26}$$

In a heterogeneous system, if we execute the sequential code on a single core s:

$$\begin{aligned} w_s &= \beta_s w, \\ w_p &= N_\beta w, \end{aligned} \tag{3.27}$$

which gives for the balanced case of parallel execution (3.11):

$$N_\beta = \sum_{i=1}^{x} \beta_i n_i, \tag{3.28}$$

For equal-share execution (3.10), power-equivalent number of Base Core Equivalents (N_β) is calculated as follows:

$$N_\beta = \min \overline{\alpha} \cdot \sum_{i=1}^{x} \frac{\beta_i n_i}{\alpha_i}. \tag{3.29}$$

N_β is called a *power-equivalent number* of BCEs. Heterogeneous power models will transform into homogeneous if α_s = power factor of sequential execution (β_s) = 1 and $N_\alpha = N_\beta = n$.

3.5.2 *Power Distribution and Scaling Models*

We express the scaling of effective power in the system via the speedup and the power distribution characteristic function ($D_w(\overline{n})$):

$$W(\overline{n}) = w D_w(\overline{n})\, S(\overline{n}). \tag{3.30}$$

$D_w(\overline{n})$ represents the relation between the power and performance in a heterogeneous configuration. Since the speedup models are known from Section 3.4, this section focuses on finding the matching power distribution functions.

From (3.27) and (3.25), we can find that in the general case:

$$D_w(\overline{n}) = \left(\beta_s t'_s(n) + N_\beta t'_p(n)\right) \cdot \frac{\theta}{I'}, \tag{3.31}$$

thus substituting the workload scaling definition (3.14) and execution times (3.15) will give us:

$$D_w(\overline{n}) = \frac{\frac{\beta_s}{\alpha_s}(1-p) + p g(\overline{n}) \frac{N_\beta}{N_\alpha}}{(1-p) + p g(\overline{n})}. \tag{3.32}$$

It is worth noting that for homogeneous systems, power distribution function for homogeneous system ($D_w(n)$) = 1 in all cases, and the effective power equation will transform into:

$$W(n) = wS(n), \tag{3.33}$$

i.e. in homogeneous systems the power scales in proportion to the speedup.

Power distribution for Amdahl's workload, $g(\overline{n}) = 1$, hence the power distribution function becomes:

$$D_w(\overline{n}) = \frac{\beta_s}{\alpha_s}(1-p) + p \cdot \frac{N_\beta}{N_\alpha}, \tag{3.34}$$

Power distribution for Gustafson's workload following the same general form (3.32) for the effective power equation, we can find power distribution functions $D_w(\overline{n})$ for two cases of workload scaling described in Section 3.4.4.

For the classical scaling mode:

$$D_w(\overline{n}) = \frac{\beta_s(1-p) + pN_\beta}{\alpha_s(1-p) + pN_\alpha}. \tag{3.35}$$

For the purely parallel scaling mode:

$$D_w(\overline{n}) = \frac{\beta_s(1-p) + (\alpha_s - (1-p))N_\beta}{\alpha_s(1-p) + (\alpha_s - (1-p))N_\alpha}. \tag{3.36}$$

3.5.3 *Energy and Power-Normalized Performance*

Power modeling is typically used for optimizing system power dissipation. Due to the power-performance trade-off, advanced metrics are required as optimization targets. For example, PNP (performance per Watt) represents the performance achievable at a given power capacity. This parameter is the reciprocal of EPI (in our case, per workload item), which can be found from dividing the total power (3.24) by the system's performance (3.1):

$$EPI = \frac{W_{total}}{\Theta(n)} = \frac{W_0 + W(\overline{n})}{\theta S(\overline{n})}. \tag{3.37}$$

For a single BCE we can denote energy per workload item as a sum of effective energy (e) and idle energy (E_0). Applying the power model (3.30) to (3.37):

$$EPI = e \cdot D_w(\overline{n}) + \frac{E_0}{S(\overline{n})}. \tag{3.38}$$

The system-wide sum of idle energy per workload item is denoted as E_0. This equation shows that the power distribution function $D_w(\overline{n})$ increases the effective component of the energy as more power-hungry cores are being active, and the speedup $S(\overline{n})$ decreases the idle energy component due to better core utilization. The total energy consumption (E) during the execution of the extended workload I' is $E = EPI \; I'$.

EDP is another optimization metric that improves energy and performance at the same time. EDP is the reciprocal of power normalized performance squared (PNP^2). It is one of the possible weighted products with power and performance as the two criteria in the optimization of power and performance. Other weights are also possible but PNP and EDP (PNP^2) are the most commonly used ones. In our method, EDP can be obtained as follows:

$$EDP(\overline{n}) = W_{total} \cdot \left(\frac{t(\overline{n})}{I'}\right)^2. \tag{3.39}$$

3.6 DISCUSSION AND CONCLUSION

The models presented in this chapter enhance our understanding of scalability in heterogeneous many-core systems and will be useful for platform designers and electronics engineers, as well as for system level software developers.

This chapter extends three classical speedup models – Amdahl's law, Gustafson's model and Sun-Ni's model – to the range of heterogeneous system configurations that can be described as a normal form of heterogeneity. The provided discussion shows that the proposed models are not reducing applicability in comparison to the original models and may serve as a foundation for multiple research directions in the future. Important aspects, such as workload distribution between heterogeneous cores and various modes of workload scaling, are included in the model derivation. In addition to performance, this chapter addresses the issue of power modeling

by calculating power dissipation for the respective heterogeneous speedup models.

The extended models of performance and power in this chapter developed to cover performance-power trade-off. In addition, they can calculate different performance/energy metrics including PNP, EPI, and EDP. These models can be extended to cover other weighted product type metrics in a straightforward manner. The capabilities of the models are therefore not restricted to the example metrics provided in the chapter. One important comment is that because these models have been shown to work with weighted product optimization metrics, they can be used to reason about any kind of performance target involving speedup/throughput, and not restricted to the notion of energy efficiency.

4

EXPERIMENTAL VALIDATION OF SPEEDUP AND POWER SCALING MODELS

4.1 INTRODUCTION

The theoretical extensions of the classical speedup models in Chapter 3 potentially expands the use of Amdahl's, Gustafson's and Sun-Ni's models to cover the normal form of core heterogeneity. However, it is essential to validate these extended models through experimental investigation. This chapter describes an extensive set of experiments which help validate the proposed models on both heterogeneous big.LITTLE CPU and CPU – GPU on a dual-GPU laptop, and explore their practical use. The major ***contributions*** of this chapter can be summarize as follows:

1. Validates the extended models from Chapter 3 on real heterogeneous platforms through a set of carefully designed experiments.
2. Practically using these models to evaluate the efficiency of the Linux scheduler's load balancing while running realistic workloads in a heterogeneous system.

This chapter is presented in two parts, focusing on CPU-only in Section 4.2 and CPU – GPU in Section 4.3.

4.2 CPU-ONLY EXPERIMENTAL VALIDATIONS

This section describes attempts of validating the models presented in Sections 3.4 and 3.5 using a set of experiments on a real CPU-only heterogeneous platform. In these experiments, the goal is to determine the accuracy of the models when all model parameters, such as the parallelization factor p, are under control.

4.2.1 *Platform Description*

This study is based on a Multi-Core mobile platform, the Odroid-XU3 board [125]. The main part of it is the 28nm application processor Exynos 5422. As shown in Figure 4.1, the platform is an SoC hosting an ARM big.LITTLE heterogeneous processor consisting of four Cortex A7 cores (C0 to C3) and four Cortex A15 cores (C4 to C7). The big Cortex A15 is a high-performance 32-bit core having 32KB instruction and 32KB data L1 caches and 2MB L2 cache and the maximum frequency of 2.0GHz. The LITTLE Cortex A7 is a low power 32-bit core including the same L1 cache size and 512 KB L2 cache, and the maximum frequency of 1.4 GHz. There are compatible Linux and Android distributions available for Odroid-XU3; in our experiments we used Ubuntu 14.04. This SoC also has four power domains: A7 power domain, A15 power domain, GPU, and memory power domains. The Odroid-XU3 board allows per-domain DVFS using predefined voltage-frequency pairs.

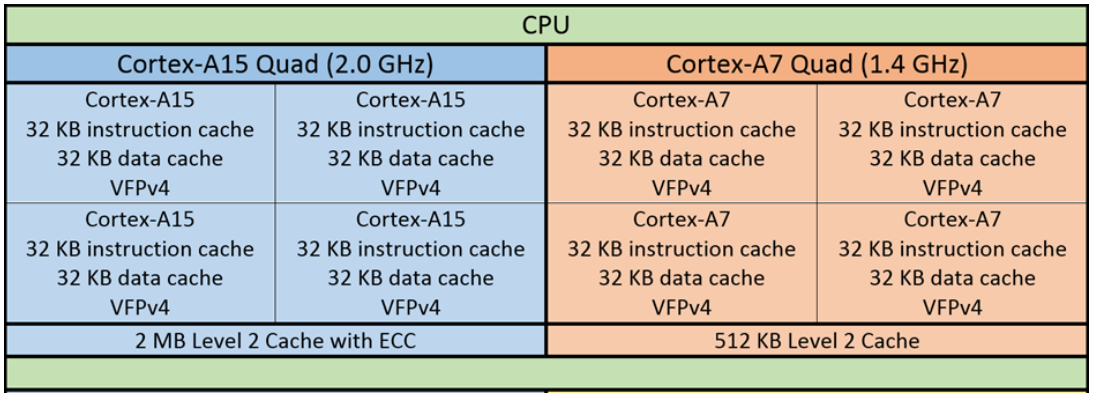


Figure 4.1: Experimental big.LITTLE platform description

The previous assumption by Hill and Marty for heterogeneous architectures, shown in Figure 3.1(b), cannot describe systems such as big.LITTLE. Our models do not suffer from these restrictions and can be applied to big.LITTLE and similar structures.

4.2.2 *Benchmark Description and Model Characterization*

The models operate on application- and platform-dependent parameters, which are typically unknown and imply high efforts in characterization. However, to prove that the proposed models work, it is sufficient to show that, if $\overline{\alpha}$, $\overline{\beta}$ and p are defined, the performance and power behavior of the system

follows the model's prediction. These parameters can be fixed by a synthetic benchmark. This benchmark does not represent realistic application behavior and was designed only for model validation purposes. Experiments with real application examples are presented in Section 4.4.

The model characterization is derived from single core experiments. These characterized models are used to predict M/MCP execution in different core configurations. The predictions are then cross-validated against experimental results.

4.2.2.1 *Controlled parameters*

The benchmark has been developed specifically for these experiments in order to provide control over the parallelization parameter p. Hence, p is not a measured parameter, but a controlled parameter that tells the application the ratio between the parallel (multi-threaded) and sequential (single thread) execution.

The application is based on Portable Operating System Interface for Unix (POSIX) threads, and its flow is shown in Figure 4.2. Core configurations, including homogeneous and heterogeneous, can be specified per application run as the sequential execution core s and the set of core allocations $(\bar{c})=(c_1,\ldots,c_N)$, where N is the number of parallel threads; $s, c_j \in \{C1,\ldots,C7\}$ for $1 \leqslant j \leqslant N$. C0 is reserved for OS and power monitors. These variables define $\overline{n}$ used in the models. We do not shut down the cores and use per-thread core pinning via `pthread_attr_setaffinity_np` to avoid unexpected task migration. To improve experimental setup and reduce the interference, we reserve one A7 core (C0) for OS and power monitors, hence it is not used to run the experimental workloads, and the following results show up to 3 A7 cores.

The workload size I and the workload scaling $g(\overline{n})$ are also given parameters, which are used to test Gustafson's models against Amdahl's law. The application implements three workload functions: square root calculation (sqrt), integer arithmetic (int), and logarithm calculation (log) repeated in a loop. These computation-heavy tasks use minimal memory access to reduce the impact of hardware on the controlled p. A fixed number of loop iterations represents one workload item. The functions are expected to give different performance characteristics, hence the characterization and cross-validation experiments are done separately for each function.

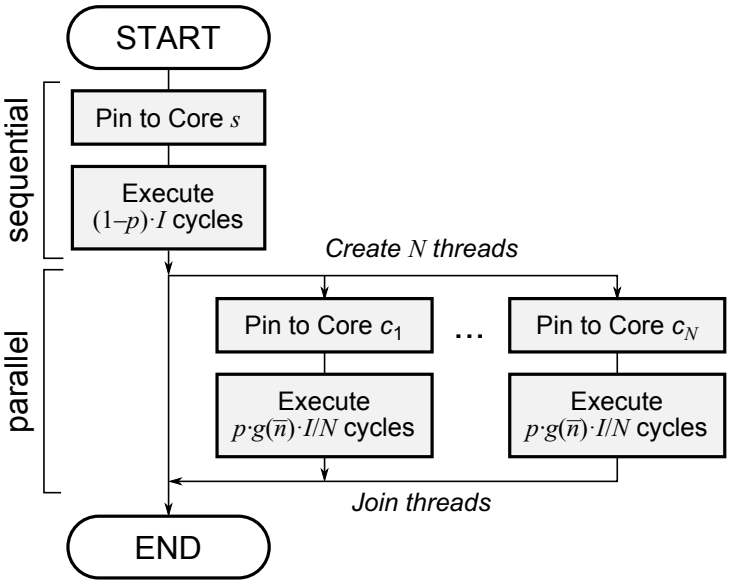


Figure 4.2: Synthetic application with controllable parallelization factor and equal-share workload distribution. Parameter p, I, $g(\overline{n})$, N, s and $\overline{c}=(c_1,\ldots,c_N)$ are specified as the program arguments.

Figure 4.2 shows equal-share workload distribution, where each parallel thread receives equal number of $pg(\overline{n})\frac{I}{N}$ workload items. This execution gives N_α and N_β that correspond to naïve load balancing according to (3.10) and (3.29). Additionally, after collecting the characterization data for $\overline{\alpha}$, we implemented a version that uses $\overline{\alpha}$ to do optimal (balanced) workload distribution by giving each core $c_j \in \overline{c}$ a performance-adjusted workload of $pg(\overline{n})\frac{I}{N}\cdot\frac{\alpha_j}{A}$, where $A=\sum_{j=1}^{N}\alpha_j$. This execution follows different N_α and N_β, which can be calculated from (3.11) and (3.28).

4.2.2.2 *Relative performances of cores*

All experiments in this section are run with both A7 and A15 cores at 1.4GHz. Running both cores at the same frequency exposes the effects of architectural differences on the performance. In addition, by avoiding higher frequencies we reduce the temperature effects and avoid throttling. In this study, we set BCE to A7, hence $\alpha_{A7}=1$; and α_{A15} can be found as a ratio of single core execution times $\alpha_{A15}=t_{A7}(1)/t_{A15}(1)$, as shown in Table 4.1. The three different workload functions have different α_{A15} values.

It can be seen that A15 is unsurprisingly faster than A7 for integer arithmetic and logarithm calculation, however, the square root calculation is faster on A7. This is confirmed multiple times in many experiments. We did not fully investigate the reason of this behavior since the board's production and support have been discontinued, and this is in any case outside the scope of

this chapter. A newer version of the board, Odroid-XU4, which is also built around Exynos 5422, does not have this issue. It is important to note that we compiled all our benchmarks using the same `gcc` settings. We include this case of non-standard behavior in our experiments to explore possible negative impacts on the performance modeling and optimization.

Table 4.1: Characterization experiments: single core execution

benchmark	sqrt		int		log	
base workload	40000		40000		40000	
core type i	A7	A15	A7	A15	A7	A15
measured execution time, ms	49969	53206	52844	42665	41820	23506
measured active power, W	0.2655	0.8361	0.2760	0.8305	0.3036	0.9496
power measurement std dev	0.82%	0.18%	0.96%	0.87%	0.93%	0.42%
calculated effective power, W	0.1158	0.4887	0.1264	0.4830	0.1540	0.6022
performance factor for the core type i (α_i)	1	0.9392	1	1.2386	1	1.7791
β_i	1	4.2183	1	3.8221	1	3.9094

4.2.2.3 *Core idle and active powers*

The Odroid-XU3 board provides power readings per power domain, i.e. one combined reading per core type, from which it is possible to derive single core characteristic values w_0 and w.

Idle powers are determined by averaging over 1 min of measurements while the platform is running only the operating system and the power logging software. The idle power values are $w_{0,A7} = 0.1571$W and $w_{0,A15} = 0.3552$W, which are used across all benchmarks. The standard deviation during the idle power measurements is 1.1% of the mean value.

Effective powers w_{A7}, w_{A15} are calculated from the measured active powers by subtracting idle power according to (3.24). The power ratios are then found as $\beta_{A7} = 1$ and $\beta_{A15} = w_{A15}/w_{A7}$; the values are presented in Table 4.1.

4.2.3 *Amdahl's Workload Outcomes*

A large number of experiments have been carried out covering all functions (sqrt, int, log) in various core configurations, and repeated for $p = 0.3$ and $p = 0.9$. This set of runs use a fixed workload of 40000 items with equal-share workload distribution between threads. Model predictions and experimental measurements for a single example are shown in Figures 4.3 and 4.4; the full data set can be found in the Appendix B, Figures B.1, B.2, B.3, B.4. The measured speedup is calculated as the measured time for a single A7 core execution $t_{A7}(1)$, shown in Table 4.1, over the benchmark's measured execution time, in other words unscaled workload execution time ($t(\overline{n})$):

$$S(\overline{n}) = \frac{t_{A7}(1)}{t(\overline{n})}. \tag{4.1}$$

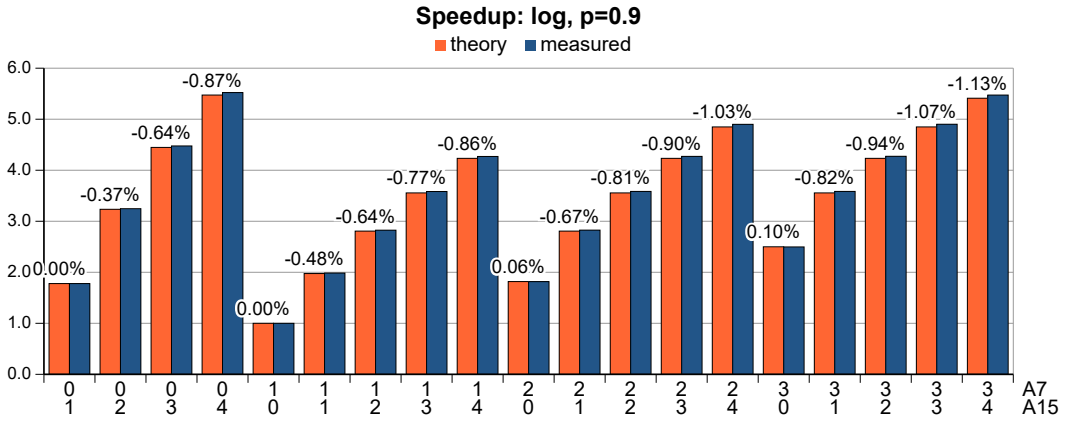


Figure 4.3: Speedup validation results for the heterogeneous Amdahl's law showing percentage error of the theoretical model in relation to the measured speedup.

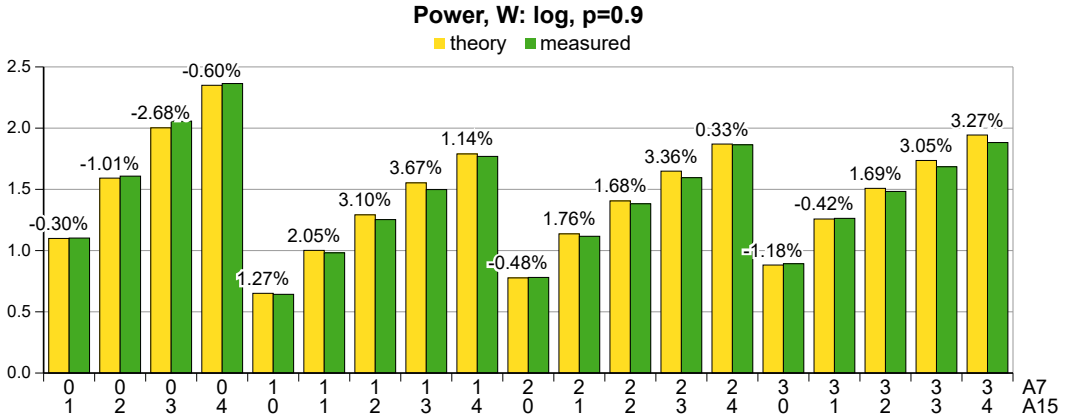


Figure 4.4: Total power dissipation results for the heterogeneous Amdahl's law showing percentage error of the theoretical model in relation to the measured power.

The observations validate the model (3.13) by showing that the differences between the model predictions and the experimental measurements are very small. The speedup error never exceeds 1%, and the power error never exceeds 4%, which is comparable to the standard deviation of the characterization measurements. A possible explanation for the low error values can be that our synthetic benchmark produces very stable $\overline{\alpha}$ and $\overline{\beta}$, and accurately emulates p. However, these small errors also prove that the model can be used with high confidence if it is possible to track these parameters. The model can also be confidently used in reverse to derive parallelization and performance properties of the system from the speedup measurements, as demonstrated in Section 4.4.

The counter intuitive result for 7-core (three A7 cores and four A15 cores) execution having lower power dissipation than four A15 cores and no A7 cores can be explained by the equal-share workload distribution. Because the parallel workload is equally split between these cores, the A15 cores finish early and wait for A7 cores. This idling reduces the average total power dissipation, however, it implies that intelligent workload distribution can improve core utilization by scheduling more tasks to A15 cores than to A7 ones so that they finish at the same time. This is investigated in Section 4.2.5.

4.2.4 *Gustafson's Workload Outcomes*

Two sets of experiments have been carried out to validate heterogeneous Gustafson's models in both purely parallel and classical workload scaling modes described in Section 3.4.4. The initial workload I is set to 40000, and the scaled workload I' is defined by (3.14). These experiments also use equal-share workload distribution and s is fixed to A15.

The measured speedup is calculated as the ratio of performances according to (3.1), or as the time ratio multiplied by the workload size ratio: $S(\overline{n}) = (t_{A7}(1)/t(\overline{n})) \cdot (I'/I)$. The observed errors are similar to Amdahl's model with the speedup estimated within 3.21% error (0.54% average) and the power dissipation estimated within 6.23% difference between the theory and the measurements. A complete set of data can be found in the Appendix B, Figures B.5, B.6, B.7, B.8.

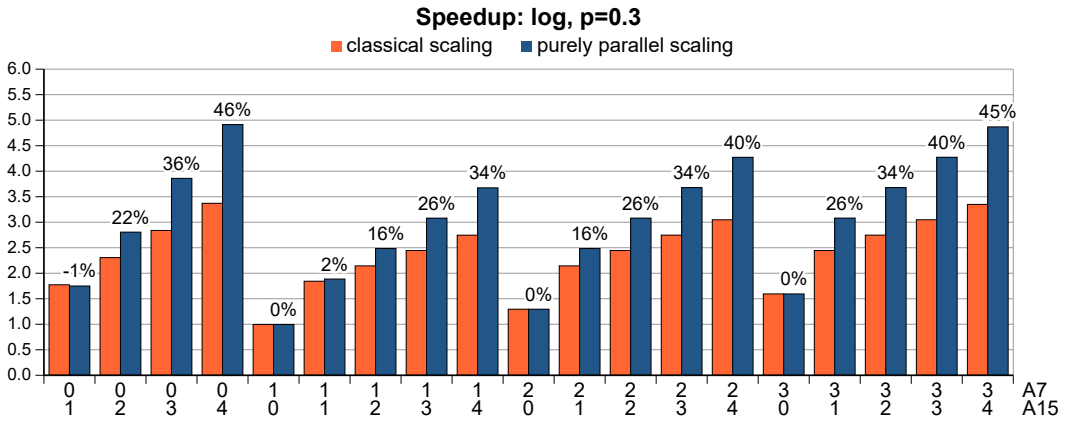


Figure 4.5: Gustafson's model outcomes showing the measured speedup gain from using the purely parallel workload scaling compared to the classical scaling.

Figure 4.5 compares the speedup between two workload scaling modes for $p = 0.3$. The purely parallel scaling has more effect for less parallelizable applications as it focuses on reducing the sequential part of the execution, hence the experiments with $p = 0.9$ show insignificant gain in the speedup and are not presented here. Even though the purely parallel scaling is harder to achieve in practice as it requires the knowledge of p, it provides a highly significant speedup gain, especially if the difference between the core performances is high, which, in the case of log, gives almost 50% better speedup.

4.2.5 *Balanced Execution*

Previously described experiments use equal-share workload distribution, which is simpler to implement, but results in faster cores being idle while waiting for slower cores. The balanced distribution, defined in (3.11), gives the optimal speedup for a given workload. This section implements balanced distribution of a fixed workload and compares it to the equal-share distribution outcomes of Amdahl's law. The results are presented for $p = 0.9$, as it provides larger differences between equal-share and balanced distributions.

In terms of model validation, the results are also very accurate, giving up to 4.63% error in power estimation and within 1.3% error for the speedup. Figure 4.6 explores the differences between the equal-share and optimal (balanced) cases of workload distribution in terms of performance and energy properties of the system. The balanced distribution gives up to 41% increase in the speedup. The average power dissipation is also increased up to 36% as the cores are exercised with as little idling as possible.

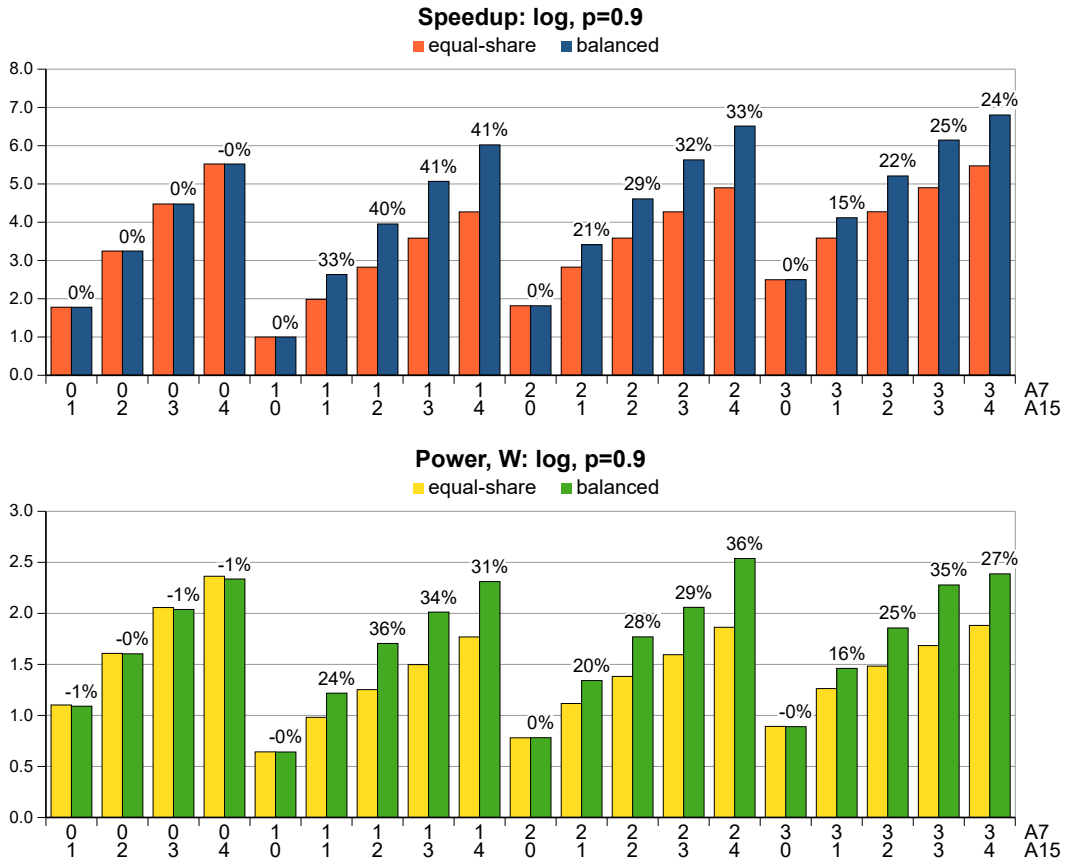


Figure 4.6: Comparison of the measured speedup, power, and energy between equal-share and balanced execution.

4.3 CPU-GPU EXPERIMENTAL VALIDATIONS

The previous section explores the heterogeneity within the devices having the same instruction set. However, many modern platforms also include specialized accelerators such as general purpose GPUs.

Open Computing Language (OpenCL) programming model [130] enables cross-platform development for parallelization by introducing the notion of a *kernel*. A kernel is a small task written in a cross-platform language that can be compiled and executed in parallel on any OpenCL device. It also provides a hardware abstraction level. GPU devices often have a complex hierarchy of parallel computation units: a few general purpose units can have access to a multitude of shader ALUs, which in turn implement vector instructions that may also be used to parallelize scalar computation. Consequently, behind the OpenCL abstraction, we consider n_i not as the number of device cores but as a *degree of parallelism* – the number of kernels that can be executed in parallel.

This section presents the experimental validation using the synthetic benchmark shown in Figure 4.2, but reimplemented in OpenCL with kernels

replacing POSIX threads. The kernels implement the same looped computation (sqrt, int, and log). The source code for OpenCL version is also available [1].

4.3.1 *Platform Description and Characterization*

The experiments presented in this section have been carried out on a 2012 Dell XPS 15 laptop with Intel Core-i7 CPU (denoted as CPU further in this section) and two GPUs: integrated GPU (intGPU) and a dedicated Nvidia card (Nvidia). Table 4.2 shows device specifications as reported by OpenCL. The platform runs Windows 7 SP1 and uses OpenCL v1.2 as a part of Nvidia Compute Unified Device Architecture (CUDA) framework. The platform has no facility to measure power to the granularity required for the power model validation, hence this section is focused only on the speedup. Time measurement is done using the combination of the system time (for long intervals) and OpenCL profiling (for short intervals).

An important feature of the platform is that both GPU devices can execute the workload at the same time. This is done by individually calling `clEnqueueNDRangeKernel` on separate OpenCL device contexts. This chapter's models cover this type of heterogeneity, while the reconfigurable Hill-Marty model [3] can model only one active type of parallel cores at a time. This has been the primary criterion for selecting a CPU – GPU platform for this section.

Table 4.2: OpenCL device capabilities

core type i	CPU	intGPU	Nvidia
device name	Intel Core i7-3520M	Intel HD Graphics 4000	GeForce GT 640M
max core freq	2.9GHz	350MHz	708MHz
compute units	4 (2+hyper)	16	2 (384 shaders)
max workgroup	1024	512	1024
max n_i	1	256	1024 (log: 64)

OpenCL adds overheads when scheduling the kernel code and copying data. However, we use computation-heavy benchmarks that do not scale the memory requirement with the workload, hence the overhead is constant, and becomes negligible if the primary computation is large enough. A series of experiments

have been carried out to find out the smallest required computation for OpenCL overheads to be negligible: 10^6 work items, as demonstrated in Figure 4.7.

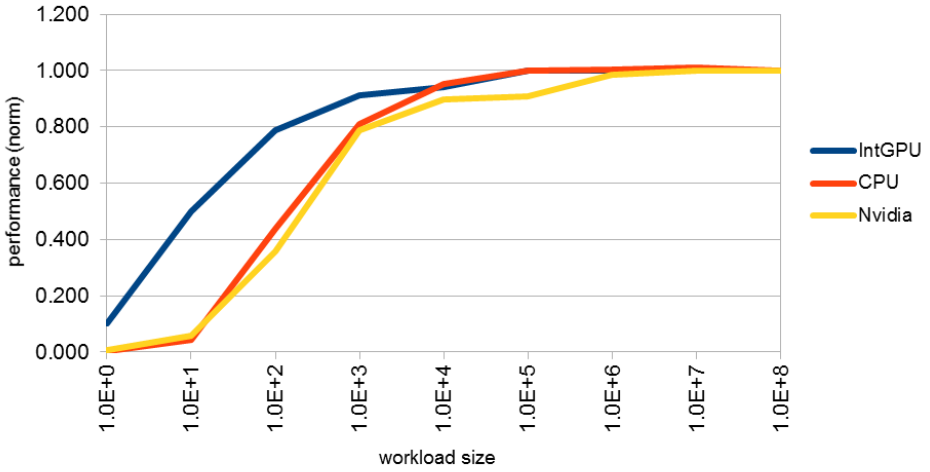


Figure 4.7: The effect of OpenCL overheads on performance, can be ignored for sufficiently large workload sizes.

Table 4.2 reports the max workgroup size, which represents the maximum number of "parallel" kernels, although OpenCL does automatic sequentialization if there are not enough real computation units. We experimentally find the real maximal degree of parallelism n_i for each benchmark by attempting to execute $p = 1$ workload and increasing the number of cores until the scaling is no longer linear. Figure 4.8 shows the outcome. The first observation is that CPU does not scale well in OpenCL, hence it has been decided to limit CPU to a single core used only for sequential execution. Nvidia scales perfectly for sqrt and int to its maximum allowed workgroup, but with log its performance starts to drop after 64 and completely flattens at 256. An interesting behavior is observed with intGPU: starting from 16 cores its performance drops by 25% (15% with log), but otherwise, the scaling continues to be perfectly linear up to 256 and slightly dips at 512. We model this by representing intGPU as two devices, as shown in Table 4.3. intGPU is used as BCE, and the performance ratios $\overline{\alpha}$ are calculated as the ratio of execution times from single core experiments (except for intGPU16+, which uses 64-core execution time).

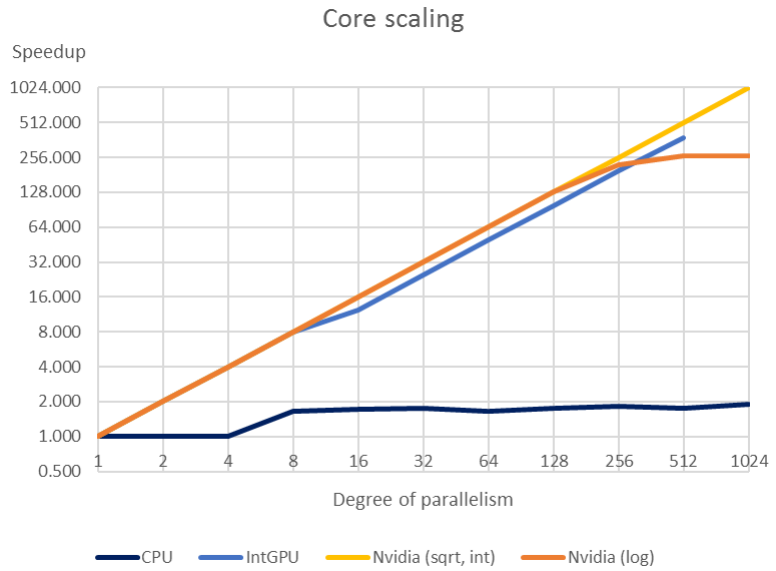


Figure 4.8: Investigating the scalability potential for the requested $p = 1$.

Table 4.3: OpenCL characterization experiments

bench	core type i	workload	exec time, ms	α_i
sqrt	CPU	$8.0 \cdot 10^7$	3335	24.351
	intGPU	$4.0 \cdot 10^6$	4060	1
	intGPU16+		5281	0.769
	Nvidia	$8.0 \cdot 10^7$	5421	14.980
int	CPU	$8.0 \cdot 10^7$	953	44.819
	intGPU	$8.0 \cdot 10^6$	4273	1
	intGPU16+		5553	0.769
	Nvidia	$8.0 \cdot 10^7$	5421	7.881
log	CPU	$8.0 \cdot 10^7$	2158	42.194
	intGPU	$8.0 \cdot 10^6$	4554	1
	intGPU16+		5318	0.856
	Nvidia	$1.0 \cdot 10^6$	4613	0.247

4.3.2 *Speedup Validation Outcomes*

Due to the sheer amount of work, it is not practical to test all possible core combinations. Instead, we select configurations evenly distributed across the range. The following models have been experimentally validated: equal-share Amdahl's law and balanced Amdahl's law. Every core type is tested in the role of sequential executor s in the case of $p = 0.9$. For $p = 0.3$, CPU is always used for s as the fastest of the devices. The speedup is calculated against the single core intGPU experiment. Since the other devices are generally much

faster, and Nvidia is capable of executing 1024 parallel kernels, the observed maximum speedup is 395.7 for Amdahl's workload, $p = 0.9$.

Figure 4.9 shows a typical result for Amdahl's law; the full set of results can be found in the Appendix B, Figures B.11, B.12, B.13. On average, the experiments with Amdahl's law show 1% error across the tested core combinations, but going up to 6-8% in a few points. The performance difference between the balanced and equal-share workload distributions is presented in Figure 4.10. Given the core performances of intGPU and Nvidia are very different, load balancing plays a crucial role, and can provide over 400 percent performance boost in some cases.

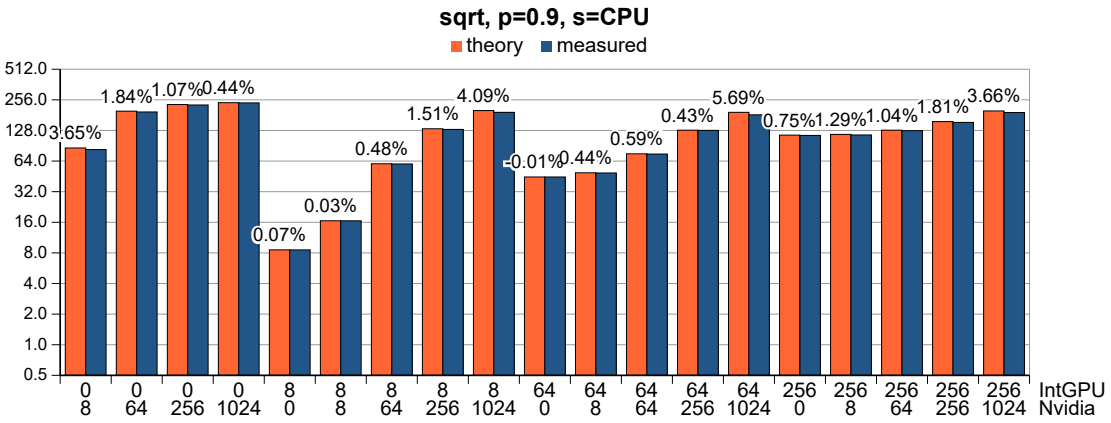


Figure 4.9: Speedup validation results for the heterogeneous Amdahl's law in the OpenCL platform.

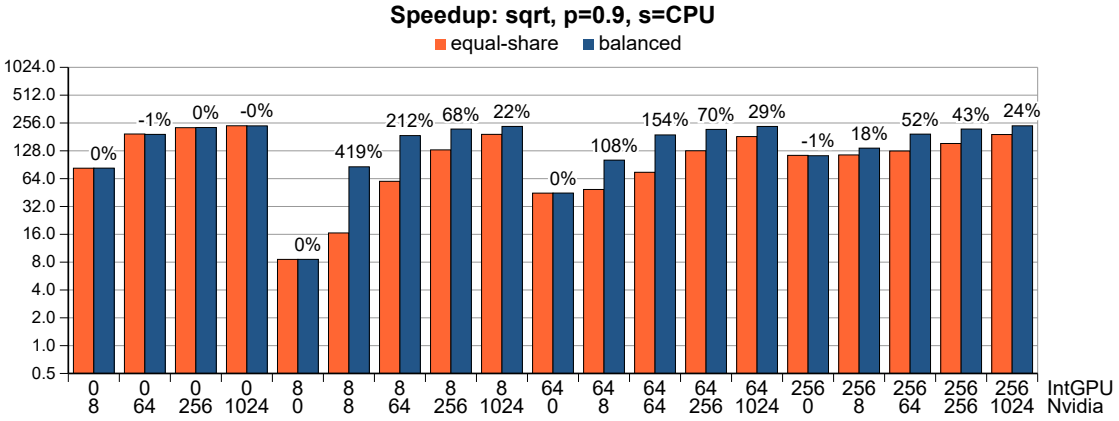


Figure 4.10: Comparison of the measured speedups between equal-share and balanced execution in the OpenCL platform.

4.4 REALISTIC APPLICATION WORKLOADS

This section is focused on experiments with realistic workloads based on the PARSEC benchmark suite [131, 132]. PARSEC is a reference application suite used in many fields including industry and academia, for studying concurrent applications on parallel hardware. Some of them parallelized with Open Multi-Processing (OpenMP), while the others parallelized with gcc-pthreads. PARSEC benchmarks are designed for parallel multi-threaded computation and include diverse workloads that are not exclusively focused on high-performance computing. Each application is supplied with a set of pre-defined input data, ranging from small sizes (test) to large (simlarge) and very large (native) sizes. Each input is assumed to generate a fixed workload on a given system. To our knowledge, PARSEC benchmarks do not implement workload scaling to Gustafson's or Sun-Ni's models, hence this section is focused on Amdahl's law only.

In our experiments we run a subset of PARSEC benchmarks, namely ferret (CPU-heavy), fluidanimate (memory-heavy), and bodytrack (mixed), and use simlarge input. Core pinning of is done at the application level using the `taskset` command in Linux. The command takes a set of cores as an argument and ensures that every thread of the application is scheduled onto one of these cores. However, the threads are still allowed to move between the cores within this set due to the influence of the system load balancer [129]. This is different from the synthetic benchmark described in Section 4.2, which by itself and not through a system function, performed secure pinning of individual threads, one thread per core.

In this chapter, we do not study the actual algorithm of the load balancing or the internal structure of PARSEC benchmarks, hence the workload distribution between the cores is considered a black box function: N_α is unknown. Section 3.4.1 addressed this issue by providing the range of values for N_α. The minimum value corresponds to equal-share workload distribution and gives the lower speedup limit ($S_{low}(\overline{n})$); the maximum value is defined by the balanced workload and gives higher speedup limit ($S_{high}(\overline{n})$).

The goal of the following experiments is to calculate these limits and to find how the real measured speedup fits in the range. The relation provides a quality metric of load balancimg algorithm (q), where q=1 corresponds to the theoretically optimal load balancer, and q=0 is equivalent to a naive approach (equal-share). Negative values may also be possible and show that

the balancing algorithm is not working properly and creates an obstacle to the workload execution. The metric q is calculated as follows:

$$q = \frac{S(\overline{n}) - S_{low}(\overline{n})}{S_{high}(\overline{n}) - S_{low}(\overline{n})}. \tag{4.2}$$

The motivation for load balancing is to improve speedup by approaching the balanced workload behavior. Hill-Marty [3] and related existing work [19, 20] covering core heterogeneity all assume that the workload is already balanced in their models, implying q=1. This work makes no such assumption and studies real load balancer behaviors for different benchmarks, using novel models facilitating quantitative comparisons.

4.4.1 *Model Characterization*

The PARSEC experiments are executed on the Odroid XU3 platform described in Section 4.2. The model characterization is obtained from the homogeneous configuration experiments, and then the models are used to predict system behavior in heterogeneous configurations. Each benchmark is studied independently. Table 4.4 shows the obtained parameter values.

Table 4.4: Characterization of PARSEC benchmark parallelizability from homogeneous system setup

	A7					
app	$S(2)$	$S(3)$	$S(4)$	p_{A7}		
bodytrack	1.8787	2.6484	3.3211	0.9336	±0.0018	
ferret	1.8833	2.6716	3.3706	0.9381	±0.0004	
fluidanimate	1.5749	—	2.2288	0.7326	±0.0025	
	A15					
app	$S(2)$	$S(3)$	$S(4)$	p_{A15}		α_{15}
bodytrack	1.7980	2.4447	3.0090	0.8881	±0.0021	1.9946
ferret	1.9111	2.7576	3.4400	0.9518	±0.0060	1.8830
fluidanimate	1.4531	—	1.9443	0.6356	±0.0120	1.8186

A7 is once again used as BCE, $\alpha_{A7} = 1$; α_{A15} values are derived from single core executions as the time ratio $t_{A7}(1)/t_{A15}(1)$. Core frequencies of both A7 and A15 are set to 1.4GHz.

Parameter α_s is not known because it is not guaranteed that the sequential part of the workload will be executed on the fastest core, and it is also possible for the sequential execution to be re-scheduled to different core types, however α_s must stay within the range of $\left[\alpha_{A7}, \alpha_{A15}\right]$.

p is determined from the measured speedup $S(n)$ for $n > 1$ solving (3.3) for p. The calculations of p is discussed in chapter 5 with more details. For different values of n, the equation gives different p, however the differences are insignificant within the same type of core. On the other hand, the differences in p for different core types are substantial and cannot be ignored.

The lowest values of the model parameters p and α_s are used to calculate the lower limit of the heterogeneous speedup $S_{low}(\overline{n})$, and the highest values are used to calculate $S_{high}(\overline{n})$.

4.4.2 *Quality of Load Balancer*

Figure 4.11 presents the outcomes of the experiments for the selected benchmarks and heterogeneous core configurations; full data set can be found in Appendix B, Figure B.14. Time measurements have been collected from 10 runs in each configuration to avoid any random flukes, however the results were surprisingly consistent within 0.2% variability. This indicates that the system scheduler and load balancer behave deterministically in given conditions.

The graphs display the calculated speedup ranges $[S_{low}(\overline{n}), S_{high}(\overline{n})]$ and the measured speedup $S(\overline{n})$. The numbers represent the load balancer quality q, calculated from (4.2).

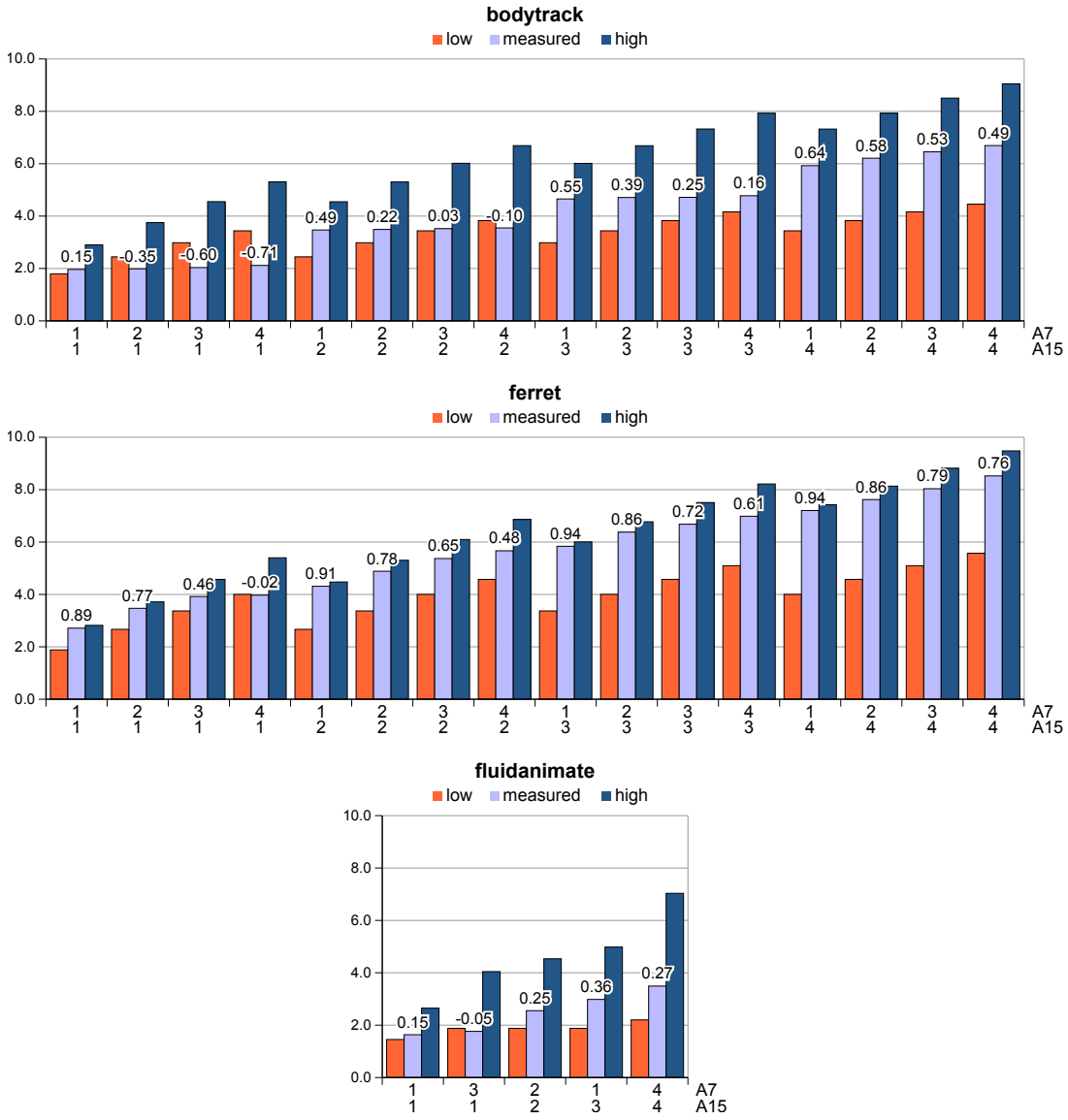


Figure 4.11: PARSEC speedup range results from heterogeneous system setup determining q.

The first interesting observation is that the ferret benchmark is executed with incredible scheduling efficiency despite the system's heterogeneity. The average value of q is 0.81 and the maximum goes to 0.94. According to the benchmark's description, its data parallelism employs a pipeline, i.e. the application implements a producer-consumer paradigm. In this case, the workload distribution is managed by the application. Consequently, the cores are always given work items to execute and the longest possible idling time is less than the execution of one item.

The observed q values never exceed 1, which validates the hypothesis that (3.11) refers to the optimal workload distribution and can be used to predict

the system's performance capacity. The lower bound of q=0 is also mostly respected. This is not a hard limit, but a guideline that separates appropriate workload distributions. This boundary is significantly violated only in one case, as described below.

Bodytrack and fluidanimate show much less efficient workload distribution, compared to ferret, and their efficiency seems to decrease when the core configuration includes more little than big cores. This effect is exceptionally impactful in the case of three A7 cores and one A15 core executing four threads of the bodytrack application. The value of q for this configuration lies far in the negative range and can serve as an evidence of load balancer malfunction. Indeed, the speedup of this four-thread execution is only slightly higher than two-threaded runs on one A7 and one A15. The execution time is close to a single thread executed on one A15 core, showing almost zero benefit from bringing in three more cores, and the result is consistent across multiple runs of the experiment. This issue requires a substantial investigation and lies beyond the scope of this Chapter, however, it demonstrates how the presented method may help analyze the system behavior and detect problems in the scheduler and load balancer.

4.5 DISCUSSION AND CONCLUSION

The practical part of this work includes experiments on multi-type CPU and CPU – GPU systems pertaining to model validation and real-life application. The models have been validated against a synthetic benchmark in a controlled environment. The experiments confirm the accuracy of the models and show that the models provide deeper insights and clearly demonstrate the effects of various system parameters on performance and energy scaling in different heterogeneous configurations.

The modeling method enables the study of the quality of load balancing, used for improving speedup. A quantitative metric for load balancing quality is proposed and a series of experiments involving PARSEC benchmarks are conducted. The modeling method provides quantitative guidelines of load balancing quality against which experimental results can be compared. The Linux load balancer is shown to not always provide high-quality results. In certain situations, it may even produce worse results than the naïve equal-share approach. The study also showed that application-specific load balancing

using pipelines can produce results of much higher quality, approaching the theoretical optimum obtained from the models.

5

SPEEDUP AND PARALLELIZATION MODELS USING PERFORMANCE COUNTERS

5.1 INTRODUCTION

The speedup and power models presented in Chapter 3 and validated in Chapter 4, similar to the classical models, deal with a workload with a p, which is constant throughout the workload. This assumption makes the task of finding simple models clear and straightforward. However, when using these models, it must always be clear what exactly is regarded as a 'workload'.

It is a usual practice that system operations are focused on individual pieces of application software, which are normally executed to satisfy particular requirements of a system's users. Systems also usually execute system software, such as operating systems, which support the running of applications. In this environment, it seems natural to regard manageable parts of system software as well as individual applications as workloads, which would each have its corresponding p value depending on its parallelizability characteristics.

For M/MCP systems, one of the most important issues is operating efficiency, which is related to speedup and energy consumption. To improve efficiency many off-line and RTM methods of system management have been developed, one example discussed in Chapters 3 and 4 is the system load balancer. Typical actuators that M/MCP systems have include DVFS of the cores and the mapping of particular software tasks to particular cores (task to core mapping). Control decisions on these actuator actions can have a very significant relationship with the parallelizability of workloads. For instance, it is intuitive that it would not be very efficient to map a workload with a very low p value onto a large number of cores - most of the cores will be idle most of the time in this case.

It may, therefore, be advantageous to have knowledge of a workload's p value before making a control decision on how to execute it on a M/MCP system. This chapter further develops the models derived in Chapter 3 and validated in Chapter 4, with an aim for improving their usability in this context. The main reason for requiring this further development is that individual applications rarely have a constant time-invariant p value. This motivates

periodic monitoring of the p values of applications. This monitoring has also been made possible by the prevalence of hardware performance counters, some of which will be used in this chapter to support p value monitoring.

The major ***contributions*** of this chapter are:

1. Extend Amdahl's and Gustafson's speedup models considering applications and system software related overhead separately.
2. Propose a new method to model parallelization and speedup via performance counters to avoid the need for instrumenting applications. We show that speedup can be accurately estimated as a ratio of instructions retired/executed per cycle of parallel M/MCP to that of a single core system.
3. Extensive analysis of synthetic and real PARSEC benchmarks to validate the speedup and p based on our proposed model.
4. Generate a novel full-domain DVFS and per-core DVFS power models for M/MCP.
5. Demonstrate the effectiveness of our method for identifying parallelization-aware energy-efficient system configurations using PNP and EDP metrics.

5.2 EXPERIMANTAL STUDIES

5.2.1 *Experimental Platforms*

In this chapter, we make use of three different Intel platforms. Table 5.1 explains the general architecture details of these platforms. All of these systems additionally allow hyper-threading. In all our experiments we disabled hyper-threading by allocating tasks to physical (not logical) cores. Although these systems are from Intel, other modern platforms such as those from ARM also provide similar hardware performance counters which support the generality of this work. Extending this investigation to other platforms will be part of our future work. Certain specialized platforms such as ARM's M-series microcontrollers may not have hardware performance counters. For those kinds of systems, this method may need the construction of software performance monitors, which may or may not be practical.

Table 5.1: Experimental platforms used in this work.

Parameters	Intel CPU Type		
Processor Name	Core i7	Xeon	Xeon Phi
Processor No.	i7-4820k	E5-2630V2	7120X
Lithography	22 nm	22 nm	22 nm
No. of Sockets	1	2	1
Cores per Socket	4	6	61
No. of Cores	4	12	61
L1D Unified Cache	32 KB	32 KB	32 KB
L1I Unified Cache	32 KB	32 KB	32 KB
L2 Unified Cache	256 KB	256 KB	512 KB
L3 Shared Cache	10 MB	15 MB	-
Base Frequency	3.7 MHz	2.60 MHz	1.24 MHz

5.2.2 *Performance Counters*

Hardware performance counters are a set of special purpose registers built into CPUs to store the counts of hardware activities in a specific system [133, 134]. Users depend on those counters to collect low-level performance data for analysis. This performance data varies depending on the performance monitoring hardware and system software configuration [135]. An interface to access platform-specific registers from userspace is provided via the Linux Model-Specific Register (MSR) module. This allows the user to extract hardware performance counter events with an unmodified Linux kernel. Likwid, used in this chapter, is a lightweight performance-oriented tool suite for x86 M/MCP [136].

The following performance counters are used in this chapter.

INSTR_RETIRED_ANY counts the instruction retired which leave the retirement unit. Such instructions have been executed and their results are correct [137].

CPU_CLK_UNHALTED_CORE counts the number of accumulated clock cycles while the core is not in a halt state. This performance counter is obtained through clock cycle recording. If the clock frequency changes the number of cycles will not be proportional to time. And halted states also affect the accuracy of using this performance counter to represent time [138].

CPU_CLK_UNHALTED_REF counts the number of reference clocks at the Time Stamp Counter (TSC) rate, while the core is not in a halt state. This event

is not affected by core frequency changes. It counts at the same frequency as the TSC [138].

5.3 PROPOSED SPEEDUP MODELS

Speedup models usually assume that the workload has a single p. In real systems, the overall workload usually consists of multiple tasks, including system software (e.g. OS), and each task may exhibit a different parallelizability and therefore different p. One potential method of RTM is the parallelizability-aware optimization of performance and/or energy. For that, it is important to treat individual applications and the system software separately. In this section, we develop a new speedup model that calculates application speedup and consider realistic system software overhead separately.

In the rest of this chapter, we deal with the case of a single application running on a real system with system software at the same time. Expanding to multiple concurrently running applications will be a future task.

5.3.1 *Modeling Basics*

We consider that the overall workload is the number of total instructions executed by the system during the execution of any specific application. The total number of instructions for Amdahl's model (I_{Amd}) when a specific application is executed includes the fixed application instructions I plus the system software instructions (ΔI).

$$I_{Amd} = I + \Delta I. \tag{5.1}$$

While the total number of instructions for Gustafson's model (I_{Gus}) when a specific application is executed includes the I' and application instruction is:

$$I_{Gus} = I' + \Delta I. \tag{5.2}$$

If the number of instructions $I + \Delta I$ in (5.1) and $I' + \Delta I$ in (5.2) can be obtained, with a view to calculate speedup we need to find out IPS. In other words, in addition to the number of instructions, we need to know the time spent on executing these instructions, which usually implies instrumenting

both applications and system software for time monitoring. On the other hand, Instructions per Clock (IPC) does not need the monitoring of time and only requires counting the number of clock cycles spent on the execution. In the next section we will explain how to obtain the relevant clock cycle, with which IPC can be calculated for Amdahl's and Gustafson's speedup models as follows:

$$IPC_{Amd} = IPC_I + IPC_{\Delta I} = \frac{I}{\mathcal{C}} + \frac{\Delta I}{\mathcal{C}}, \tag{5.3}$$

$$IPC_{Gus} = IPC_{I'} + IPC_{\Delta I} = \frac{I'}{\mathcal{C}} + \frac{\Delta I}{\mathcal{C}}, \tag{5.4}$$

where number of clock cycle ($\mathcal{C}$) is the number of clock spent on the execution. In a M/MCP system the estimation of effective IPC_{Amd} and IPC_{Gus} for a parallel workload given by (5.3) and (5.4) respectively can be challenging as the instructions retired per core against their corresponding execution cycles cannot be used to estimate an overall average IPC_{Amd} and IPC_{Gus}. This is because some cores execute parallel workloads independent of the other cores, while the core that is in charge of spawning threads executes mostly sequential, but also some parallel workloads. The execution of a workload therefore causes participating cores to record different numbers of clock cycles. We hypothesize that $\mathcal{C}_{max}$ (recorded from the core with the highest unhalted clock $\mathcal{C}$ value among all cores), generally gives a good indication of the overlapped parallel execution times, measured by the time-stamp counter. As such, the effective IPC_{Amd} and IPC_{Gus} in (5.3) and (5.4) can be defined as:

$$IPC_{Amd} = \frac{I}{\mathcal{C}_{Max}} + \frac{\Delta I}{\mathcal{C}_{Max}}, \tag{5.5}$$

$$IPC_{Gus} = \frac{I'}{\mathcal{C}_{Max}} + \frac{\Delta I}{\mathcal{C}_{Max}}, \tag{5.6}$$

Our experiments in Sections (5.2) and (5.4) will show that (5.5) and (5.6) can be used with confidence to model speedup. The resulting throughput expressed by IPS as follows:

$$IPS_I = IPC_I \cdot F, \tag{5.7}$$

$$IPS_{I'} = IPC_{I'} \cdot F, \tag{5.8}$$

where clock frequency (F) F is the system operating frequency. This supports the calculations of sequential and parallel execution time in (3.2) and (3.4).

5.3.2 *Speedup Calculations*

For Amdahl's speedup, we can substitute IPC for IPS as the system frequency is eliminated from the equation in case of the system running at the same frequency, i.e.

$$S_{Amd} = \frac{IPS_{I(n)}}{IPS_I} = \frac{IPC_{I(n)}}{IPC_I}, \tag{5.9}$$

where IPC_I and IPS_I are the instructions per clock and instructions per second, respectively in single core with full sequential workload, and $IPC_{I(n)}$ and $IPS_{I(n)}$ are the instructions per clock and instructions per second, respectively for given p and n configurations of a parallel application on an M/MCP. In other words, speedup is also the ratio of the throughput achieved by executing on n cores to the throughput achieved by executing on a single core.

The Gustafson's speedup is similar:

$$S_{Gus} = \frac{IPS_{I'(n)}}{IPS_I} = \frac{IPC_{I'(n)}}{IPC_I}, \tag{5.10}$$

where $IPC_{I'(n)}$ and $IPS_{I'(n)}$ are the instructions per clock and instructions per second, respectively for given p and n configurations of a parallel application for Gustafson's model on an M/MCP.

The Gustafson's speedup can also be defined as:

$$S_{Gus} = \frac{I'}{I}. \tag{5.11}$$

In our model, we need both the number of instructions and the number of clock cycles for calculating IPC. For the number of instructions, we use INSTR_RETIRED_ANY as application workload I in (5.1) and I' in (5.2). For the number of clock cycles we use CPU_CLK_UNHALTED_CORE as IPC_I in (5.3) and $IPC_{I'}$ in (5.4) which represent the accurate perfromance counter to calculate IPC [138].

In Section 5.3 we showed that the number of cycles represents time. CPU_CLK_UNHALTED_REF shows the number of cycles which includes halted cycles, hence is closer to representing real execution time. In the real world, halted cycles occur when the system has nothing to run. This occurs when threads wait for interrupt thus the counting includes halted cycles in real execution time [139, 140]. However, it is not always available in Intel, as Intel focuses on unhalted clock for IPC calculations [138]. In our experiments, we explore the use of unhalted clock to calculate speedup, and the outcome is presented in Section 5.4.

In addition, hardware performance counters exist that provide power and energy information. For instance, PWR_PKG_ENERGY counts the CPU energy consumption [138]. In previous work, it has been shown that this performance counter produces reliable results validated through direct measurements such as DC instrumentation [141]. It is used in conjunction with the execution time information inferred from unhalted clock performance counters in Section (5.5.1) to derive all power and energy information. In this chapter we focus on CPU energy which changes with p and disregard other energy consumption which has weak correlations with these factors (e.g. memory energy).

From (3.3), it is possible to calculate speedup if $t(1)$ and $t(n)$ can be obtained. However, this requires running a workload at least twice, with different core configurations. To avoid having to run a workload more than once, time-based calculations of p require the knowledge of the sequential and parallel time, which requires instrumenting the code of a workload. By using the performance counters listed above, however, it may be possible to obtain the same functionality as instrumenting separate parallel and sequential time monitoring, while only needing to monitor the start and end of a workload. This means that there is no need to modify workloads in any way.

Even though we motivate our work to avoid time measurements, in our experiments we use time-instrumented workload code when possible as well as make pairs of runs with different numbers of cores. This helps demonstrate the validity of our approach of avoiding direct time measurements through comparisons.

5.3.3 *Estimation of Parallelization Factor*

Once the speedup of an application is known through (5.9) it can be used to calculate Amdahl's parallelization factor (p_{Amd}) for Amdahl's speedup model from (3.3) as:

$$p_{Amd} = \frac{n \cdot (1 - S_{Amd})}{S_{Amd} \cdot (1 - n)}. \tag{5.12}$$

This expression is used in Section (5.4.3) to calculate parallelization using another reason for pulling hardware performance counters forward. Note that the calculation for $n > 1$ gives negative numerator and denominator, thus it gives positive parallelization value.

In the same manner we can calculate Gustafson's parallelization factor (p_{Gus}) for Gustafson's speedup model form (3.6) as:

$$p_{Gus} = \frac{(S_{Gus} - 1)}{(n - 1)}. \tag{5.13}$$

5.3.4 *Average Power Consumption Models*

In this chapter, the power consumption models are established under the hypothesis that the idle power and leakage power are not zero. We use hardware performance counters to calculate total power consumption, the modeling of power consumption in M/MCP will explain with more details in Section 5.5.1.

5.3.5 *Power and Energy Normalized Performance*

PNP is an established metric related to the power efficiency of systems. It is simple to model the performance achievable at the same cooling capacity

by calculating performance per watt ($Perf/Watt$) [21]. PNP can be calculated from dividing the system performance from (5.9) by the W_{total}:

$$PNP_{Amd} = \frac{IPS_{I(n)}}{W_{total}}. \tag{5.14}$$

$$PNP_{Gus} = \frac{IPS_{I'(n)}}{W_{total}}. \tag{5.15}$$

PNP model is the reciprocal of energy per instruction EPI because performance is the reciprocal of execution time [21]. Thus, EPI can be calculated from dividing the W_{total} by the system's performance (5.9):

$$EPI_{Amd} = \frac{W_{total}}{IPS_{I(n)}}. \tag{5.16}$$

$$EPI_{Gus} = \frac{W_{total}}{IPS_{I'(n)}}. \tag{5.17}$$

As metrics, EPI and PNP can be limiting. For instance, if an execution progresses extremely slowly but consumes very little energy, it can result in good EPI and PNP numbers because it consumes almost zero power. On the other hand, it may not get anything useful done. In effect, EPI and PNP promote the minimization of energy but does not care much about performance. To capture this concern the metric known as EDP [142] puts more emphasis on the completion of tasks by explicitly incorporating delay.

$$EDP_{Amd} = W_{total} \cdot (\frac{t(n)}{I})^2. \tag{5.18}$$

$$EDP_{Gus} = W_{total} \cdot (\frac{t(n)}{I'})^2. \tag{5.19}$$

5.3.6 *Benchmark Applications*

We use two methods to validate the extended models, the synthetic and PARSEC benchmarks. We design the synthetic benchmark (described in Section 4.2.2) to experience the extended models with a control in p. Compared with our synthetic benchmark, PARSEC benchmarks are closer to real applications that typically exploit the parallelism provided by M/MCP systems. PARSEC is described in Section 4.4. It consists of 12 applications representing a diverse set of commercial and emerging workloads [131]. In this chapter we choose 9 PARSEC benchmarks having different parralelizabilties and memory usage intensities, [131, 132, 143]. The input set used is "native" (large enough for stable outcomes) and the benchmarks chosen are bodytrack, blackscholes, facesim, fluidanimate, freqmine, swaptions, streamcluster, canneal and dedup.

PARSEC benchmarks are executed at the base frequency on the Core i7 platform only (See Section 5.2.1).

5.4 RESULTS AND VALIDATION

This section describes the model calculations and experimental outcomes. We classify the calculations into fixed workload part I (for Amdahl's model), scaled workload I' (for Gustafson's model) and extra workload ΔI, and demonstrate the validations of execution time and speedup and the estimation of p.

5.4.1 *System Software Instructions Calculation*

In the first stage of our experiments, we use the Core i7 platform to find I, I' and ΔI. The synthetic benchmark application was run on all core configurations from $n = 1$ to $n = 4$ and programmed p ranging from 0 to 1. The first observation was that for all $n = 1$ experiments Core 0 showed exactly the same number of instructions retired with no random variation, which is an indication that all system workloads have been scheduled on idle cores (cores that do not have applications running), and Core 0 has been exclusively running the application workload I, whose size is 5.6E+09. I' is obtained by running the synthetic benchmark as Gustafson's model, the results show that the highest additional workload occurs at high p.

Knowing I, we are able to calculate IPC_I from (5.3) and the speedup based on IPC_I from (5.9) for Amdahl's model. The results are presented in Figure

5.1. Figure 5.1(a) shows the throughput, in IPS, that is achieved with the application's programmed p value ranging from 0 to 1 and the number of cores ranging from 1 to 4. The maximum throughput is clearly achieved with p = 1 and n = 4. It is important to note that with a programmed p of 0 (i.e. non-parallelizable code), increasing the number of cores does not affect the throughput, and with a single core, no matter what the programmed p is the throughput is also constant. Figure 5.1(b) shows the speedup as a function of n and p. It can be seen that the maximum speedup achievable with n = 4 and p = 1 is close to 4, which shows that the synthetic benchmark does not have hidden synchronizations and other effects limiting parallelizability, and the hardware platform's impact on IPC-based speedup is small.

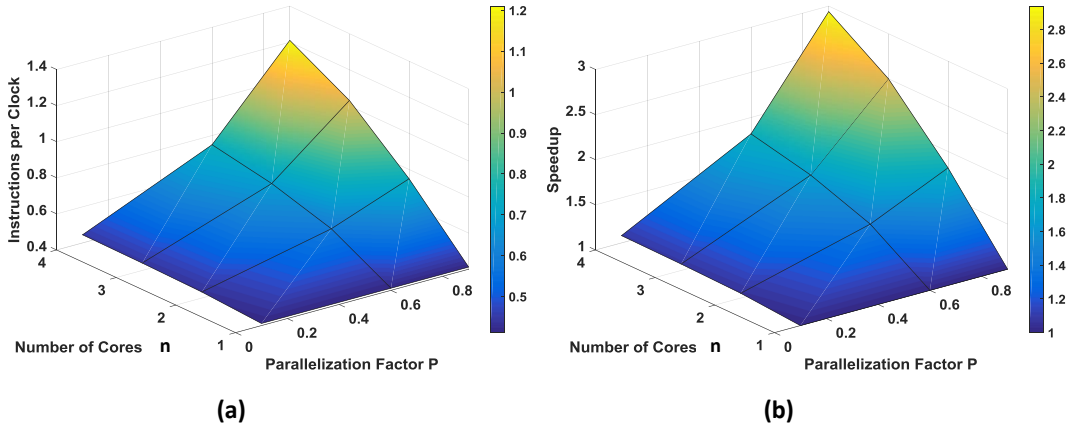


Figure 5.1: Synthetic benchmark using variable n and p for Amdahl's model (a) Application instructions per clock. (b) Performance counter based speedup.

Knowing I′ enables the calculation of $IPC_{I'}$ from (5.4) and the speedup based on $IPC_{I'}$ from (5.10). In addition, the speedup can also be calculated from (5.11). Figure 5.2(a) shows the throughput, in IPS, that is achieved with the application's programmed p value ranging from 0 to 1 and the number of cores ranging from 1 to 4. As expected, different from Amdahl-type workloads Gustafson-type workloads lead to linear speedup, as shown in Figure 5.2.

The second finding is that ΔI reduces with n and p increasing. We tested the hypothesis that system workload ΔI is proportional to time, and confirmed that $\Delta I/t(n)$ approximates to a constant with the average of 6.58E+04 and the standard deviation of 9.53E+03.

Also, the system software workload is very small i.e. 1-2%. However, these extra instructions can cause resource constraints and result in halt cycles. In

our experiments we have observed a 1.55% increase of halt cycles for $p = 0$ and $n = 1$.

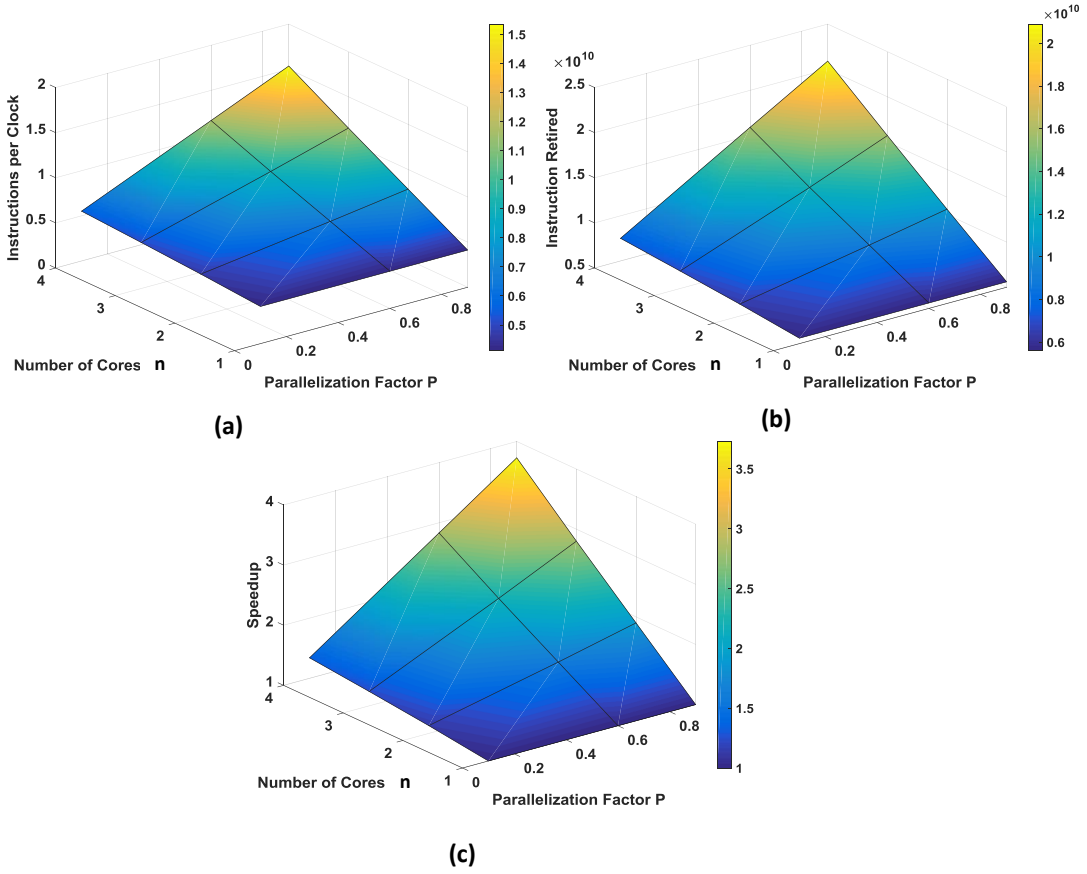


Figure 5.2: Synthetic benchmark using variable n and p for Gustafson's model (a) Application instructions per clock. (b) I'. (c) Performance counter based speedup.

In the second stage, we run PARSEC benchmarks; the applications run in all core configurations from $n = 1$ to $n = 4$. In PARSEC we do not have programmed p. The first observation is that the total instruction retired have fixed values for each application, with small changes < 6%, the total instructions reduced with n increasing and execution time decreasing. Thus, we use linear regression to calculate fixed I and variable ΔI, where ΔI is a function of execution time and number of cores n.

$$\Delta I = \alpha t + \beta n. \tag{5.20}$$

It confirmed that $\Delta I/t(n)$ approximates to a constant in most cases as shown in Table 5.2, where the standard deviations tend to be much smaller than

the averages. The benchmarks bodytrack, facesim, streamcluster, canneal and dedup have a small changes in ΔI rather than blacksholes, freqmine, swaptions and fluidanimate.

Table 5.2: System software workloads over execution time for different PARSEC applications.

Name	Average	Standard Deviation
bodytrack	1.37E+09	4.15E+08
blackscholes	3.92E+08	1.47E+07
facesim	6.78E+08	2.33E+08
fluidanimate	6.66E+08	4.16E+08
freqmine	3.09E+08	4.36E+07
swaptions	3.12E+08	5.86E+07
streamcluster	4.88E+09	1.89E+09
canneal	3.44E+08	1.83E+07
dedup	4.79E+08	1.21E+08

5.4.2 *Time and Speedup Validation*

The results of model calculations and validation measurements of time and speedup for Amdahl's and Gustafson's models are summarized in Tables 5.3 and 5.4. Table 5.3 shows the validation results of the synthetic benchmark of the Amdahl's speedup estimated with performance counters using (5.9), against the traditionally used time measurements. From Table 5.3, two observations can be made. Firstly we validate the use of performance counters by comparing the measured execution time with the time calculated from (3.2) and (5.7) by using the programmed p and the measured IPC and I. We then validate the use of performance counters for Amdahl's speedup estimation by comparing the measured speedup, as the execution time ratio ($t(1)/t(n)$), to the IPC-based speedup calculated from the performance counters according to (5.9) and (5.10). The differences between performance counter results and time measurement results are small, showing that performance counters can be used to replace time measurements, thus avoiding having to run an application twice.

In addition, we validate the use of performance counters for Gustafson's speedup estimation. Table 5.4 shows the validation results of synthetic

benchmark of the Gustafson's speedup estimated with performance counters using (5.10) and (5.11). Table 5.4 again shows small errors.

Table 5.3: Cross-validation results for I using synthetic benchmark [4].

			Time, ms			Speedup		
	p	n	Measured	Calculated	Error %	Measured	Calculated	Error %
Core i7	0.1	2	3492	3492	0.01	1.05	1.052	0.05
	0.1	3	3430	3430	0.03	1.07	1.071	0.03
	0.1	4	3400	3400	0.01	1.08	1.081	0.02
	0.9	1	3675	3676	0.03	1	0.999	0.02
	0.9	3	1470	1570	0.03	2.5	2.501	0.09
	0.9	4	1205	1194	0.86	3.05	3.074	0.84
Xeon	0.1	1	521	534.171	2.47	1	1.000	0.00
	0.1	4	483	494.108	2.25	1.078	1.080	0.36
	0.1	12	474	485.205	2.31	1.099	1.100	0.01
	0.9	2	294	293.794	0.07	1.772	1.666	2.62
	0.9	8	115	113.511	1.31	4.530	3.331	3.83
	0.9	12	95	93.4799	1.63	5.484	3.747	4.11
Xeon Phi	0.1	8	29939	30540.26	1.97	1.118	1.118	2.06
	0.1	16	29744	30331.08	1.94	1.126	1.126	2.09
	0.1	61	30182	30176.76	0.02	1.109	1.108	0.05
	0.9	1	32853	33468.78	1.84	1.019	1.019	1.91
	0.9	4	10655	10877.35	2.04	3.143	3.145	2.23
	0.9	32	4372	4288.18	1.95	7.660	7.848	0.55

Table 5.4: Cross-validation results for fixed time using synthetic benchmark

			Speedup		
	p	n	Theoretical	Calculated	Error %
Core i7	0.1	2	1.1	1.12	2.66
	0.1	3	1.2	1.23	3.31
	0.1	4	1.3	1.34	3.12
	0.9	1	1	1	0.00
	0.9	3	2.8	2.8	0.04
	0.9	4	3.7	3.73	0.82

In these experiments, the errors are generally small; however, they increase to nearly 8% when $p = 1$. This is expected as the programmed p value does not correspond to the real p in the platforms. The observation is that real platforms cannot keep up with the programmed parallelization, presumably due to extra interactions between components.

However, the speedup based on unhalted clock calculation of IPC matches the theoretical speedup from Amdahl's law (3.3) with virtually no error <0.5%. This result can be found in the full set of data and calculations [4]. It indicates that the discrepancy between the measured speedup and the unhalted clock-based speedup is due to halting of the cores. Additionally, this property can be exploited to estimate the application software p value as discussed in Section 5.4.3. Similar discussions can be made for the Gustafson's type workload case.

For PARSEC benchmarks, we run the 9 PARSEC applications on the Intel Core i7 platform at its base frequency of 3.7 GHz. We collect the appropriate hardware performance counters in Section 5.2.2, so that the speedup can be calculated by (5.9). Figure 5.3 shows the speedup calculations for these benchmarks. The performance counter based speedup calculations show good cross-validation with general execution time-based speedup calculations. The full calculations of PARSEC including IPS, power, energy, PNP and EDP can be found in Appendix C. The error ratio does not exceed 6.5%. Finally, The p can be calculated as explained in Section 5.4.3.

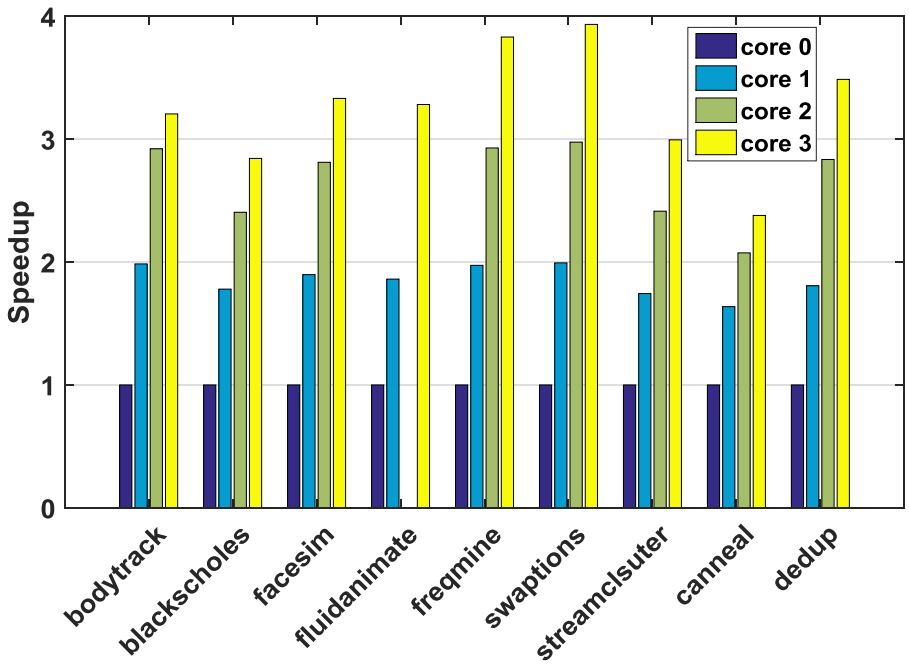


Figure 5.3: Performance counter based speedup for PARSEC benchmark applications.

5.4.3 *Estimating the Parallelization Factor* p

The effectiveness of scaling to more cores in order to obtain more speedup is related to the value of the p (see Section 5.3). In general, from (3.3) and (3.6), scaling to more cores may not improve speedup for a smaller p as much as for a larger p. If it is possible to determine the p value of running any task on any platform, this knowledge may be useful for RTM task to core scheduling. This may be called p-aware RTM.

p can be estimated if the speedup is known. This can be done for any known n through (5.12) and (5.13). It is also possible to determine p using data from experiments based on multiple n configurations through the method of regression, based on such criteria as least squares [144]. Regression has been used for RTM optimization based on learning for M/MCP [145] where the models are unknown. Given (5.12), the motivation of using potentially expensive regression during RTM is weaker here. However, we first need to establish that (5.12) and (5.13) provides the same quality as regression-based methods.

The other question we must consider is the avoidance of instrumenting applications for time. Can we replace time measurements with clock-related performance counter data for p estimation? In this section we attempt to estimate p from speedup derived from both clock performance counters and from direct time measurements, using both regression and (5.12) calculations, and compare the results. These again cover both the synthetic as well as PARSEC benchmarks.

Tables 5.5 and 5.6 show the results for the synthetic benchmark used for Amdahl's and Gustafson's speedup models respectively. Here it is regarded as desirable if the estimated p_{LS} values obtained with least squares regression and p_{EQ} values obtained with (5.12) and (5.13) are closer to the software-programmed p_{SW} set within the benchmark. It can be observed that the differences among regression and (5.12) are small with p_{EQ} tracking p_{LS} closely in both time measurement and clock derived cases. It can also be observed that p_{LSC} (least squares clock-based p) values are very close to p_{LST} (least squares time based p) values and p_{EQC} (equation-clock based p) values are very close to p_{EQT} (equation-time based p) values as shown in table 5.5 for Amdahl's speedup model, meaning that using clock performance counters is valid. And finally, all the estimated p values are very close to the programmed p values set in the benchmark. In other words, this shows that 1) estimating p from speedup is a valid approach and 2) using clock performance counter

data to replace time instrumentation is a valid approach, and 3) our equations produce as good results as the much more expensive linear regression method. Table 5.6 shows these are also true for Gustafson's type workloads.

Table 5.5: p calculations for synthetic benchmark using Amdahl's speedup model [4].

	p_{SW}	p_{LST}	p_{EQT} (5.12)	p_{LSC}	p_{EQC} (5.12)
Core i7	0.1	0.0994	0.0993	0.0999	0.0998
	0.4	0.3990	0.3996	0.3990	0.3999
	0.7	0.6840	0.6834	0.6990	0.6999
	0.9	0.8970	0.8985	0.8820	0.8999
Xeon Phi	0.1	0.0900	0.0892	0.1002	0.1002
	0.3	0.2940	0.2912	0.3002	0.3001
	0.7	0.7000	0.6852	0.7001	0.6999
	0.9	0.8905	0.8849	0.9000	0.9001
Xeon Phi	0.1	0.1008	0.1086	0.1007	0.1087
	0.4	0.4003	0.4007	0.4012	0.4015
	0.5	0.5037	0.5038	0.5038	0.5036
	0.9	0.8892	0.9020	0.9008	0.9029

Table 5.6: p calculations for synthetic benchmark using Gustafson's speedup model.

	p_{SW}	p_{LSC}	p_{EQC} (5.13)
Core i7	0.1	0.099	0.1
	0.4	0.399	0.4
	0.7	0.699	0.7
	0.9	0.889	0.9

Table 5.7 shows the results for PARSEC benchmarks, whose intrinsic p values are unknown and not explicitly set within the programs. They also have more memory access which might introduce unpredictable waiting and synchronization effects making their p values potentially different from run to run. As a result, it is not possible to compare estimated p values to a reference value and the comparison tries to answer two questions: Is it a valid approach to use (5.12) to avoid regression and is it a valid approach to make use of clock performance counter data to avoid instrumenting applications for time monitoring. The results show that the answer is yes for both questions with differences between the approaches generally being very small.

Table 5.7: p calculations of PARSEC benchmarks.

Benchmark	p_{LST}	p_{EQT} (5.12)	p_{LSC}	p_{EQC} (5.12)
bodytrack	0.981	0.925	0.937	0.965
blackscholes	0.841	0.852	0.868	0.872
facesim	0.900	0.920	0.941	0.948
fluidanimate	0.895	0.902	0.927	0.926
freqmine	0.985	0.984	0.985	0.986
swaptions	0.990	0.993	0.994	0.995
streamcluster	0.859	0.858	0.884	0.873
canneal	0.757	0.762	0.774	0.776
dedup	0.940	0.927	0.953	0.938

As a result, we propose to make use of (5.12) and (5.13) directly to estimate p from speedup estimated from clock performance counters in RTM p-aware scaling management.

5.5 PARALLELIZATION-AWARE ENERGY EFFICIENT COMPUTING

As mentioned in Section 5.4.3, if the p value of an application running on a platform is known, RTM decisions may be made based on this knowledge to improve speedup. Beyond just speedup, Shafique *et al* have shown that in dark silicon operations, the p values of workloads need to be considered to arrive at optimal resource allocations through such techniques as dark silicon patterning [117, 115, 116].

In this section, we investigate whether it is possible to optimize energy efficiency with a knowledge of p. For this purpose experiments are carried out to relate metrics of energy efficiency to p. We don't make dark silicon assumptions in this study. Parallelization in dark silicon will be a future topic of research.

5.5.1 *Power and Energy Data*

Energy consumption data is collected from experiments described in the preceding sections using the method described in Section 5.2.2. The energy performance counter PWR_PKG_ENERGY gives total energy consumed by all cores. The study in this chapter does not include *uncore* power calculations

[72] which may be a topic for future work. We calculate the total power consumption of cores W_{total} by dividing energy by the realistic execution time obtained by CPU_CLK_UNHALTED_REF.

In this chapter, we aim to model W_{total} from the collected data by specific hardware performance counters. Basically, we consider that total power W_{total} includes total background power (idle switching power) W_0, total effective power of homogeneous system ($W(n)$) and leakage power (W_l) [146].

$$W_{total} = W_0 + W(n) + W_l, \tag{5.21}$$

In general W_{total} equation can be expressed as [72]:

$$W_{total} = a \cdot C \cdot V^2 \cdot F + W_l, \tag{5.22}$$

where activity factor (a), switch capacitance (C). The W_0 and $W(n)$ are the function of supply voltage (V) and F, while W_l is constant.

In this chapter we establish *full-domain* DVFS power model. In addition, we model *per-core* DVFS [69] later in this section to cover lower power states such as shutting down individual idle cores.

The steps taken to find all models are explained as follows. Firstly, we run a full workload into the Core i7 processor with a wide available range of frequency scaling to collect the voltage readings. We use the Linux-monitoring sensor lm_sensor, which is the Linux tool for monitoring voltage as shown in Table. 5.8.

Secondly, we use (5.22) in Matlab curve fitting to find W_l for Amdahl's and Gustafson's models. We use the collected power data for all available different F, p and n.

The curve fitting for W_l produces good quality results, with best R-squared value $\geqslant 0.994$. The results of W_l is virtually constant for all p, n and F as expected. The average $= 10.349$ with standard deviation $= 0.113$ for Amdahl's speedup, and the average $= 10.14$ with standard deviation $= 0.437$ for Gustafson's model.

Table 5.8: Voltage Frequency Scaling Readings

Frequency - GHz	Voltage - Volt	Power - Watt
3.7	1.1	49.683
3.5	1.07	45.785
3.3	1.05	42.263
3.2	1.04	40.668
3.0	1.02	37.111
2.8	0.99	34.209
2.6	0.97	31.490
2.4	0.94	29.075
2.3	0.93	27.754
2.1	0.91	25.234
1.9	0.89	23.198
1.7	0.86	21.157
1.6	0.85	20.492
1.4	0.82	18.886
1.2	0.81	17.181

The power models of M/MCP are the functions of p, n and F. Thus, finally, we use the experimental data to curve-fit by Matlab and derive the power equations for extended Amdahl's and Gustafson's models for all p, n and F. The curve fitting produces good results with R-squared values = 0.994 and = 0.989 for Amdahl's and Gustafson's models respectively. These models are given by (5.23) and (5.24).

$$W_{Amd} = V^2 \cdot F[A_0 + A_n \cdot (\frac{1}{(1-p) + \frac{p}{n}})] + W_l. \tag{5.23}$$

(5.23) is the Amdahl's total power consumption (W_{Amd}). The product of activity factor by capacitance for idle power consumption (A_0) = 3.506E-09 and the product of activity factor by capacitance for effective power (A_n) = 1.203E-09.

$$W_{Gus} = V^2 \cdot F[A_0 + A_n \cdot ((1-p) + p \cdot n)] + W_l. \tag{5.24}$$

(5.24) gives the Gustafson's total power consumption (W_{Gus}) where A_0 = 3.009E-09 and A_n = 1.337E-09.

Fine grain supply power adjustments, down to per-core DVFS and the turning off of individual cores, have been suggested [69]. Per-core DVFS allowing individual cores to be turned off are not available on our experimental platform but would provide much greater flexibility of control for power and performance optimization. Here we investigate what may happen if per-core DVFS is supported by our experimental platform. The modeling techniques are generally applicable to any future system with per-core DVFS and the on-demand turning off of individual cores.

The power calculations in (5.23) and (5.24) describe W_0, $W(n)$ and W_l individually. Thus, it is possible to model each independently. We calculate the activity factor multiplied to capacitance for per-core idle power as $A_{0i} = A_0/n_{max}$, and the per-core leakage power as $W_{li} = W_l/n_{max}$. n_{max} is the maximum number of cores in the system. Consequently, when only n out of n_{max} cores are turned on, the total power becomes:

$$W_{total} = V^2 \cdot F[\sum_{i=1}^{n} A_{0i} + A_n \cdot (S(n))] + \sum_{i=1}^{n} W_{li} \quad . \tag{5.25}$$

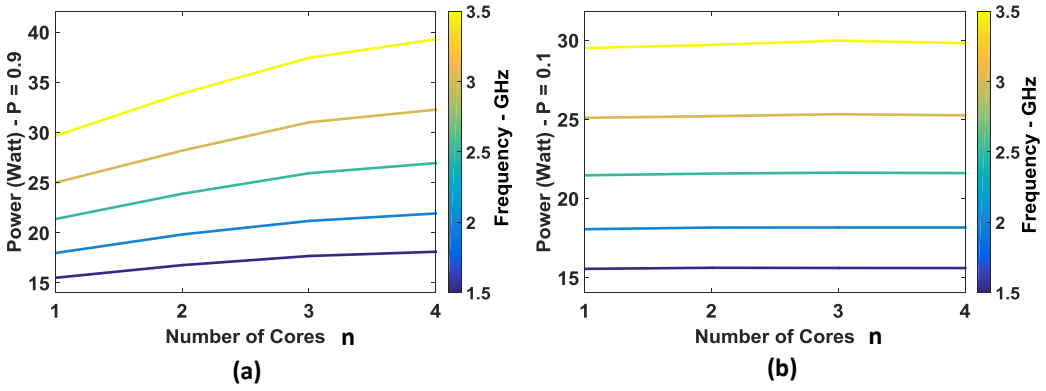


Figure 5.4: Power consumption for synthetic application of extended Amdahl's power model using: a) high p = 0.9, b) low p = 0.1.

Figures 5.4 and 5.5 show the Amdahl's and Gustafson's power consumption respectively in Intel Core i7 (without per-core DVFS) (i.e. idle cores are not turned off) for the synthetic benchmark. Applications that have

high p consume high power in high frequency scaling and maximum numbers of cores as shown in Figures 5.4(a) and 5.5(a) for p = 0.9, whereas applications with low p consume lower power as shown in Figures 5.4(b) and 5.5(b) for p = 0.1. We observe the linear behavior of Gustafson's power model which emphasizes the linearity of Gustafson's speedup model rather than Amdahl's speedup model.

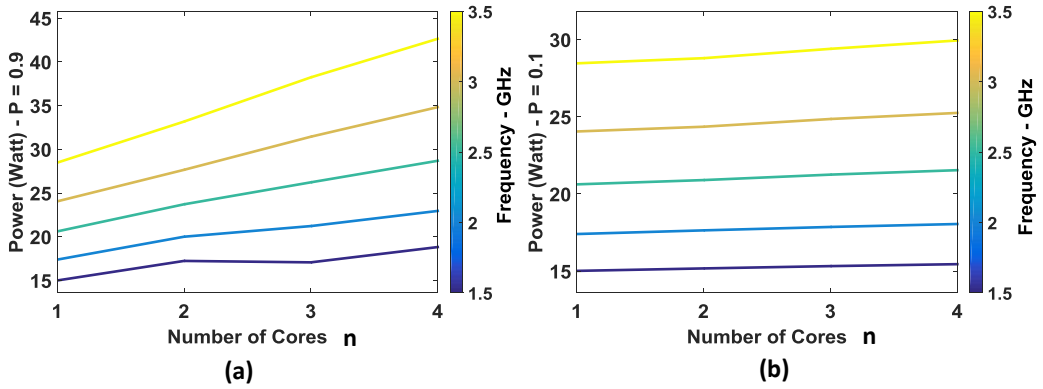


Figure 5.5: Power consumption for synthetic application of extended Gustafson's power model using: a) high p = 0.9, b) low p = 0.1.

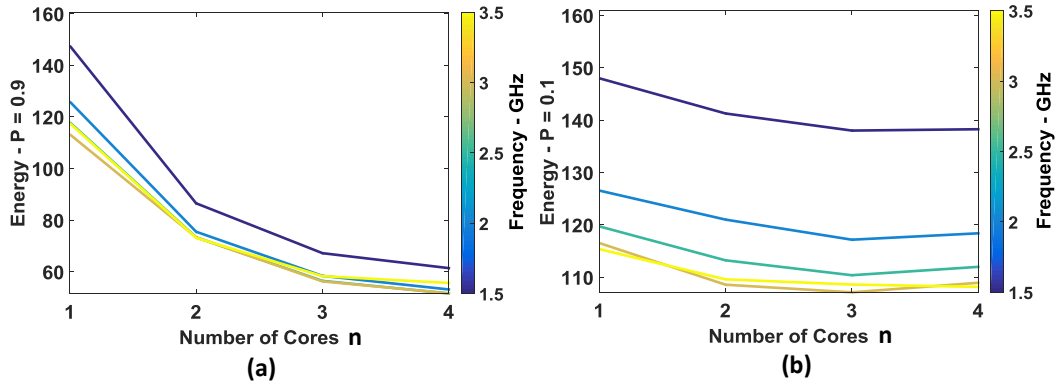


Figure 5.6: Energy consumption for synthetic application of extended Amdahl's energy model using: a) high p = 0.9, b) low p = 0.1.

Figures. 5.6 and 5.7 show the energy consumption of extended Amdahl's and Gustafson's speedup models respectively in Intel Core i7 (without per-core DVFS) for the synthetic benchmark. Applications that have high p in Amdahl's model consume less energy in high frequency scaling and maximum numbers of cores as shown in Figure 5.6(a) for p = 0.9, whereas applications with low p consume higher energy as shown in Figure 5.6(b) for p = 0.1. We observe that for low p the best energy consumption is obtained with 3 - 4 cores.

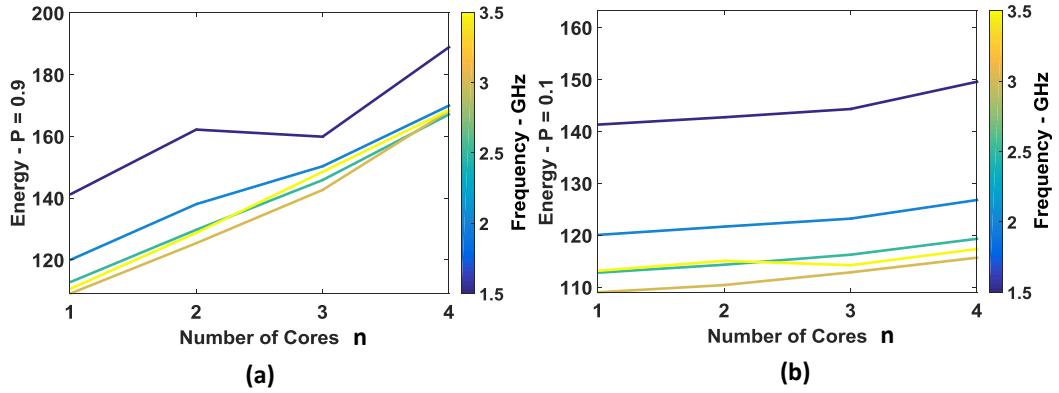


Figure 5.7: Energy consumption for synthetic application of extended Gustafson's energy model using: a) high $p = 0.9$, b) low $p = 0.1$.

Applications that have high p in Gustafson's model consume low energy in 1 core as shown in Figure 5.7(a) for $p = 0.9$, whereas applications with low p consume lower energy in 1 core as shown in Figure 5.7(b) for $p = 0.1$.

These results show that even without per-core DVFS, the p may be significant in a task to core mapping controls when optimizing energy.

5.5.2 *Power Normalized Performance (PNP) and Energy-Delay Product (EDP)*

Both PNP and EDP are metrics for energy efficiency, with different emphasizes, as discussed in Section 5.3.5. Here we calculate PNP from (5.14) and (5.15), while we EDP from (5.18) and (5.19). Figures 5.8 and 5.9 show PNP of the synthetic benchmark for extended Amdahl's and Gustafson's models in full-domain DVFS. The best performance is obtained from maximum number of cores in high and low p. We observe that Gustafson's model achieves highest performance over power in the same system.

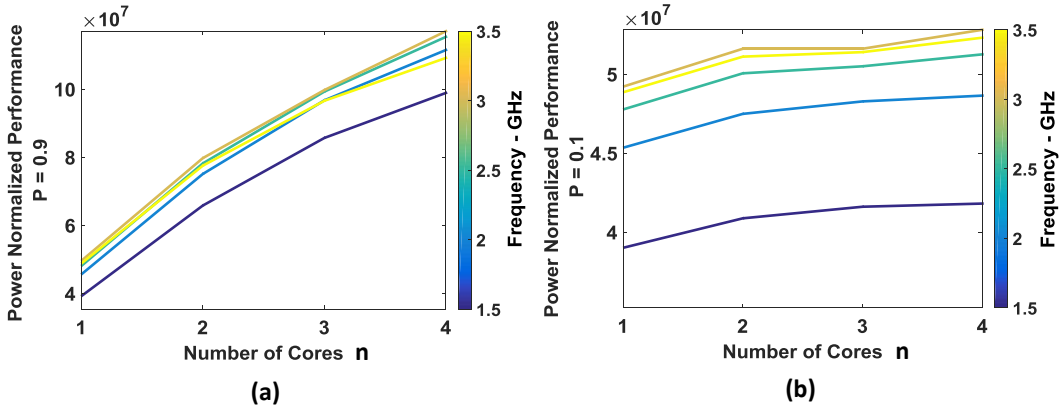


Figure 5.8: PNP results in full-domain DVFS for synthetic application of extended Amdahl's speedup model using: a) high $p = 0.9$, b) low $p = 0.1$.

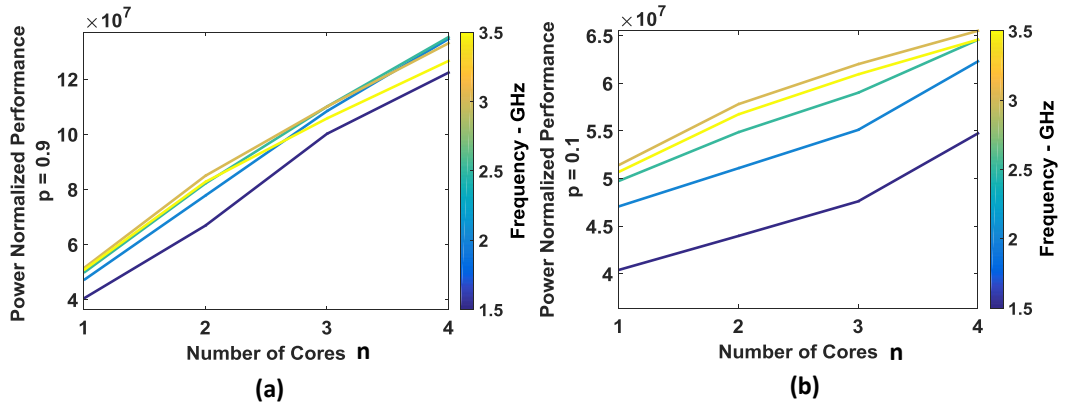


Figure 5.9: PNP results in full-domain DVFS for synthetic application of extended Gustafson's speedup model using: a) high p = 0.9, b) low p = 0.1.

On the other hand, Figures 5.10 and 5.11 show PNP of the synthetic benchmark for extended Amdahl's and Gustafson's models in per-core DVFS.

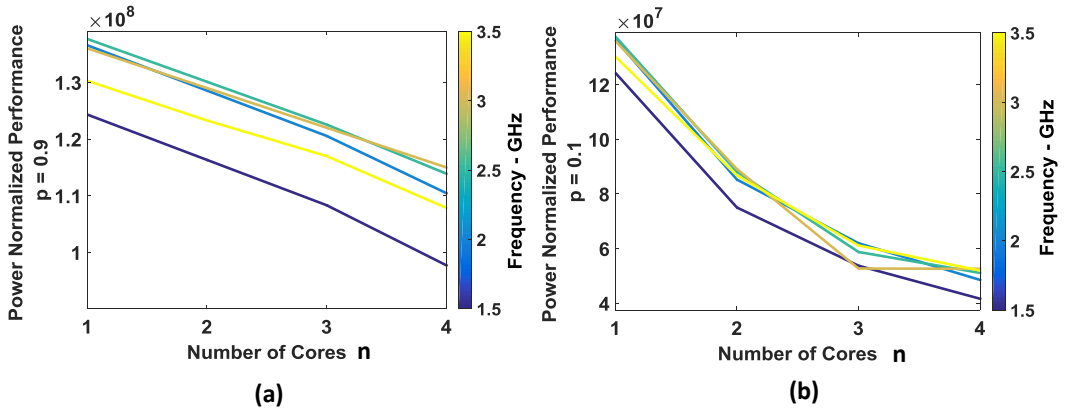


Figure 5.10: PNP results in per-core DVFS for synthetic application of extended Amdahl's speedup model using: a) high p = 0.9, b) low p = 0.1.

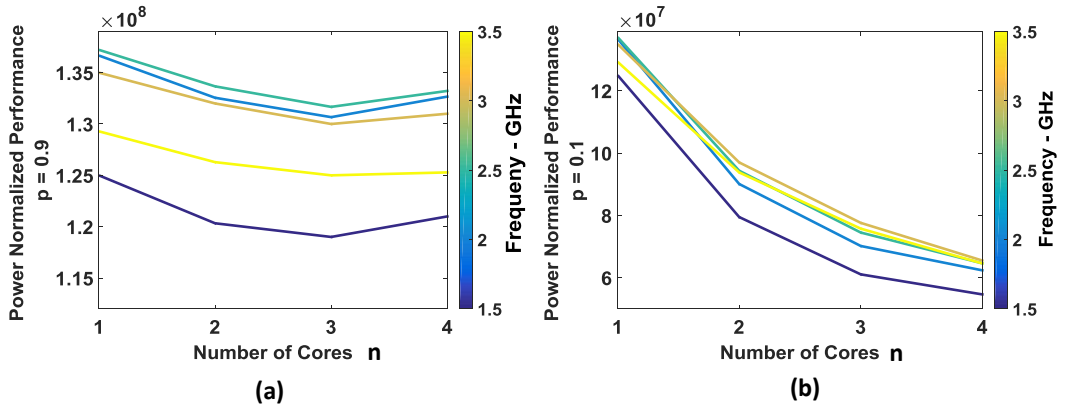


Figure 5.11: PNP results in per-core DVFS for synthetic application of extended Gustafson's speedup model using: a) high p = 0.9, b) low p = 0.1.

In all cases, it is evident from our data that with up to 4 cores, the optimal point of operation, even with per-core DVFS, is to run just a single core. However, with high p for PNP, it can be seen from Figure 5.11(a) that if there are more than 4 cores, it is possible that p-aware optimization may turn out to be more relevant.

Figures 5.12 and 5.13 show the calculation of EDP in full-domain DVFS of synthetic benchmark for Amdahl's and Gustafson's models respectively. For both Amdahl's and Gustafson's models, in high and low p the lowest EDP is obtained from maximum numbers of cores as shown in Figures 5.12 and 5.13.

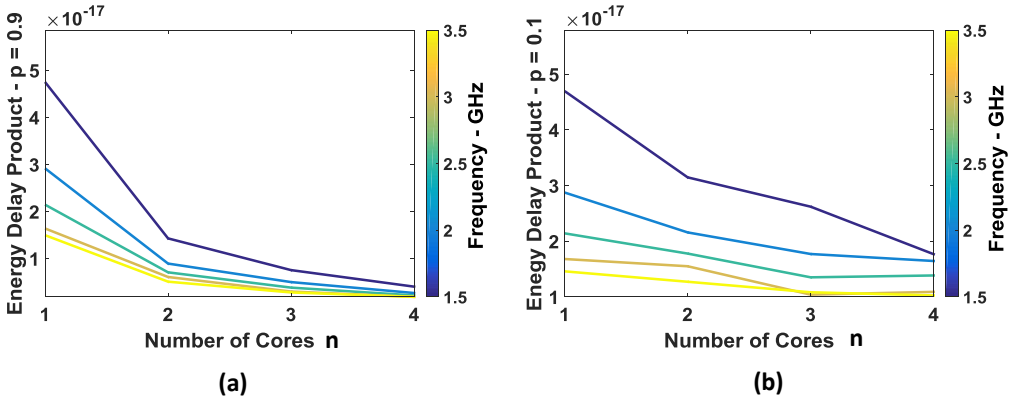


Figure 5.12: EDP results in full-domain DVFS for synthetic application of extended Amdahl's speedup model using: a) high p = 0.9, b) low p = 0.1.

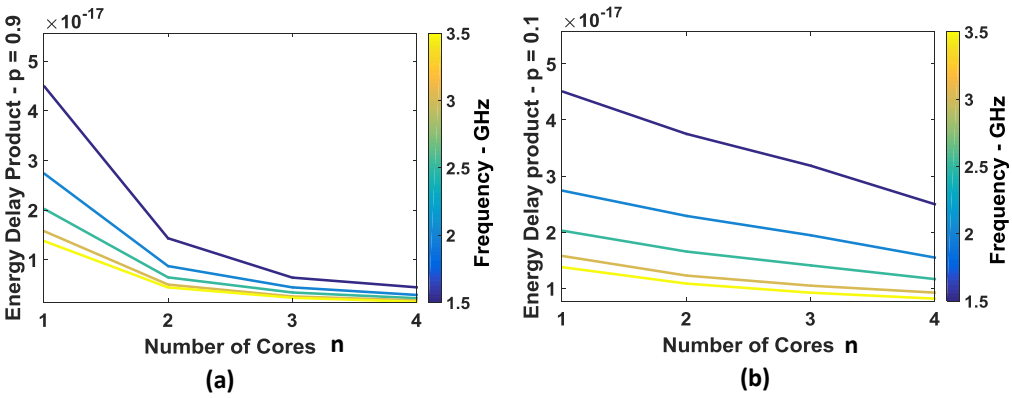


Figure 5.13: EDP results in full-domain DVFS for synthetic application of extended Gustafson's speedup model using: a) high p = 0.9, b) low p = 0.1.

Figures 5.14 and 5.15 show EDP of the synthetic benchmark for extended Amdahl's and Gustafson's models in per-core DVFS.

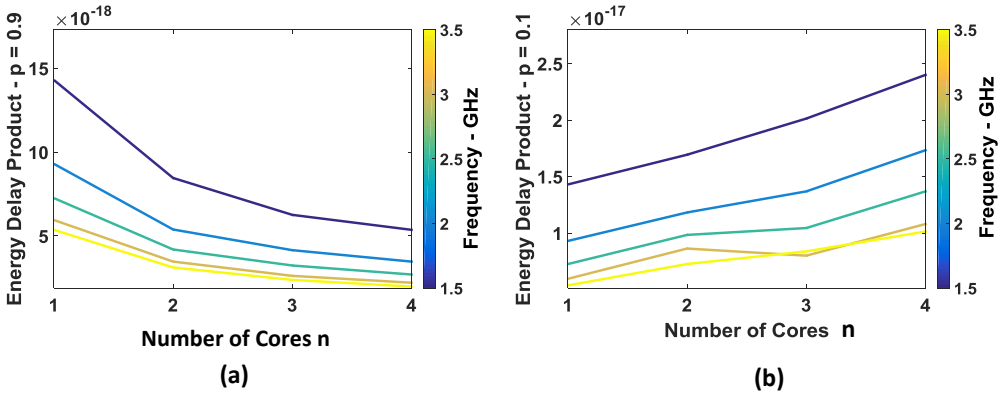


Figure 5.14: EDP results in per-core DVFS for synthetic application of extended Amdahl's speedup model using: a) high p = 0.9, b) low p = 0.1.

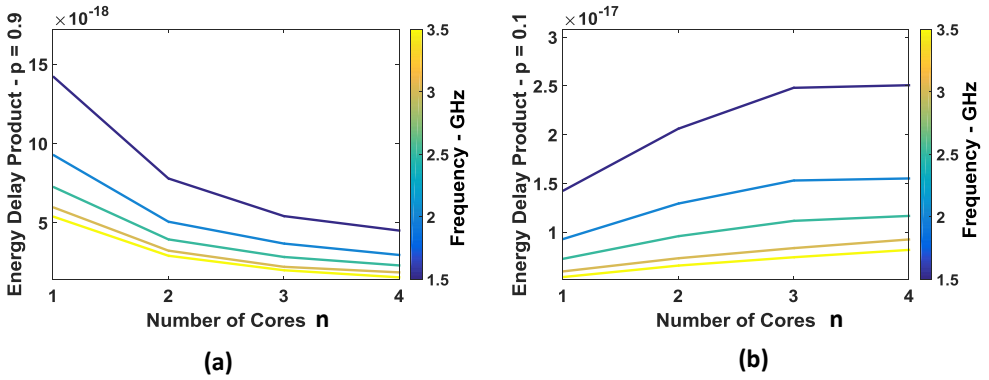


Figure 5.15: EDP results in per-core DVFS for synthetic application of extended Gustafson's speedup model using: a) high p = 0.9, b) low p = 0.1.

Figures 5.14 and 5.15 show that with per-core DVFS running Amdahl's and Gustafson's type workloads, the optimal task to core decision is p-dependent. For different values of p, the best task to core mapping decisions are different. This shows the relevance of p-aware optimization, already with a system having only 4 cores.

The data presented in this section shows that optimal energy efficiency, as measured in either metric, may be a function of p. As a result, the idea of parallelization-aware energy-efficient computing is valid, and we propose to study more examples and develop optimization methods that may be used at RTM as part of our immediate future work.

5.6 CONCLUSIONS AND DISCUSSIONS

This chapter is the first attempt to address the problem of making use of Amdahl's and Gustafson's models without knowing the p and without

instrumenting applications for time monitoring. Performance counters are proposed as a solution to this problem. Speedup can be indicated by IPS data from before and after parallelization rather than directly from time delays. And by using IPC in place of IPS we make it possible to use instruction and clock performance counters for calculating speedup.

In this chapter, we also solve the problem of differentiating application instructions from system software instructions and discover the behavior of typical system software instructions.

Extensive cross-validations have been performed by comparing model-calculated speedup with speedup derived from the measured time, with small errors shown. The maximum error, which rarely occurs, is 8%. We followed the performance counter speedup model to calculate PARSEC benchmark speedup, the outcomes show a sound error of no more than 6.5% related to results derived from measured time.

We also propose a method of determining p once speedup is known, and this is cross-validated by comparing with the programmed p values in our experimental benchmark with very small errors $\leqslant 3.26\%$. Furthermore, the p of PARSEC benchmarks are calculated via the same model.

Based on these parallelization and speedup models we developed models for power, energy, PNP and EDP to explore the energy efficiency of core scaling.

The extended power models of M/MCP in this work achieve good validations with minimum errors $\leqslant 4.1\%$. These models can be used to extended power calculations of EDP and PNP.

We hypothesize that the speedup, parallelization, and EDP models developed in this chapter will give rise to a new method of RTM system control optimizing speedup and/or energy efficiency. This may be called parallelization-aware RTM for performance and/or efficiency. We will focus on this direction in our future work.

6

CONCLUSIONS AND FUTURE WORK

6.1 SUMMARY AND CONCLUSION

The ongoing developments in scaling technology are enabling increases in the number of cores in the same die area. The trends of performance improvements and reduction in power/energy consumption are continuing in M/MCP systems. This leads to some challenges in managing performance-energy tradeoffs for modern and future M/MCP systems. In the past, important ideas were presented to model M/MCP systems by speedup models, such as Amdahl's law which is the most familiar model. It calculates speedup by comparing the execution of a fixed workload in a single core with the same workload in a M/MCP system. Many studies followed Amdahl's law and have developed methods to improve performance calculations such as Gustafson's model which considers the increase in workload in fixed execution time, corresponding to core numbers, and Sun-Ni's model which considers changes in workload size and execution time based on the capability of memory accesses. In addition, the Hill-Marty model developed the assumption of Amdahl's law by considering the performance of a specified area of BCE to cover M/MCP heterogeneity.

The work in this thesis has improved speedup models based on the understanding of the scalability of the M/MCP system by extending the classical Amdahl's law, Gustafson, and Sun-Ni in homogeneous and heterogeneous M/MCP. In addition, a novel method is proposed to model software parallelization factor based on hardware performance counter readings. The presented models extend the principles of Hill-Mary models to cover more general forms of heterogeneity. Furthermore, the presented models are shown to be useful for the study of load balancing quality which is related to speedup. All models derived in this thesis are useful for platform designers, computer, and software engineers, as well as for system level software developers.

The extended models in this thesis can be used for improvements in both software and hardware in relation to M/MCP. In the software field, the programmers can consider several techniques to improve their codes such as

load balancing, task migration, and core affinity. Also, parallelization is the most important parameter that should be considered. In addition, the electronics and computer designer in the hardware field can test different types of homogeneous and heterogeneous core configurations with diverse architectures in order to improve energy efficiency of their designs. In addition, FPGAs can be used as an attractive alternative for implementing computational-rich applications by design a part of hardware to perform the heavy number crunching. Depending on the expected data flow, our models may be extended to cover a greater range of parallelism including pipelining in addition to M/MCP parallelism, leading to being useful for more complex systems with both inter-core and intra-core parallelism.

The work in this thesis can be divided into two main parts, the first of which extends the classical speedup models to apply to the range of heterogeneous system configurations that can be described as the normal forms of heterogeneity. A valid cross-platform performance comparison model using workload measures in workload items instead of instructions is built, and power calculations are implemented in terms of energy per workload item. By moving away from the ISA-specific instructions to ISA-independent workload items, all models are valid and can be applied to different types of heterogeneity such as CPU and GPU. The extended models have been demonstrated using real platforms in comparison to the original models. They include substantial additional elements, such as the distribution of workload between heterogeneous cores and various modes of workload scaling. Moreover, this thesis has addressed the issue of power modeling by calculating power dissipation for the respective homogeneous and heterogeneous speedup models.

The subsequent practical work described in this thesis includes extensive experiments on many different types of CPU and CPU – GPU systems relating to model validation using synthetic and real applications. A synthetic benchmark in a controlled environment has been used to validate the models. The accuracy of the models has been demonstrated by the results of experiments which shows that the models supply deeper insights and clearly explain the effects of different system parameters on performance and energy scaling in various heterogeneous configurations. The heterogeneous Amdahl's speedup validations with equal-share workload present accurate results with low errors (not exceeding 1.13%) in all available core configurations. In addition, the power dissipation outcomes for the same speedup model present accurate result with average absolute error = 1.32%

and do not exceed 5.57% in the worst case. The validations of heterogeneous Amdahl's speedup with a balanced workload show low error $\leqslant$ 1.26%. Furthermore, the power dissipation for the same model present maximum error $\leqslant$ 4.63% and average absolute error $=$ 1.2%. Side by side, the comparison between balanced and equal-share results show a gain in the speedup, but also power increase, EDP can have up to 30% reduction for balanced execution. The speedup validations for Gustafson's model with equal-share workload and classical workload scaling present accurate outcomes with the absolute error not exceeding 0.49%. Power dissipation results for Gustafson's law show low average absolute error $=$ 1.65% with error $=$ 6.23% in the worst case. Furthermore, Gustafson's model with equal-share workload and purely parallel workload scaling give accurate outcomes for validations. The largest absolute error of 3.21% is observed for one case, while the other errors do not go above 1.06%. The average absolute error is 1.96% across all core combinations for power dissipation outcomes for Gustafson's model with equal-share workload and purely parallel workload scaling, and the errors do not exceed 5.21%.

In addition, a series of experiments involving PARSEC benchmarks are conducted and a quantitative metric for the qualities of load balancing is proposed. The modeling method supplies quantitative guidelines for load balancing quality to compare the experimental results. The results show that the Linux load balancer does not always provide high-quality results. It may even in some cases produce worse results than the naïve equal-share approach. This study explains that by using pipelines application-specific load balancing can achieve much better results close to the optimum theoretical results from the models.

In addition, a large number of validation experiments have been made on OpenCL devices. The experiments use the following the number of cores for the integrated GPU (IntGPU): 0, 8, 64, 256. Nvidia GPU is run with the following core numbers: 0, 8, 64, 256, 1024; however, for log the upper limit is reduced to 64 as the higher degree of parallelization is not achievable in Nvidia for this computation. The speedup validation for Amdahl's law with equal-share workload presents the average absolute error $=$ 1.5% across all tested core combinations. The largest absolute error $=$ 9.0% is observed for the case of 256 IntGPU and 64 Nvidia cores. In addition, the speedup validation for Amdahl's law with balanced workload presents the average absolute error $=$ 1.5% across all tested core combinations. The largest absolute error $=$ 9.3% is observed for the case of 256 IntGPU and 64 Nvidia cores. The comparison

between balanced and equal-share results showing the speedup gain in the balanced workload.

The second part of the work in this thesis represents the first attempt to extend the use of Amdahl's and Gustafson's speedup models without instrumenting applications for time monitoring as well as without knowing the parallelization factor p. This part of the study has tackled these problems by using hardware performance counters. This work focuses on extending homogeneous speedup models for performance and energy calculations as well as deriving novel parallelization factor p models. The performance calculations are performed by using an IPS scale which is also used to calculate the speedup before and after parallelization rather than using time delays. Furthermore, the study finds that IPC can be used instead of IPS in order to calculate speedup, and it is possible to use the clock and instruction retired hardware performance counters to calculate speedup. In addition, the problem of differentiating application instructions from system software instructions has been solved, and the behavior of system software instructions has been investigated. Alongside the parallelization and speedup models, a new method of power modeling has been developed based on the speedup models. Also, the extended models for energy, PNP and EDP have been explored to cover the energy efficient computing of many-core scaling.

Extensive sets of experiments have been designed and performed for all methods in the second part of the work. Cross-validations have been performed by comparing the derived speedup based on execution time measurements with the model-calculated. The validation shows small errors, the maximum errors occur rarely at $\leqslant 8\%$. Additionally, the performance counter speedup models have been used in order to calculate PARSEC benchmark speedup, the results show low errors from the measured time of $\leqslant 6.5\%$.

A method of determining the parallelization factor p once speedup is known has been proposed. The cross-validation of this approach has been established by comparing the results using this method with the programmed p values in our synthetic benchmark and the outcomes show small errors of $\leqslant 3.26\%$. In addition, the parallelization factors p of the PARSEC benchmarks are calculated using the same method.

The calculations of the derived theoretical power model provide very accurate results in comparison with the experimental results. The calculations appear to be highly accurate with low error $\leqslant 4.1\%$.

6.2 FUTURE WORK

Different research approaches can be used employing the extended speedup and parallelization models proposed in this thesis. Some of the research directions that can extend the present work described in this thesis are explained in this section.

The speedup and parallelization models as well as PNP and EDP models that have been developed in this thesis can provide guidance to designing parallelization-aware RTM algorithms for the optimization of speedup and energy efficiency based on the parallelization factor p. Additional techniques can be included into the RTM algorithms, such as task migration and load balancing to improve energy efficient system.

The speedup models may be further extended to include calculations of the effect of network access time in a distributed network or NoC.

The models proposed in this thesis can be modified to include memory wall effects, particularly for a single and concurrent application which can be classified based on memory usage for different types of application management.

One important motivation for such more precise modeling into parts of applications is using the models in RTM towards the optimization of goals related to performance and/or power dissipation. It is more typical for RTM control decisions to be made at regular intervals of time, unrelated to the start and completion of whole applications, hence the importance of phases within each application.

The parallelization factor models in this thesis can help electronics and computer designers to manage dark silicon calculations in M/MCP. For instance through task partition and mapping decisions. Goals of such management and control may include the reduction of power and managing the trade-offs between thermal design power diversity and reliability.

Part II

Thesis Appendices

A

BENCHMARK APPLICATION

A.1 SYNTHETIC BENCHMARK

```
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <math.h> // compile with -lm
#include <pthread.h> // compile with -lpthread
#include <time.h> // compile with -lrt
struct thread_info {
        pthread_t thread_id;
        int thread_num;
};
int verbose = 0;
struct thread_info *tinfo;
static void* thread_start(void *arg) {
        int n = *(int*) arg;
        double x = 0.0;
        long i, j;
        for(j=0; j<n; j++)
                for(i=0; i<4000L; i++) {
                        x = sqrt(3.0*x+4.0);
                }
        return NULL;
}
unsigned long print_time(unsigned long prev_tstamp, const char* prompt)
  {
        struct timespec t;
        unsigned long tstamp;
        clock_gettime(CLOCK_REALTIME, &t);
        tstamp = (t.tv_sec)*1000L + (t.tv_nsec)/1000000L;
        printf("%s:\t%lu\n", prompt, tstamp - prev_tstamp);
```

```
        return tstamp;
}
void parallelize(int num_threads, char *affinities, void *(*start_
   routine) (void *), void *arg) {
        int tnum, err;
        pthread_attr_t attr;
        cpu_set_t cpus;
        pthread_attr_init(&attr);
        for(tnum = 0; tnum < num_threads; tnum++) {
                tinfo[tnum].thread_num = tnum + 1;
        // set affinity attribute
                CPU_ZERO(&cpus);
                CPU_SET((int)(affinities[tnum]-48), &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(&tinfo[tnum].thread_id, &attr,
                   start_routine, arg);
                if(err)
                        printf("*error(%d): pthread_create, %d\n", __
                           LINE__, err);
        }
        // wait for all threads to finish
        for(tnum = 0; tnum < num_threads; tnum++) {
                pthread_join(tinfo[tnum].thread_id, NULL);
        }
}
int main(int argc, char *argv[])
{
        pthread_t t0;
        cpu_set_t cpus;
        int num_threads, err, opt;
        int npar, nseq, r;
        int repeat = 5;
        int ntotal = 10000;
        float p = 0.5;
```

```
float adjust = 1.0; // parallelization overread adjustment, 1 =
   no adjustment
char* affinities = "01";
int t0_affinity = 0;
unsigned long prev_tstamp = 0L;
unsigned long start_tstamp;
// parse arguments
while((opt = getopt(argc, argv, "vj:p:w:r:z:c:")) != -1) {
        switch (opt) {
                case 'v':
                        verbose = 1;
                        break;
                case 'p':
                        p = atof(optarg);
                        break;
                case 'j':
                        adjust = atof(optarg);
                        break;
                case 'w':
                        ntotal = atoi(optarg);
                        break;
                case 'r':
                        repeat = atoi(optarg);
                        break;
                case 'z':
                        t0_affinity = atoi(optarg);
                        break;
                case 'c':
                        affinities = malloc(strlen(optarg));
                        strcpy(affinities, optarg);
                        break;
                default:
                        fprintf(stderr, "Usage: %s [-v] [-p
                           parallel_ratio] [-j adjust] [-w
                           workload] [-r repeat] [-z parent_
                           affinity] [-c child_affinities]\n",
                           argv[0]);
                        exit(EXIT_FAILURE);
        }
}
```

```
        // init
        npar = (int)(ntotal*p*adjust);
        nseq = (int)(ntotal*(1.0-p));
        num_threads = strlen(affinities);
        printf("%d threads...\n", num_threads);
        tinfo = calloc(num_threads, sizeof(struct thread_info));
        // set thread0 affinity, also used for sequential operations
        t0 = pthread_self();
        CPU_ZERO(&cpus);
        CPU_SET(t0_affinity, &cpus);
        err = pthread_setaffinity_np(t0, sizeof(cpu_set_t), &cpus);
        if(err)
                printf("*error(%d): pthread_getaffinity_np, %d\n", __
                  LINE__, err);
                prev_tstamp = print_time(prev_tstamp, "Start");
        start_tstamp = prev_tstamp;
                for(r=0; r<repeat; r++) {
        // execute parallel workload
                parallelize(num_threads, affinities, &thread_start, &
                  npar);
                if(verbose)
                        prev_tstamp = print_time(prev_tstamp, "Parallel
                          ");
        // execute sequential workload
                thread_start(&nseq);
                if(verbose)
                        prev_tstamp = print_time(prev_tstamp, "
                          Sequential");
        }
        print_time(start_tstamp, "Total");
        return 0;
}
```

B

DATA SET

This appendix provides a full set of validation data for the Odroid XU3 platform (Figures B.1–B.10) and OpenCL results obtained from Dell XPS 15 laptop (Figures B.11–B.13).

B.1 ODROID XU3

Figure B.1 presents the speedup validation for Amdahl's law with equal-share workload. The largest absolute error of 1.13% is observed for the case of 3 A7 and 4 A15 cores running log with $p = 0.9$. The average absolute error is 0.20% across all core combinations. Power dissipation results are for Amdahl's law are shown in Figure B.2. The largest absolute error of 5.57% is observed for the case of 3 A7 and 4 A15 cores running int with $p = 0.9$. The average absolute error is 1.32% across all core combinations.

Figure B.3 presents the speedup validation for Amdahl's law with balanced workload. The largest absolute error of 1.26% is also observed for the case of 3 A7 and 4 A15 cores running log with $p = 0.9$. The average absolute error is 0.19% across all core combinations. Power dissipation results are for balanced Amdahl's law are shown in Figure B.4. The largest absolute error of 4.63% is observed for the case of 2 A7 and 4 A15 cores running log with $p = 0.9$. The average absolute error is 1.20% across all core combinations.

Figure B.9 displays balanced and equal-share results side by side with the numbers showing the speedup gain, but also power increase ($p = 0.9$). Energy-delay product can have up to 30% reduction for balanced execution.

Figure B.5 presents the speedup validation for Gustafson's model with equal-share workload and classical workload scaling. The largest absolute error is 0.49%, and there are multiple points with similar accuracy. The average absolute error is 0.10% across all core combinations. Power dissipation results are for Gustafson's law are shown in Figure B.6. The largest absolute error of 6.23% is observed for the case of 3 A7 and 4 A15 cores running sqrt with $p = 0.9$. The average absolute error is 1.65% across all core combinations.

Figure B.7 presents the speedup validation for Gustafson's model with equal-share workload and purely parallel workload scaling. The largest absolute

error of 3.21% is observed for the case of 3 A7 and 4 A15 cores running sqrt with $p = 0.3$. The other heterogeneous points of sqrt with $p = 0.3$ also show increased errors (above 1%). Excluding sqrt benchmark, the error does not go above 1.06%, and the average absolute error is 0.33% across remaining core combinations. Power dissipation results are for parallel-scaled Gustafson's law are shown in Figure B.8. The largest absolute error of 5.21% is observed for the case of 0 A7 and 4 A15 cores running log with $p = 0.9$. The average absolute error is 1.96% across all core combinations.

Figure B.10 displays two Gustafson's workload scaling modes side by side with the numbers showing the speedup gain ($p = 0.3$), but also demonstrates how improper use may cause poor performance, as seen in the sqrt example using A15 for the sequential execution, which is slower than A7 for this function.

B.2 OPENCL

It would be very impractical to test all core combinations for the available OpenCL devices. In the following experiments we use the following numbers of cores for the integrated GPU (IntGPU): 0, 8, 64, 256. Nvidia GPU is ran with the following core numbers: 0, 8, 64, 256, 1024; however, for log the upper limit is reduced to 64 as the higher degree of parallelization is not achievable in Nvidia for this computation. Parameter s denotes the core type used in sequential workload execution. In this section, the speedup is shown on a logarithmic scale.

Figure B.11 presents the speedup validation for Amdahl's law with equal-share workload. The largest absolute error of 9.0% is observed for the case of 256 IntGPU and 64 Nvidia parallelization when running int with $p = 0.3$. The average absolute error is 1.5% across all tested core combinations.

Figure B.12 presents the speedup validation for Amdahl's law with balanced workload. The largest absolute error of 9.3% is observed for the case of 256 IntGPU and 64 Nvidia parallelization when running int with $p = 0.9$. The average absolute error is 1.5% across all tested core combinations.

Figure B.13 displays balanced and equal-share results side by side with the numbers showing the speedup gain.

B.3 PARSEC

The experimental results for PARSEC benchmarks are shown in Figure B.14. Only heterogeneous core combinations are shown as the homogeneous configurations have been used for model characterization. For fluidanimate, the number of configurations is limited as the benchmark can only run a power-of-two number of threads.

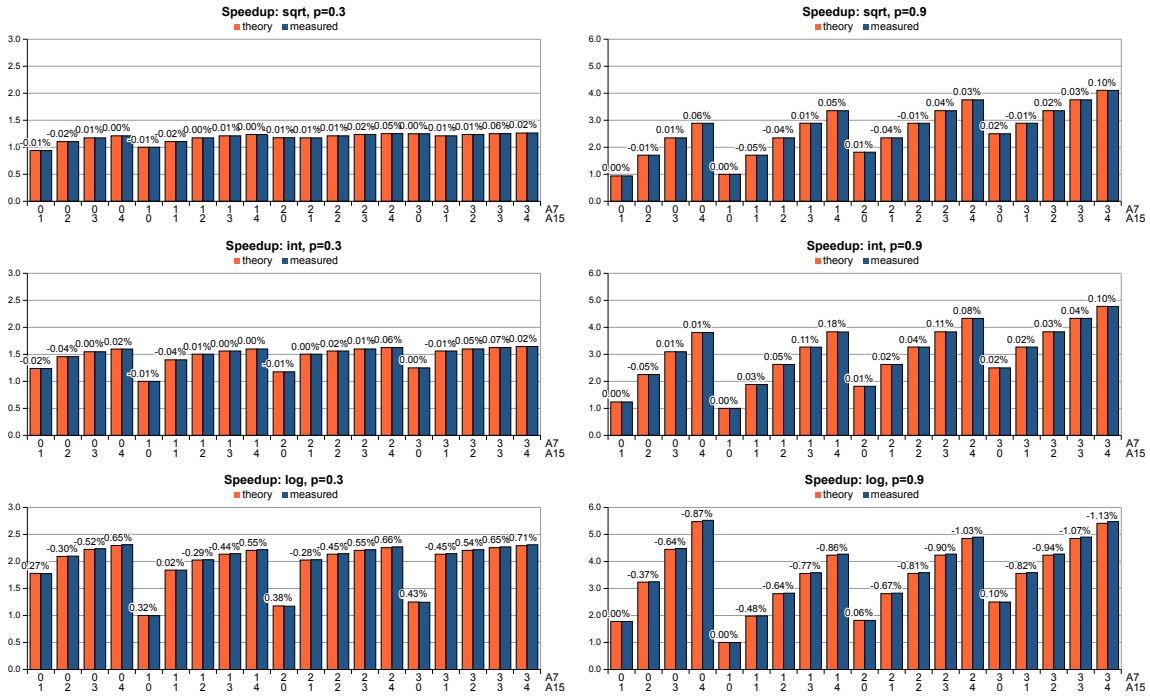


Figure B.1: Speedup validation results for the heterogeneous Amdahl's law showing percentage error of the theoretical model in relation to the measured speedup.

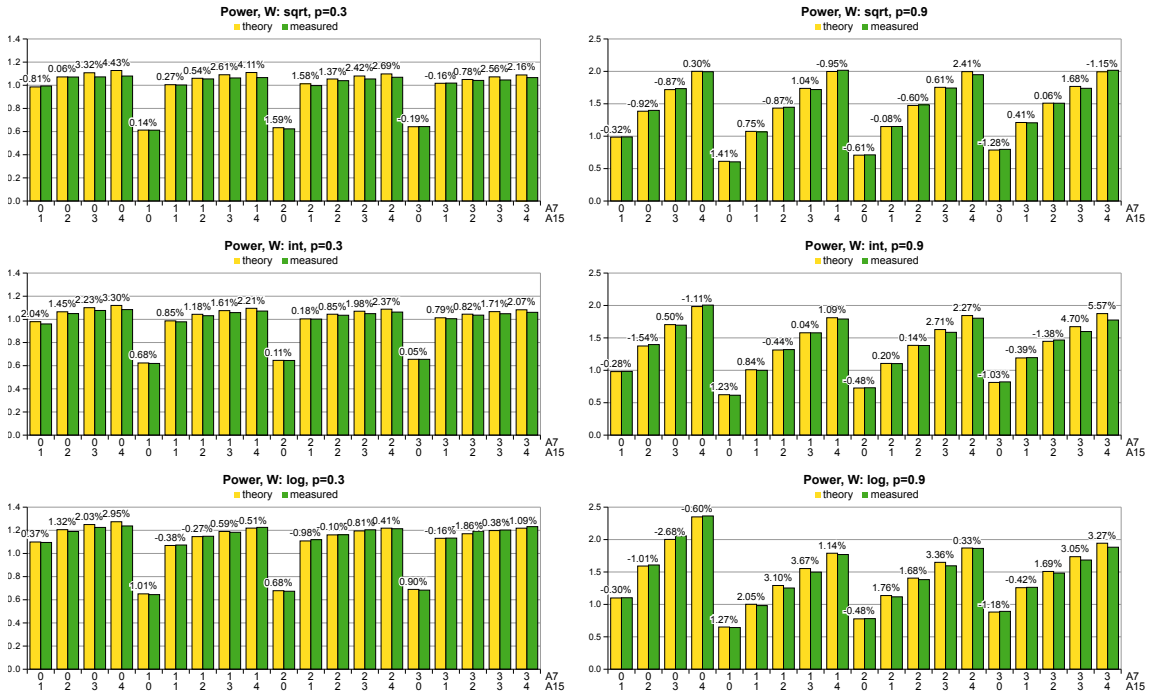


Figure B.2: Total power dissipation results for the heterogeneous Amdahl's law showing percentage error of the theoretical model in relation to the measured speedup.

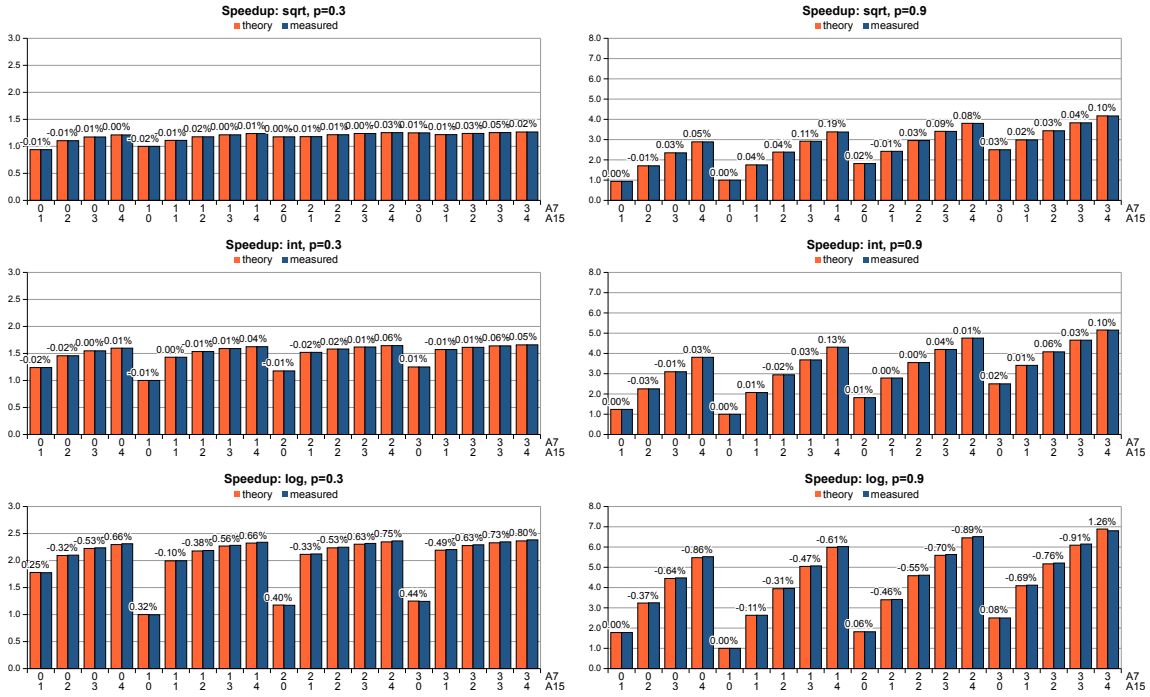


Figure B.3: Speedup validation results for the heterogeneous Amdahl's law with balanced workload showing percentage error of the theoretical model in relation to the measured speedup.

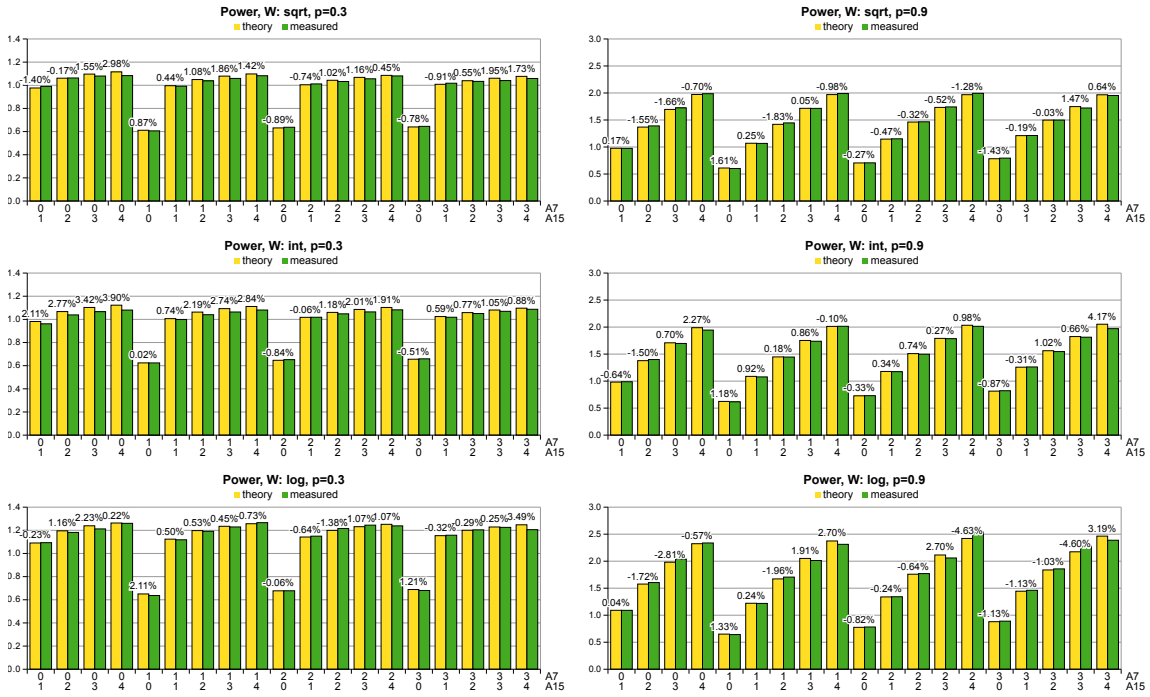


Figure B.4: Total power dissipation results for the heterogeneous Amdahl's law with balanced workload showing percentage error of the theoretical model in relation to the measured speedup.

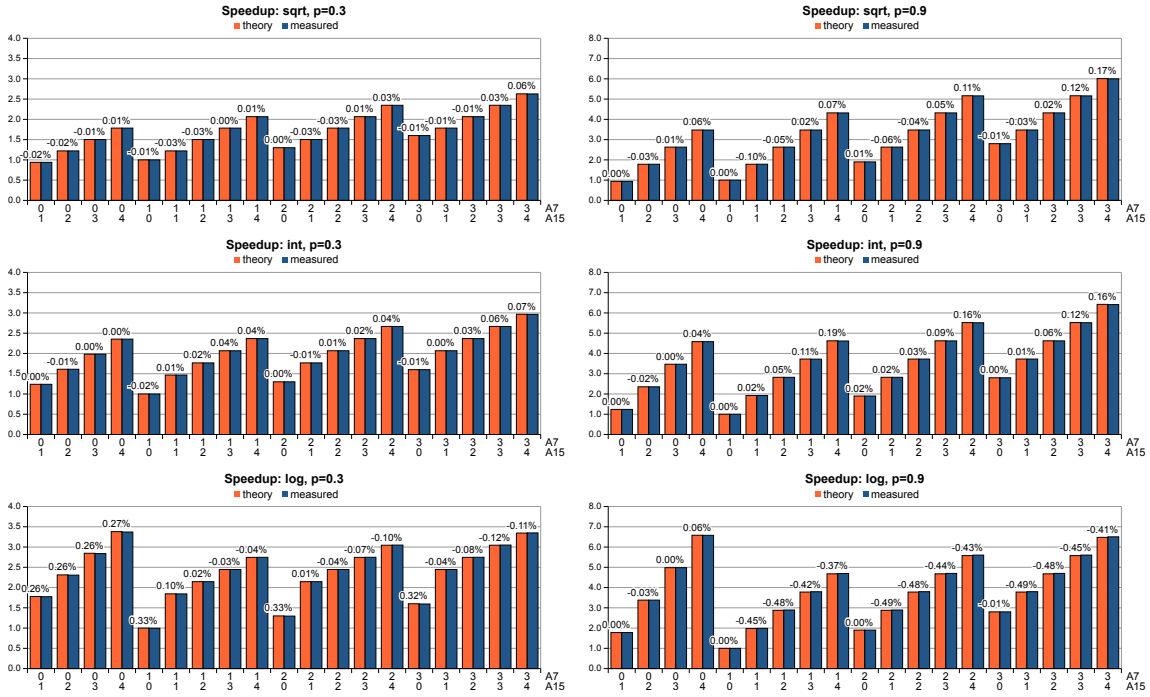


Figure B.5: Speedup validation results for the heterogeneous Gustafson's model with classical scaling showing percentage error of the theoretical model in relation to the measured speedup.

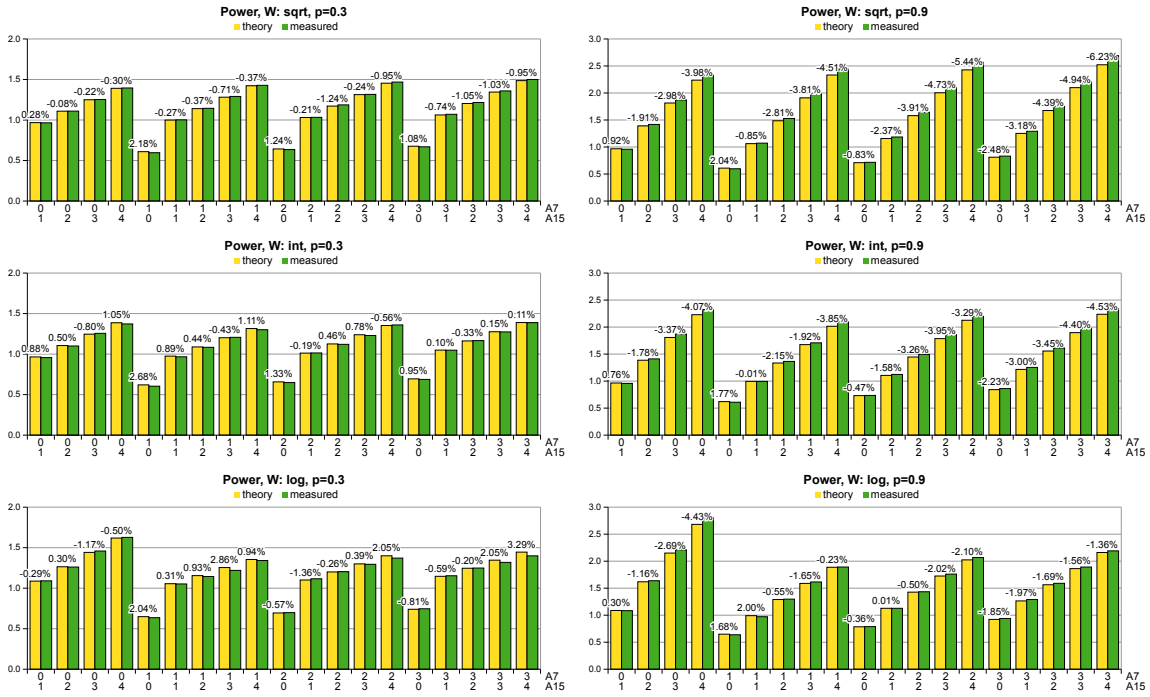


Figure B.6: Total power dissipation results for the heterogeneous Gustafson's model with classical scaling showing percentage error of the theoretical model in relation to the measured speedup.

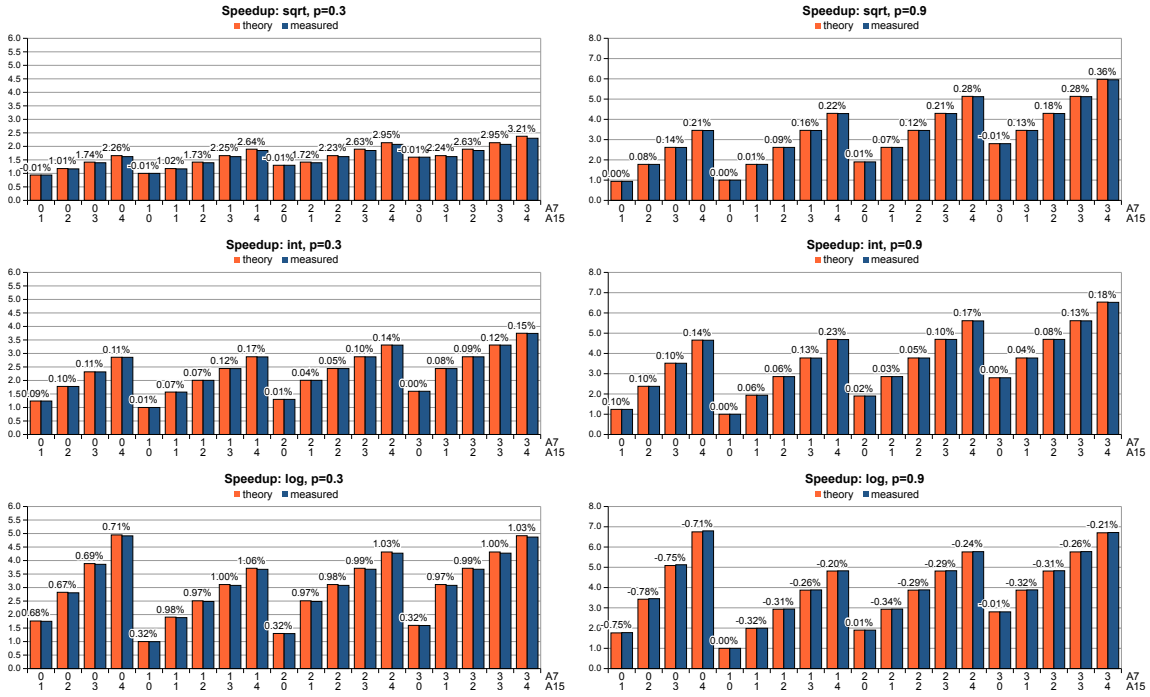


Figure B.7: Speedup validation results for the heterogeneous Gustafson's model with purely parallel scaling showing percentage error of the theoretical model in relation to the measured speedup.

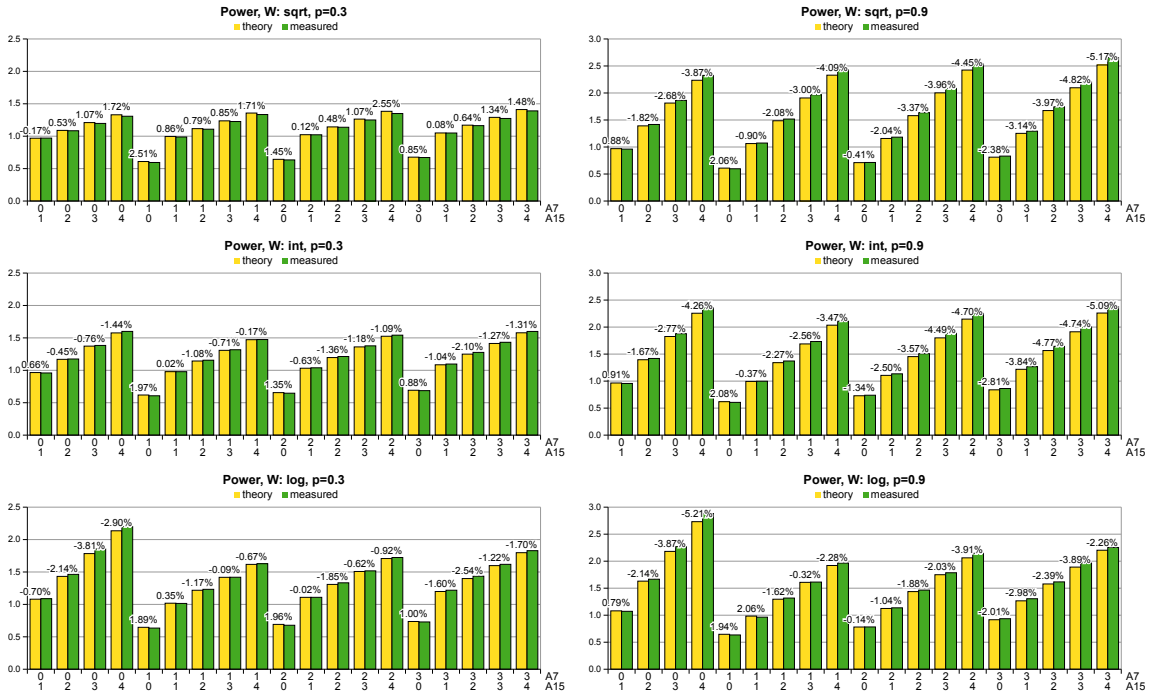


Figure B.8: Total power dissipation results for the heterogeneous Gustafson's model with purely parallel scaling showing percentage error of the theoretical model in relation to the measured speedup.

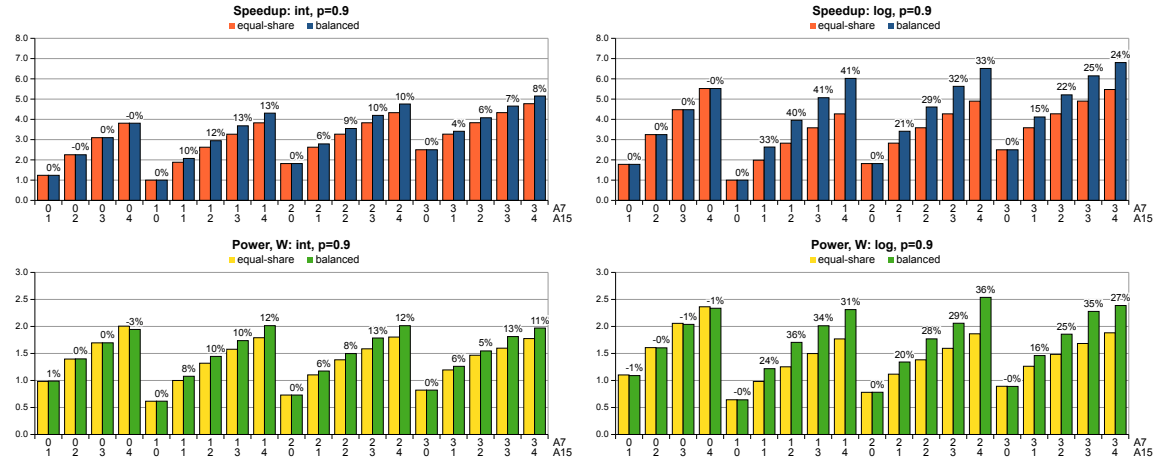


Figure B.9: Comparison of the measured speedup, power and energy between equal-share and balanced execution.

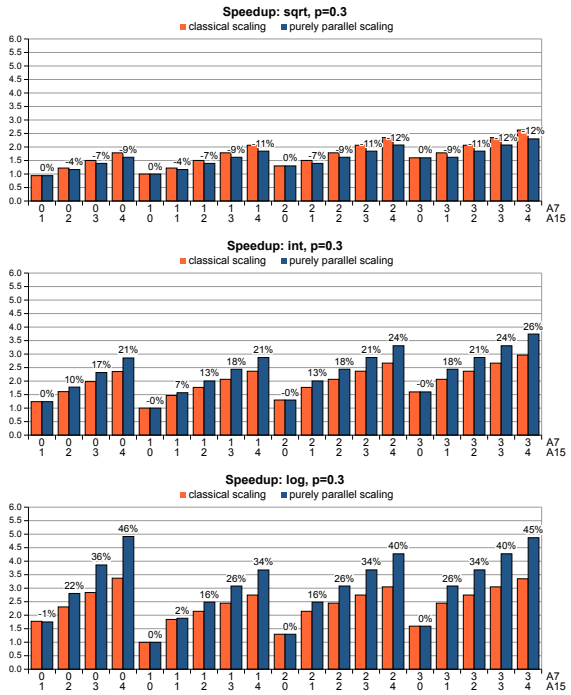


Figure B.10: Gustafson's model outcomes showing the measured speedup gain from using the purely parallel workload scaling compared to the classical scaling.

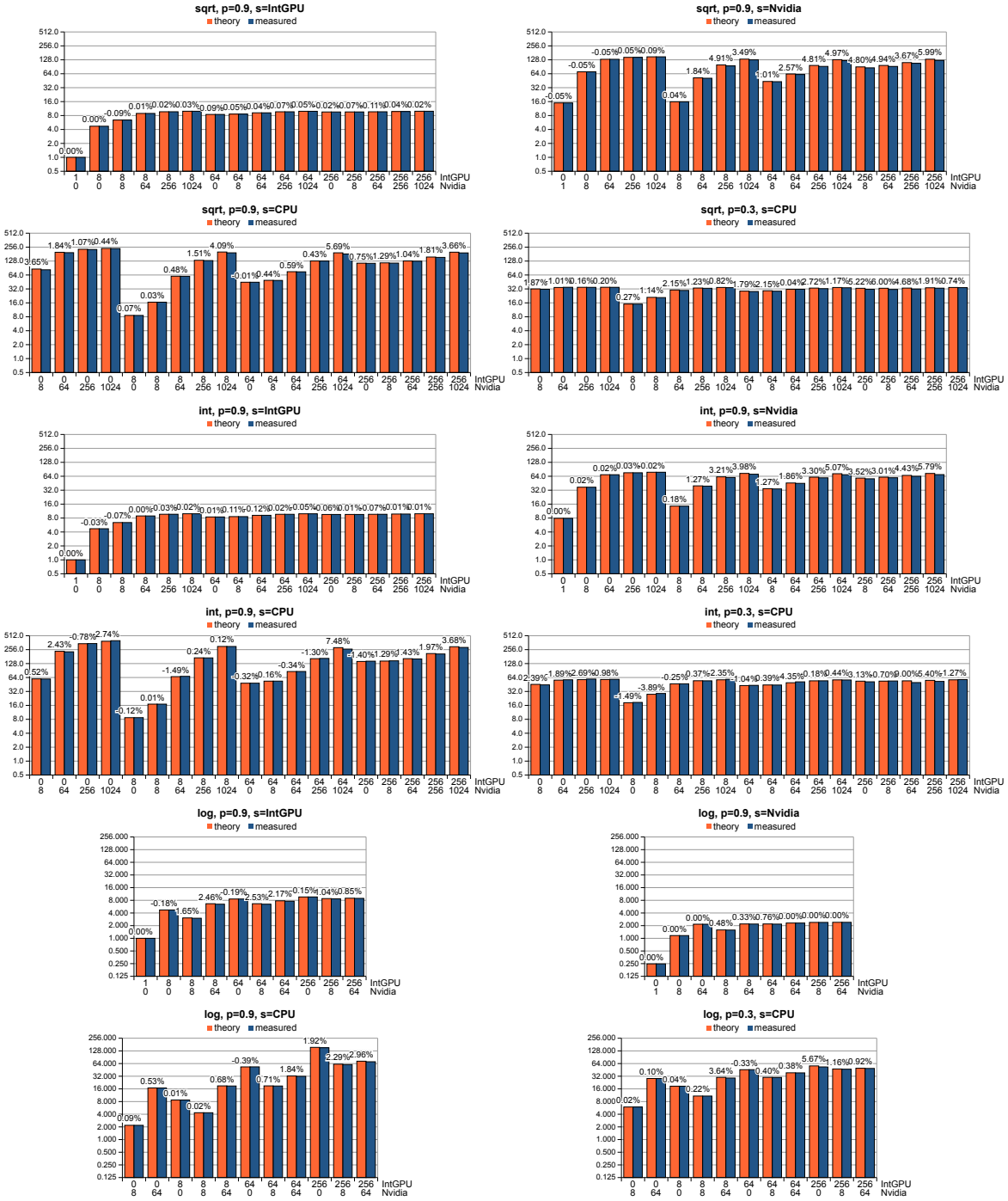


Figure B.11: OpenCL speedup validation results for the heterogeneous Amdahl's law showing percentage error of the theoretical model in relation to the measured speedup.

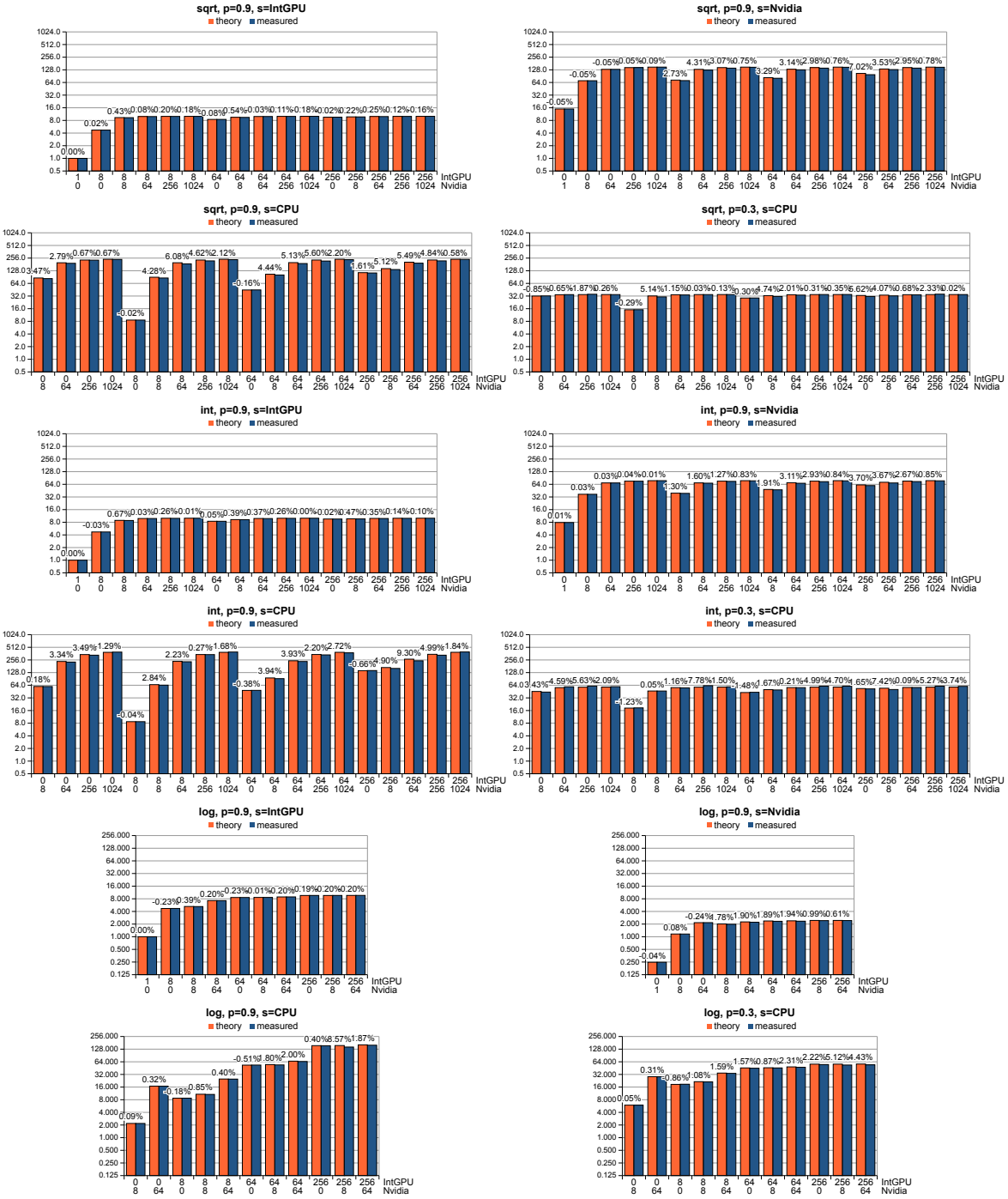


Figure B.12: OpenCL speedup validation results for the heterogeneous Amdahl's law with balanced workload showing percentage error of the theoretical model in relation to the measured speedup.

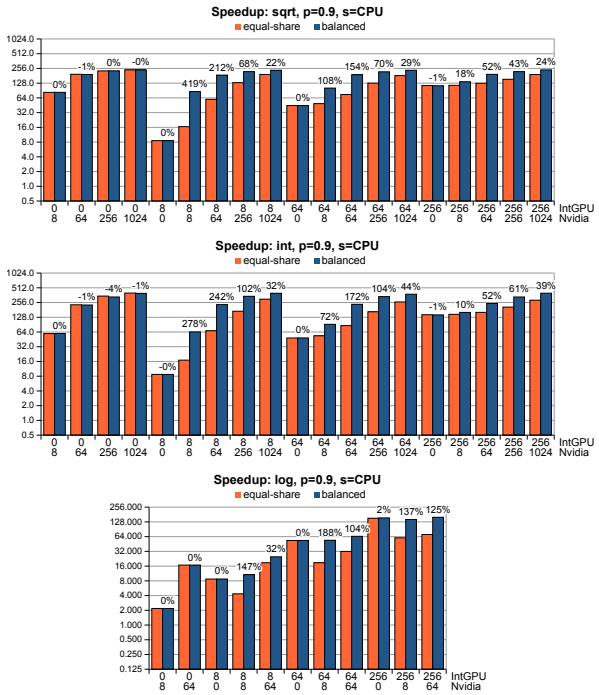


Figure B.13: Comparison of the measured OpenCL speedup between equal-share and balanced execution.

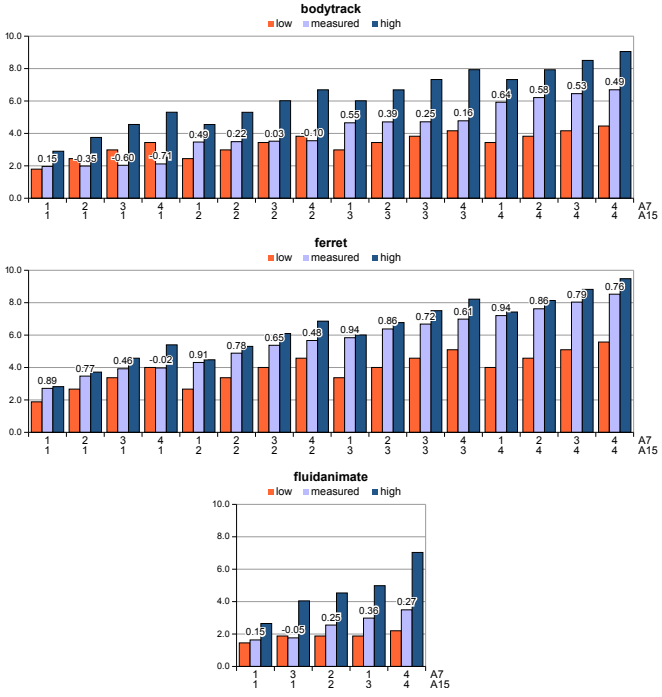


Figure B.14: PARSEC speedup range results from heterogeneous system setup determining q – the quality of the system load balancer.

C

PARSEC RESULTS

Table C.1: Performance and power calculations of Bodytrack

Bodytrack						
No. of Cores	freq MHz	IPS	Energy	power	PNP	EDP
1	3.7	6.92E+09	3905.54	33.66	2.05E+08	453139.58
2	3.7	1.36E+10	2523.52	40.96	3.31E+08	155468.47
3	3.7	1.93E+10	2048.08	46.90	4.12E+08	89438.83
4	3.7	2.44E+10	1826.32	52.19	4.67E+08	63909.41
1	3	5.63E+09	3798.48	26.66	2.11E+08	541109.88
2	3	1.09E+10	2407.64	31.53	3.47E+08	183875.11
3	3	1.56E+10	1909.43	35.39	4.41E+08	103023.83
4	3	1.97E+10	1680.10	38.83	5.07E+08	72685.82
1	2.1	4.05E+09	3882.82	19.03	2.13E+08	792088.21
2	2.1	7.71E+09	2381.46	22.03	3.50E+08	257416.19
3	2.1	1.09E+10	1863.76	24.18	4.52E+08	143670.22
4	2.1	1.37E+10	1623.61	26.11	5.26E+08	100978.11
1	1.2	2.31E+09	5117.72	14.39	1.61E+08	1819750.76
2	1.2	4.39E+09	2986.41	15.69	2.80E+08	568370.17
3	1.2	6.11E+09	2311.13	16.74	3.65E+08	319044.15
4	1.2	7.47E+09	2024.61	17.62	4.24E+08	232635.80

Table C.2: Performance and power calculations of Blackscholes

Blackscholes						
No. of Cores	freq MHz	IPS	Energy	power	PNP	EDP
1	3.7	5.53E+09	4199.55	32.94	1.68E+08	535373.45
2	3.7	9.78E+09	2799.67	37.81	2.59E+08	207320.16
3	3.7	1.34E+10	2277.58	41.91	3.20E+08	123788.89
4	3.7	1.64E+10	2060.89	45.00	3.64E+08	94391.06
1	3	4.46E+09	4116.64	25.98	1.72E+08	652423.69
2	3	7.99E+09	2697.26	29.44	2.71E+08	247118.99
3	3	1.11E+10	2163.13	32.28	3.43E+08	144975.08
4	3	1.34E+10	1919.82	34.41	3.90E+08	107109.81
1	2.1	3.22E+09	4105.80	18.85	1.71E+08	894514.59
2	2.1	5.67E+09	2642.31	21.02	2.70E+08	332224.35
3	2.1	7.72E+09	2137.11	22.61	3.41E+08	202009.36
4	2.1	9.45E+09	1897.12	23.97	3.94E+08	150129.83
1	1.2	1.84E+09	5366.67	14.11	1.31E+08	2041890.24
2	1.2	3.28E+09	3294.99	15.12	2.17E+08	717863.56
3	1.2	4.41E+09	2631.79	15.82	2.79E+08	437734.28
4	1.2	5.44E+09	2309.18	16.57	3.28E+08	321872.04

Table C.3: Performance and power calculations of Facesim

Facesim						
No. of Cores	freq MHz	IPS	Energy	power	PNP	EDP
1	3.7	7.24E+09	10174.34	32.40	2.23E+08	3195365.02
2	3.7	1.38E+10	7094.84	42.85	3.21E+08	1174764.14
3	3.7	2.04E+10	5911.50	49.80	4.10E+08	701793.49
4	3.7	2.51E+10	5532.28	55.91	4.49E+08	547414.66
1	3	5.91E+09	10187.28	27.31	2.17E+08	3799874.77
2	3	1.13E+10	6622.48	33.11	3.40E+08	1324685.99
3	3	1.71E+10	5347.50	38.02	4.50E+08	752199.75
4	3	2.14E+10	4905.55	41.86	5.11E+08	574879.13
1	2.1	4.20E+09	10311.39	19.62	2.14E+08	5418475.93
2	2.1	8.02E+09	6458.28	23.03	3.48E+08	1811418.81
3	2.1	1.22E+10	5109.18	25.97	4.71E+08	1005290.81
4	2.1	1.52E+10	4694.00	28.08	5.40E+08	784657.39
1	1.2	2.42E+09	13659.55	15.00	1.62E+08	12438636.71
2	1.2	4.64E+09	7829.55	16.12	2.88E+08	3803940.35
3	1.2	3.09E+09	6187.93	17.51	1.76E+08	2186225.66
4	1.2	8.35E+09	5812.14	18.21	4.59E+08	1855359.78

Table C.4: Performance and power calculations of Fluidanimate

Fluidanimate						
No. of Cores	freq MHz	IPS	Energy	power	PNP	EDP
1	3.7	1.04E+08	8555.73	33.17	3.14E+06	2206957.18
2	3.7	1.95E+08	5622.75	40.79	4.79E+06	775017.83
3	3.7					
4	3.7	3.45E+08	4284.47	53.97	6.39E+06	340131.28
1	3	5.29E+09	8375.15	26.26	2.01E+08	2671430.18
2	3	1.03E+10	5320.71	31.54	3.27E+08	897523.41
3	3					
4	3	1.90E+10	3829.62	40.35	4.70E+08	363482.28
1	2.1	3.81E+09	8424.10	19.02	2.00E+08	3730606.00
2	2.1	7.35E+09	5152.43	22.01	3.34E+08	1206237.49
3	2.1					
4	2.1	1.36E+10	3601.20	27.05	5.02E+08	479351.12
1	1.2	2.20E+09	10887.19	14.19	1.55E+08	8356016.27
2	1.2	4.26E+09	6293.88	15.56	2.74E+08	2546224.42
3	1.2					
4	1.2	7.75E+09	4317.46	17.79	4.36E+08	1047881.06

Table C.5: Performance and power calculations of Freqmine

Freqmine						
No. of Cores	freq MHz	IPS	Energy	power	PNP	EDP
1	3.7	6.90E+09	12335.36	33.06	2.09E+08	4602386.39
2	3.7	1.37E+10	7717.39	40.97	3.34E+08	1453586.16
3	3.7	2.04E+10	6100.80	48.29	4.23E+08	770736.60
4	3.7	2.70E+10	5270.18	54.97	4.90E+08	505257.03
1	3	5.43E+09	12364.13	26.10	2.08E+08	5857848.46
2	3	1.11E+10	7279.42	31.43	3.54E+08	1686011.80
3	3	1.66E+10	5662.12	36.37	4.56E+08	881511.56
4	3	2.18E+10	4846.90	40.85	5.34E+08	575086.55
1	2.1	3.89E+09	12558.70	18.99	2.05E+08	8306133.98
2	2.1	7.85E+09	7195.71	21.94	3.58E+08	2359692.03
3	2.1	1.16E+10	5469.50	24.65	4.71E+08	1213807.70
4	2.1	1.52E+10	4633.93	27.29	5.56E+08	786893.13
1	1.2	2.26E+09	16121.58	14.12	1.60E+08	18407108.16
2	1.2	4.50E+09	8876.99	15.51	2.90E+08	5081527.99
3	1.2	6.62E+09	6231.96	16.00	4.14E+08	2427138.47
4	1.2	8.45E+09	5501.99	18.01	4.69E+08	1680969.72

Table C.6: Performance and power calculations of Swaptions

Swaptions						
No. of Cores	freq MHz	IPS	Energy	power	PNP	EDP
1	3.7	6.99E+09	6694.45	32.97	2.12E+08	1359383.58
2	3.7	1.41E+10	4098.59	40.54	3.47E+08	414328.34
3	3.7	2.08E+10	3250.14	47.55	4.37E+08	222133.61
4	3.7	2.82E+10	2745.95	54.58	5.17E+08	138151.80
1	3	5.71E+09	6468.57	26.00	2.20E+08	1609337.81
2	3	1.13E+10	3933.77	31.34	3.61E+08	493687.58
3	3	1.72E+10	2984.39	36.18	4.76E+08	246194.01
4	3	2.29E+10	2517.65	40.56	5.64E+08	156278.45
1	2.1	4.09E+09	6583.68	18.94	2.16E+08	2288314.45
2	2.1	8.19E+09	3798.71	21.87	3.74E+08	659906.92
3	2.1	1.21E+10	2872.44	24.42	4.95E+08	337933.93
4	2.1	1.59E+10	2425.18	27.14	5.86E+08	216712.97
1	1.2	2.34E+09	8549.87	14.09	1.66E+08	5188724.66
2	1.2	4.69E+09	4684.30	15.44	3.04E+08	1421583.79
3	1.2	6.81E+09	3482.71	16.65	4.09E+08	728602.71
4	1.2	8.89E+09	2867.65	17.93	4.96E+08	458714.42

Table C.7: Performance and power calculations of Streamcluster

Streamcluster						
No. of Cores	freq MHz	IPS	Energy	power	PNP	EDP
1	3.7	2.92E+09	12729.37	32.30	9.03E+07	5017185.18
2	3.7	4.98E+09	8495.96	39.41	1.26E+08	1831561.94
3	3.7	6.91E+09	7135.69	45.83	1.51E+08	1111041.45
4	3.7	8.59E+09	6422.53	51.63	1.66E+08	798922.63
1	3	2.42E+09	11123.39	25.90	9.34E+07	4776373.74
2	3	4.46E+09	7256.75	30.87	1.45E+08	1705906.09
3	3	6.36E+09	5913.80	35.30	1.80E+08	990714.33
4	3	8.02E+09	5304.18	39.57	2.03E+08	711084.76
1	2.1	2.02E+09	9809.43	19.03	1.06E+08	5056064.61
2	2.1	3.74E+09	6139.67	21.94	1.71E+08	1717964.59
3	2.1	5.36E+09	4858.40	24.60	2.18E+08	959626.05
4	2.1	6.78E+09	4302.31	26.80	2.53E+08	690611.58
1	1.2	1.38E+09	10707.24	14.27	9.70E+07	8031485.07
2	1.2	2.58E+09	6461.65	15.93	1.62E+08	2621338.88
3	1.2	3.77E+09	4873.99	17.01	2.22E+08	1396793.43
4	1.2	4.73E+09	4270.35	18.06	2.62E+08	1009682.33

Table C.8: Performance and power calculations of Canneal

Canneal						
No. of Cores	freq MHz	IPS	Energy	power	PNP	EDP
1	3.7	8.06E+08	4942.14	31.73	2.54E+07	769661.46
2	3.7	1.35E+09	3392.85	36.01	3.75E+07	319695.35
3	3.7	1.76E+09	2885.52	39.17	4.49E+07	212571.81
4	3.7	2.06E+09	2667.93	41.41	4.97E+07	171896.23
1	3	7.36E+08	4349.86	25.49	2.89E+07	742371.09
2	3	1.24E+09	2968.02	28.64	4.32E+07	307568.67
3	3	1.59E+09	2490.65	30.74	5.19E+07	201828.43
4	3	1.95E+09	2265.42	32.20	6.06E+07	159372.19
1	2.1	6.08E+08	3861.71	18.69	3.25E+07	798076.69
2	2.1	1.02E+09	2571.75	20.42	5.00E+07	323939.42
3	2.1	1.35E+09	2143.70	21.65	6.25E+07	212224.63
4	2.1	1.61E+09	1963.60	22.80	7.05E+07	169112.30
1	1.2	4.21E+08	4216.27	14.11	2.98E+07	1259953.94
2	1.2	7.20E+08	2731.83	14.96	4.82E+07	498996.81
3	1.2	9.64E+08	2243.60	15.53	6.21E+07	324140.69
4	1.2	1.17E+09	2025.54	15.89	7.35E+07	258206.94

Table C.9: Performance and power calculations of Dedup

Dedup						
No. of Cores	Freq GHz	IPS	Energy	power	PNP	EDP
1	3.7	6.19E+09	808.47	33.02	1.88E+08	19794.43
2	3.7	1.22E+10	577.42	34.16	3.58E+08	9760.99
3	3.7	1.74E+10	472.52	35.15	4.96E+08	6352.48
4	3.7	2.31E+10	459.63	35.94	6.43E+08	5877.73
1	3	4.94E+09	818.74	25.66	1.92E+08	26128.20
2	3	9.11E+09	464.00	31.52	2.89E+08	6831.29
3	3	1.41E+10	351.17	34.85	4.05E+08	3538.35
4	3	1.88E+10	329.17	37.21	5.06E+08	2911.60
1	2.1	3.52E+09	780.43	19.43	1.81E+08	31344.91
2	2.1	6.68E+09	482.47	20.91	3.19E+08	11129.99
3	2.1	9.90E+09	421.82	22.06	4.49E+08	8065.08
4	2.1	1.26E+10	317.61	25.40	4.96E+08	3971.90
1	1.2	2.01E+09	1024.96	14.49	1.39E+08	72482.96
2	1.2	3.85E+09	566.03	15.66	2.46E+08	20464.02
3	1.2	5.65E+09	413.03	16.68	3.39E+08	10225.63
4	1.2	7.22E+09	383.16	17.26	4.18E+08	8507.99

Part III

Thesis Bibliography

BIBLIOGRAPHY

[1] "PThreads benchmark." https://github.com/ashurrafiev/PThreads.

[2] R. C. Jaeger and T. N. Blalock, *Microelectronic Circuit Design, 4th Edition*. New York, NY, USA: McGraw Hill, 2011.

[3] M. D. Hill and M. R. Marty, "Amdahl's law in the multicore era," *Computer*, vol. 41, pp. 33–38, July 2008.

[4] M. A. N. Al-hayanni, R. Shafik, A. Rafiev, F. Xia, and A. Yakovlev, "Data of speedup and parallelization models for many-core systems using performance counters." http://async.org.uk/data/speed-up-2016/.

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

[6] G. E. Moore, "Cramming more components onto integrated circuits, reprinted from electronics, volume 38, number 8, april 19, 1965, pp.114 ff.," *IEEE Solid-State Circuits Society Newsletter*, vol. 11, pp. 33–35, September 2006.

[7] J. Koomey, S. Berard, M. Sanchez, and H. Wong, "Implications of historical trends in the electrical efficiency of computing," *IEEE Annals of the History of Computing*, vol. 33, pp. 46–54, March 2011.

[8] "The international technology roadmap for semiconductors." http://www.itrs2.net/, 2017.

[9] "IEEE international solid-state circuits conference (ISSCC)." http://isscc.org/, 2017.

[10] F. J. Pollack, "New microarchitecture challenges in the coming generations of cmos process technologies (keynote address)(abstract only)," in *Proceedings of the 32nd Annual ACM/IEEE International Symposium on Microarchitecture*, MICRO 32, (Washington, DC, USA), p. 2, IEEE Computer Society, 1999.

[11] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," *IEEE Solid-State Circuits Society Newsletter*, vol. 12, pp. 19–20, Summer 2007.

[12] J. L. Gustafson, "Reevaluating amdahl's law," *Communications of ACM*, vol. 31, pp. 532–533, May 1988.

[13] X. H. Sun and L. M. Ni, "Another view on parallel speedup," in *Proceedings Super Computing '90*, pp. 324–333, November 1990.

[14] X.-H. Sun and L. M. Ni, "Scalable problems and memory-bounded speedup," *Journal of Parallel and Distributed Computing*, vol. 19, no. 1, pp. 27–37, 1993.

[15] J. Tschanz, S. Narendra, Y. Ye, B. Bloechel, S. Borkar, and V. De, "Dynamic-sleep transistor and body bias for active leakage power control of microprocessors," pp. 102–481 vol.1, February 2003.

[16] S. Eyerman and L. Eeckhout, "Fine-grained DVFS using on-chip regulators," *ACM Transactions on Architecture and Code Optimization*, vol. 8, pp. 1:1–1:24, February 2011.

[17] A. Das, M. Schuchhardt, N. Hardavellas, G. Memik, and A. Choudhary, "Dynamic directories: A mechanism for reducing on-chip interconnect power in multicores," in *2012 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 479–484, March 2012.

[18] T. Somu Muthukaruppan, A. Pathania, and T. Mitra, "Price theory based power management for heterogeneous multi-cores," *SIGARCH Computer Architecture News*, vol. 42, pp. 161–176, February 2014.

[19] X.-H. Sun and Y. Chen, "Reevaluating amdahl's law in the multicore era," *J. Parallel Distributed Computer*, vol. 70, pp. 183–188, February 2010.

[20] N. Ye, Z. Hao, and X. Xie, "The speedup model for manycore processor," in *2013 International Conference on Information Science and Cloud Computing Companion*, pp. 469–474, December 2013.

[21] D. H. Woo and H. H. S. Lee, "Extending amdahl's law for energy-efficient computing in the many-core era," *Computer*, vol. 41, pp. 24–31, December 2008.

[22] U. Gupta, S. Korrapati, N. Matturu, and U. Y. Ogras, "A generic energy optimization framework for heterogeneous platforms using scaling models," *Microprocessors and Microsystems*, vol. 40, no. Supplement C, pp. 74 – 87, 2016.

[23] P. Greenhalgh, "White paper: big.little processing with arm cortex-A15 and cortex-A7 - improving energy efficiency in high-performance mobile platforms," 2011.

[24] E. Humenay, D. Tarjan, and K. Skadron, "Impact of process variations on multicore performance symmetry," in *2007 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 1–6, April 2007.

[25] B. R. Rau and J. A. Fisher, "Instruction-level parallelism," in *Encyclopedia of Computer Science*, pp. 883–887, John Wiley and Sons Ltd.

[26] J. T. Oplinger, D. L. Heine, and M. S. Lam, "In search of speculative thread-level parallelism," in *1999 International Conference on Parallel Architectures and Compilation Techniques (Cat. No.PR00425)*, pp. 303–313, 1999.

[27] A. B. Downey, "A model for speedup of parallel programs," tech. rep., Berkeley, CA, USA, 1997.

[28] D. Moncrieff, R. E. Overill, and S. Wilson, "Heterogeneous computing machines and amdahl's law," *Parallel Computing*, vol. 22, no. 3, pp. 407–413, 1996.

[29] S. Sridharan, G. Gupta, and G. S. Sohi, "Adaptive, efficient, parallel execution of parallel programs," in *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '14, (New York, NY, USA), pp. 169–180, ACM, 2014.

[30] H. Wong and T. M. Aamodt, "The performance potential for single application heterogeneous systems," in *8th Workshop on Duplicating, Deconstructing, and Debunking*, 2009.

[31] T. Y. Morad, U. C. Weiser, A. Kolodny, M. Valero, and E. Ayguade, "Performance, power efficiency and scalability of asymmetric cluster chip multiprocessors," *IEEE Computer Architecture Letter*, vol. 5, pp. 4–17, January 2006.

[32] T. Sato, H. Mori, R. Yano, and T. Hayashida, "Importance of single-core performance in the multicore era," in *Proceedings of the 35th Australasian Computer Science Conference - Volume 122*, ACSC '12, (Darlinghurst, Australia, Australia), pp. 107–114, Australian Computer Society, Inc., 2012.

[33] T. Zidenberg, I. Keslassy, and U. Weiser, "Multiamdahl: How should I divide my heterogenous chip?," *IEEE Computer Architecture Letters*, vol. 11, pp. 65–68, July 2012.

[34] A. Morad, T. Y. Morad, Y. Leonid, R. Ginosar, and U. Weiser, "Generalized multiamdahl: Optimization of heterogeneous multi-accelerator soc," *IEEE Computer Architecture Letters*, vol. 13, pp. 37–40, January 2014.

[35] S. Mittal, "A survey of techniques for improving energy efficiency in embedded computing systems," *International Journal of Computer Aided Engineering and Technology*, vol. 6, no. 4, pp. 440–459, 2014.

[36] Y. H. Chen, Y. L. Tang, Y. Y. Liu, A. C. H. Wu, and T. Hwang, "A novel cache-utilization based dynamic voltage frequency scaling (DVFS) mechanism for reliability enhancements," in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 79–84, March 2016.

[37] L. F. Wilson and W. Shen, "Experiments in load migration and dynamic load balancing in speedes," in *1998 Winter Simulation Conference. Proceedings (Cat. No.98CH36274)*, vol. 1, pp. 483–490 vol.1, December 1998.

[38] J. C. Ryou and J. S. K. Wong, "A task migration algorithm for load balancing in a distributed system," in *Proceedings of the Twenty-Second Annual Hawaii International Conference on System Sciences. Volume II: Software Track*, vol. 2, pp. 1041–1048 vol.2, January 1989.

[39] S. Johari and A. Kumar, "Algorithmic approach for applying load balancing during task migration in multi-core system," in *2014 International Conference on Parallel, Distributed and Grid Computing*, pp. 27–32, December 2014.

[40] Y. Li, J. Niu, X. Long, and M. Qiu, "Energy efficient scheduling with probability and task migration considerations for soft real-time systems,"

in *2014 IEEE Computers, Communications and IT Applications Conference*, pp. 287–293, October 2014.

[41] A. B. Kaul, *Microelectronics to Nanoelectronics: Materials, Devices and Manufacturability*. CRC Press, August 2012.

[42] R. H. Dennard, F. H. Gaensslen, V. L. Rideout, E. Bassous, and A. R. LeBlanc, "Design of ion-implanted mosfet's with very small physical dimensions," *IEEE Journal of Solid-State Circuits*, vol. 9, pp. 256–268, October 1974.

[43] M. Bohr, "A 30 year retrospective on dennard mosfet scaling paper," *IEEE Solid-State Circuits Society Newsletter*, vol. 12, pp. 11–13, Winter 2007.

[44] S. Borkar, "Design challenges of technology scaling," *IEEE Micro*, vol. 19, pp. 23–29, Julay 1999.

[45] S. Borkar, "Exascale computing - a fact or a fiction?," in *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, pp. 3–3, May 2013.

[46] R. Ramanathan, "White paper: Intel multi-core processors making the move to quad-core and beyond," tech. rep., 2006.

[47] J. Parkhurst, J. Darringer, and B. Grundmann, "From single core to multi-core: Preparing for a new exponential," in *2006 IEEE/ACM International Conference on Computer Aided Design*, pp. 67–72, November 2006.

[48] J. Rabaey and M. Pedram, *Low Power Design Methodologies*. The Springer International Series in Engineering and Computer Science, Springer US, 2012.

[49] E. Yao, Y. Bao, G. Tan, and M. Chen, "Extending amdahl's law in the multicore era," *SIGMETRICS Perform. Eval. Rev.*, vol. 37, pp. 24–26, October 2009.

[50] A. Vajda, *Multi-core and Many-core Processor Architectures*. Springer US, 2011.

[51] R. Kumar, D. M. Tullsen, P. Ranganathan, N. P. Jouppi, and K. I. Farkas, "Single-isa heterogeneous multi-core architectures for multithreaded

workload performance," in *Proceedings. 31st Annual International Symposium on Computer Architecture, 2004.*, pp. 64–75, June 2004.

[52] M. G. Arenas, A. M. Mora, G. Romero, and P. A. Castillo, *GPU Computation in Bioinspired Algorithms: A Review*, pp. 433–440. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.

[53] C. M. Wittenbrink, E. Kilgariff, and A. Prabhu, "Fermi GF100 GPU architecture," *IEEE Micro*, vol. 31, pp. 50–59, March 2011.

[54] M. Andersch, J. Lucas, M. A. LvLvarez-Mesa, and B. Juurlink, "On latency in GPU throughput microarchitectures," in *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 169–170, March 2015.

[55] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen, "Single-isa heterogeneous multi-core architectures: the potential for processor power reduction," in *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36.*, pp. 81–92, December 2003.

[56] A. Venkat and D. M. Tullsen, "Harnessing isa diversity: Design of a heterogeneous-isa chip multiprocessor," in *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, pp. 121–132, June 2014.

[57] V. W. Lee, E. Grochowski, and R. Geva, "Performance benefits of heterogeneous computing in hpc workloads," in *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops PhD Forum*, pp. 16–26, May 2012.

[58] N. P. Khanyile, J.-R. Tapamo, and E. Dube, "An analytic model for predicting the performance of distributed applications on multicore clusters," 2012.

[59] E. S. Chung, P. A. Milder, J. C. Hoe, and K. Mai, "Single-chip heterogeneous computing: Does the future include custom logic, FPGAs, and GPGPUs?," in *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 225–236, December 2010.

[60] A. Marowka, "Extending amdahl's law for heterogeneous computing," in *2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications*, pp. 309–316, July 2012.

[61] J. Power, A. Basu, J. Gu, S. Puthoor, B. M. Beckmann, M. D. Hill, S. K. Reinhardt, and D. A. Wood, "Heterogeneous system coherence for integrated CPU-GPU systems," in *2013 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 457–467, December 2013.

[62] A. Pathania, Q. Jiao, A. Prakash, and T. Mitra, "Integrated CPU-GPU power management for 3D mobile games," in *Proceedings of the 51st Annual Design Automation Conference*, DAC '14, (New York, NY, USA), pp. 40:1–40:6, ACM, 2014.

[63] V. W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. D. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. Hammarlund, R. Singhal, and P. Dubey, "Debunking the 100x GPU vs. CPU myth: An evaluation of throughput computing on CPU and GPU," *SIGARCH Computer Architecture News*, vol. 38, pp. 451–460, June 2010.

[64] J. Palacios and J. Triska, "A comparison of modern GPU and CPU architectures: And the common convergence of both," 2011.

[65] C. Cullinan, C. Wyant, T. Frattesi, and X. Huang, "Computing performance benchmarks among CPU, GPU, and FPGA," *Internet: www. wpi. edu/Pubs/E-project/Available/E-project-030212-123508/unrestricted/Benchmarking Final*, 2013.

[66] K. J. Duda and D. R. Cheriton, "Borrowed-virtual-time (BVT) scheduling: Supporting latency-sensitive threads in a general-purpose scheduler," *SIGOPS Oper. Syst. Rev.*, vol. 33, pp. 261–276, December 1999.

[67] A. Agarwal, R. Bianchini, D. Chaiken, K. L. Johnson, D. Kranz, J. Kubiatowicz, B.-H. Lim, K. Mackenzie, and D. Yeung, "The mit alewife machine: Architecture and performance," *SIGARCH Computer Architecture News*, vol. 23, pp. 2–13, May 1995.

[68] B. Boothe and A. Ranade, "Improved multithreading techniques for hiding communication latency in multiprocessors," *SIGARCH Computer Architecture News*, vol. 20, pp. 214–223, April 1992.

[69] W. Kim, M. S. Gupta, G. Y. Wei, and D. Brooks, "System level analysis of fast, per-core DVFS using on-chip switching regulators," in *2008 IEEE 14th International Symposium on High Performance Computer Architecture*, pp. 123–134, February 2008.

[70] K. Ma, X. Li, M. Chen, and X. Wang, "Scalable power control for many-core architectures running multi-threaded applications," in *2011 38th Annual International Symposium on Computer Architecture (ISCA)*, pp. 449–460, June 2011.

[71] "Intel." https://www.intel.co.uk/content/www/uk/en/homepage.html, 2017.

[72] B. Goel and S. A. McKee, "A methodology for modeling dynamic and static power consumption for multicore processors," in *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 273–282, May 2016.

[73] P. Bogdan, R. Marculescu, and S. Jain, "Dynamic power management for multidomain system-on-chip platforms: An optimal control approach," *ACM Transactions on Design Automation of Electronic Systems (TODAES)-Special Section on Networks on Chip: Architecture, Tools, and Methodologies*, vol. 18, pp. 46:1–46:20, October 2013.

[74] A. Bhattacharjee and M. Martonosi, "Thread criticality predictors for dynamic performance, power, and resource management in chip multiprocessors," *SIGARCH Computer Architecture News*, vol. 37, pp. 290–301, June 2009.

[75] S. Huang, M. Lang, S. Pakin, and S. Fu, "Measurement and characterization of haswell power and energy consumption," in *Proceedings of the 3rd International Workshop on Energy Efficient Supercomputing*, E2SC '15, (New York, NY, USA), pp. 7:1–7:10, ACM, 2015.

[76] P. B. S. Raju and P. Govindarajulu, "Pi-tool to improve performance of application in multi-core architecture," *International Journal of Computer Science and Security (IJCSS)*, vol. 8, no. 4, pp. 84–96, 2014.

[77] Y. Li, J. Niu, J. Zhang, M. Atiquzzaman, and X. Long, "An optimized RM algorithm by task affinity on multi-core processor," in *2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 286–293, December 2016.

[78] Q. Hu, P. Liu, and M. C. Huang, "Threads and data mapping: Affinity analysis for traffic reduction," *IEEE Computer Architecture Letters*, vol. 15, pp. 133–136, July 2016.

[79] J. Ye, S. Li, T. Chen, M. Wu, and L. Liu, "Core affinity code block schedule to reduce inter-core data synchronization of spmt," in *IEEE International Conference on High Performance Computing and Communications, IEEE 6th Intl Symp on Cyberspace Safety and Security, IEEE 11th International Conference on Embedded Software and System (HPCC,CSS,ICESS)*, pp. 1002–1007, August 2014.

[80] K. Singh, M. Bhadauria, and S. A. McKee, "Real time power estimation and thread scheduling via performance counters," *ACM SIGARCH Computer Architecture News*, vol. 37, no. 2, pp. 46–55, 2009.

[81] M. A. Qayum, N. A. Siddique, M. A. Haque, and A. S. M. Tayeen, "Future of multiprocessors: Heterogeneous chip multiprocessors," in *2012 International Conference on Informatics, Electronics Vision (ICIEV)*, pp. 372–376, May 2012.

[82] E. Yao, Y. Bao, and M. Chen, "What Hill–Marty model learn from and break through Amdahl's law?," *Information Processing Letters*, vol. 111, no. 23, pp. 1092–1095, 2011.

[83] B. Juurlink and C. H. Meenderinck, "Amdahl's law for predicting the future of multicores considered harmful," *SIGARCH Computer Architecture News*, vol. 40, pp. 1–9, May 2012.

[84] W. A. Wulf and S. A. McKee, "Hitting the memory wall: Implications of the obvious," *SIGARCH Computer Architecture News*, vol. 23, pp. 20–24, Mar. 1995.

[85] B. M. Rogers, A. Krishna, G. B. Bell, K. Vu, X. Jiang, and Y. Solihin, "Scaling the bandwidth wall: Challenges in and avenues for cmp scaling," *SIGARCH Computer Architecture News*, vol. 37, pp. 371–382, June 2009.

[86] R. W. W. Sally A. McKee, *Memory Wall*. Springer US, 2011.

[87] R. Kumar, V. Zyuban, and D. M. Tullsen, "Interconnections in multi-core architectures: understanding mechanisms, overheads and scaling," in *32nd International Symposium on Computer Architecture (ISCA'05)*, pp. 408–419, June 2005.

[88] E. R. Rodrigues, F. L. Madruga, P. O. A. Navaux, and J. Panetta, "Multi-core aware process mapping and its impact on communication overhead

of parallel applications," in *2009 IEEE Symposium on Computers and Communications*, pp. 811–817, July 2009.

[89] T. B. Ahmad and M. Ciesielski, "An approach to multi-core functional gate-level simulation minimizing synchronization and communication overheads," in *2013 14th International Workshop on Microprocessor Test and Verification*, pp. 77–82, December 2013.

[90] R. Zurawski, *Embedded Systems Handbook, Second Edition: Embedded Systems Design and Virification*. Taylor and Francis, 2009.

[91] X.-H. Sun, Y. Chen, and S. Byna, "Scalable computing in the multicore era," in *Proceedings of the International Symposium on Parallel Architectures, Algorithms and Programming*, 2008.

[92] S. Pei, J. Zhang, L. Jiang, M.-S. Kim, and J.-L. Gaudiot, "Reevaluating the overhead of data preparation for asymmetric multicore system on graphics processing.," *KSII Transactions on Internet & Information Systems*, vol. 10, no. 7, 2016.

[93] S. Pei, M. S. Kim, and J. L. Gaudiot, "Extending amdahl's law for heterogeneous multicore processor with consideration of the overhead of data preparation," *IEEE Embedded Systems Letters*, vol. 8, pp. 26–29, March 2016.

[94] G. H. Loh, "The cost of uncore in throughput-oriented many-core processors," in *In Proc. of Workshop on Architectures and Languages for Troughput Applications (ALTA)*, pp. 1–9, 2008.

[95] E. Blem, H. Esmaeilzadeh, R. S. Amant, K. Sankaralingam, and D. Burger, "Multicore model from abstract single core inputs," *IEEE Computer Architecture Letters*, vol. 12, pp. 59–62, July 2013.

[96] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The splash-2 programs: Characterization and methodological considerations," *SIGARCH Computer Architecture News*, vol. 23, pp. 24–36, May 1995.

[97] L. Yavits, A. Morad, and R. Ginosar, "The effect of communication and synchronization on amdahl's law in multicore systems," *Parallel Computing*, vol. 40, no. 1, pp. 1 – 16, 2014.

[98] X. Li and M. Malek, "Analysis of speedup and communication/computation ratio in multiprocessor systems," in *Proceedings. Real-Time Systems Symposium*, pp. 282–288, December 1988.

[99] X. Chen, Z. Lu, A. Jantsch, and S. Chen, "Speedup analysis of data-parallel applications on multi-core nocs," in *2009 IEEE 8th International Conference on ASIC*, pp. 105–108, October 2009.

[100] N. P. Khanyile, J.-R. Tapamo, and E. Dube, "Performance prediction model for distributed applications on multicore clusters," 2012.

[101] T. Huang, Y. Zhu, M. Qiu, X. Yin, and X. Wang, "Extending amdahl's law and gustafson's law by evaluating interconnections on multi-core processors," *The Journal of Supercomputing*, vol. 66, pp. 305–319, October 2013.

[102] R. Joseph and M. Martonosi, "Run-time power estimation in high performance microprocessors," in *Proceedings of the 2001 International Symposium on Low Power Electronics and Design*, ISLPED '01, (New York, NY, USA), pp. 135–140, ACM, 2001.

[103] K. Rajamani, H. Hanson, J. Rubio, S. Ghiasi, and F. Rawson, "Application-aware power management," in *Workload Characterization, 2006 IEEE International Symposium on*, pp. 39–48, IEEE, 2006.

[104] B. Su, J. Gu, L. Shen, W. Huang, J. L. Greathouse, and Z. Wang, "Ppep: Online performance, power, and energy prediction framework and DVFS space exploration," in *Microarchitecture (MICRO), 2014 47th Annual IEEE/ACM International Symposium on*, pp. 445–457, IEEE, 2014.

[105] K. K. Pusukuri, R. Gupta, and L. N. Bhuyan, "Thread reinforcer: Dynamically determining number of threads via OS level monitoring," in *2011 IEEE International Symposium on Workload Characterization (IISWC)*, pp. 116–125, November 2011.

[106] H. Sasaki, S. Imamura, and K. Inoue, "Coordinated power-performance optimization in manycores," in *Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques*, pp. 51–61, September 2013.

[107] K. W. Cameron and R. Ge, "Generalizing amdahl's law for power and energy," *Computer*, vol. 45, pp. 75–77, March 2012.

[108] S. Cho and R. Melhem, "Corollaries to amdahl's law for energy," *IEEE Computer Architecture Letters*, vol. 7, pp. 25–28, January 2008.

[109] S. M. Londono and J. P. de Gyvez, "Extending amdahl's law for energy-efficiency," in *2010 International Conference on Energy Aware Computing*, pp. 1–4, December 2010.

[110] J. Issa and S. Figueira, "Performance and power-consumption analysis of mobile internet devices," in *30th IEEE International Performance Computing and Communications Conference*, pp. 1–6, November 2011.

[111] H. Nejatollahi and M. E. Salehi, "Effect of voltage scaling on symmetric multicore's speed-up," in *2014 22nd Iranian Conference on Electrical Engineering (ICEE)*, pp. 862–867, May 2014.

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

[113] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, "Power challenges may end the multicore era," *Communications of ACM*, vol. 56, pp. 93–102, February 2013.

[114] M. Shafique, S. Garg, J. Henkel, and D. Marculescu, "The eda challenges in the dark silicon era: Temperature, reliability, and variability perspectives," in *Proceedings of the 51st Annual Design Automation Conference*, DAC '14, (New York, NY, USA), pp. 185:1–185:6, ACM, 2014.

[115] M. Shafique, D. Gnad, S. Garg, and J. Henkel, "Variability-aware dark silicon management in on-chip many-core systems," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, pp. 387–392, EDA Consortium, 2015.

[116] H. Khdr, S. Pagani, M. Shafique, and J. Henkel, "Thermal constrained resource management for mixed ILP-TLP workloads in dark silicon chips," in *52nd ACM/EDAC/IEEE (DAC)*, pp. 1–6, June 2015.

[117] S. Pagani, H. Khdr, J. J. Chen, M. Shafique, M. Li, and J. Henkel, "Thermal safe power (TSP): Efficient power budgeting for heterogeneous

manycore systems in dark silicon," *IEEE Transactions on Computers*, vol. 66, pp. 147–162, January 2017.

[118] T. Zidenberg, I. Keslassy, and U. Weiser, "Optimal resource allocation with MultiAmdahl," *Computer*, vol. 46, pp. 70–77, July 2013.

[119] B. M. Al-Babtain, F. J. Al-Kanderi, M. F. Al-Fahad, and I. Ahmad, "A survey on Amdahl's law extension in multicore architectures," vol. 3, pp. 30–46, 01 2013.

[120] D. Moncrieff, R. E. Overill, and S. Wilson, "α critical for parallel processors," *Parallel computing*, vol. 21, no. 3, pp. 467–471, 1995.

[121] "Juno arm development platform soc technical overview." http://www.arm.com/, 2016.

[122] M. A. N. Al-hayanni, A. Rafiev, R. Shafik, and F. Xia, "Power and energy normalized speedup models for heterogeneous many core computing," in *2016 16th International Conference on Application of Concurrency to System Design (ACSD)*, pp. 84–93, June 2016.

[123] M. A. N. Al-hayanni, A. Rafiev, R. Shafik, F. Xia, and A. Yakovlev, "Extended power and energy normalized performance models for many-core systems," Tech. Rep. NCL-EEE-MICRO-TR-2016-198, Newcastle University, 2016.

[124] A. I. Elnashar, "To parallelize or not to parallelize, speed up issue," *arXiv preprint arXiv:1103.5616*, 2011.

[125] "Odroid platform." http://www.hardkernel.com/main/products/, 2015.

[126] J. L. Peterson, *Petri Net Theory and the Modeling of Systems*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1981.

[127] J. Baeten and K. Middelburg, *Process Algebra with Timing : Real Time and Discrete Time*. Elsevier, 2001.

[128] K. Georgiou, S. Kerrison, Z. Chamski, and K. Eder, "Energy transparency for deeply embedded programs," *ACM Trans. Archit. Code Optim.*, vol. 14, pp. 8:1–8:26, Mar. 2017.

[129] J.-P. Lozi, B. Lepers, J. Funston, F. Gaud, V. Quéma, and A. Fedorova, "The linux scheduler: a decade of wasted cores," in *Proceedings of the Eleventh European Conference on Computer Systems*, p. 1, ACM, 2016.

[130] "OpenCL overview." https://www.khronos.org/opencl.

[131] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: characterization and architectural implications," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pp. 72–81, ACM, 2008.

[132] C. Bienia and K. Li, "Parsec 2.0: A new benchmark suite for chip-multiprocessors," in *Proceedings of the 5th Annual Workshop on Modeling, Benchmarking and Simulation*, June 2009.

[133] B. Sprunt, "The basics of performance-monitoring hardware," *IEEE Micro*, vol. 22, no. 4, pp. 64–71, 2002.

[134] X. Wu and V. Taylor, "Utilizing hardware performance counters to model and optimize the energy and performance of large scale scientific applications on power-aware supercomputers," in *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 1180–1189, May 2016.

[135] G. T. Chetsa, L. Lefévre, J. Pierson, P. Stolf, and G. D. Costa, "Exploiting performance counters to predict and improve energy performance of hpc systems," *Future Generation Computer Systems*, vol. 36, pp. 287 – 298, 2014. Special Section: Intelligent Big Data Processing Special Section: Behavior Data Security Issues in Network Information Propagation Special Section: Energy-efficiency in Large Distributed Computing Architectures Special Section: eScience Infrastructure and Applications.

[136] J. Treibig, G. Hager, and G. Wellein, "Likwid: A lightweight performance-oriented tool suite for x86 multicore environments," in *ICPPW*, ICPPW '10, (Washington, DC, USA), pp. 207–216, IEEE Computer Society, 2010.

[137] V. M. Weaver and S. A. McKee, "Can hardware performance counters be trusted?," in *Workload Characterization, 2008. IISWC 2008. IEEE International Symposium on*, pp. 141–150, September 2008.

[138] Intel, *Intel 64 and IA-32 Architectures Software Developer's Manual. Volume 3B: System Programming Guide, Part 2*. 2016.

[139] P. Calafiura, S. Eranian, D. Levinthal, S. Kama, and R. A. Vitillo, "Gooda: The generic optimization data analyzer," in *Journal of Physics: Conference Series*, vol. 396, p. 052072, IOP Publishing, 2012.

[140] A. Nowak, D. Levinthal, and W. Zwaenepoel, "Hierarchical cycle accounting: a new method for application performance tuning," in *(ISPASS)*, pp. 112–123, IEEE, 2015.

[141] D. Hackenberg, T. Ilsche, R. Schöne, D. Molka, M. Schmidt, and W. E. Nagel, "Power measurement techniques on standard compute nodes: A quantitative comparison," in *(ISPASS)*, pp. 194–204, IEEE, 2013.

[142] J. H. Laros III, K. Pedretti, S. M. Kelly, W. Shu, K. Ferreira, J. Van Dyke, and C. Vaughan, *Energy-efficient high performance computing: measurement and tuning*. Springer Science & Business Media, 2012.

[143] M. Bhadauria, V. weaver, and S. A. Mckee, "A characterization of the PARSEC benchmark suite for CMP design," Tech. Rep. CSL-TR-2008-1052, Computer System Laboratory, Cornell University, Ithaca, NY, 2008.

[144] F. Xia, A. Rafiev, A. Aalsaud, M. A. N. Al-hayanni, J. Davis, J. Levine, A. Mokhov, A. Romanovsky, R. Shafik, A. Yakovlev, and S. Yang, "Voltage, throughput, power, reliability, and multicore scaling," *Computer*, vol. 50, no. 8, pp. 34–45, 2017.

[145] S. Yang, R. A. Shafik, G. V. Merrett, E. Stott, J. M. Levine, J. Davis, and B. M. Al-Hashimi, "Adaptive energy minimization of embedded heterogeneous systems using regression-based learning," in *2015 25th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, pp. 103–110, September 2015.

[146] A. P. Chandrakasan and R. W. Brodersen, "Minimizing power consumption in digital cmos circuits," *Proceedings of the IEEE*, vol. 83, pp. 498–523, April 1995.