COORDINATED MANAGEMENT OF THE PROCESSOR AND MEMORY FOR OPTIMIZING ENERGY EFFICIENCY

A Dissertation
Presented to
The Academic Faculty

By

Karthik Rao

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering

Georgia Institute of Technology

August 2018

COORDINATED MANAGEMENT OF THE PROCESSOR AND MEMORY FOR OPTIMIZING ENERGY EFFICIENCY

Approved by:

Dr. Sudhakar Yalamanchili, Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Yorai Wardi, Co-Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Magnus Egerstedt
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Saibal Mukhopadhyay
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Santosh Pande
School of Computer Science
Georgia Institute of Technology

Dr. Joseph Greathouse
Senior Member of Technical Staff
Advanced Micro Devices Inc.

Date Approved: May 14, 2018

कर्मण्येवाधिकारस्ते मा फलेषु कदाचन ।
मा कर्मफलहेतुर्भूर्मा ते सङ्गोऽस्त्वकर्मणि ॥ २-४७

You have the right to work only but never to its fruits.

Let not the fruits of action be your motive, nor let your attachment be to inaction.

Bhagavad Gita Chapter 2 Verse 47.

To Amma, Anna and Deepa.

ACKNOWLEDGEMENTS

गुरुर्ब्रह्मा गुरुर्विष्णुर्गुरुर्देवो महेश्वरः ।
गुरुः साक्षात् परब्रह्मा तस्मै श्री गुरवे नमः ॥

The Guru is Brahma (creator); the Guru is Vishnu (protector); the Guru is Maheshwara (destroyer). The Guru is verily the Supreme Brahman. I offer my salutations to that Guru, the remover of my darkness and my ignorance.

To my advisers, my *Gurus*, Dr. Sudhakar Yalamanchili and Dr. Yorai Wardi, I offer my sincere salutations. They have been the guiding lights in my journey through graduate school. The technical discussions were truly enriching and have expanded my perspective not just in my own area of research but on research in general as well. They have taught me to appreciate the art of identifying subtle insights and also the rigor involved in presenting ideas in a crisp and articulate manner. Looking back at my first meeting with both of them five years ago, when I was quite unsure about working in computer architecture, I can now say safely that I am a better engineer and hopefully a better researcher because of their guidance. Their general life advice and subtle course corrections have steered my ship to where it is now. I will never forget the non-technical discussions with Dr. Wardi that spanned a plethora of topics. Remembering Dr. Wardi's words, "You are not out of my orbit yet!", I eagerly look forward to continuing this relationship with both my advisers.

I would also like to thank my committee Dr. Magnus Egerstedt, Dr. Saibal Mukhopadhyay, Dr. Santosh Pande and Dr. Joseph Greathouse for providing their valuable suggestions and comments that have improved the quality of this thesis. I must thank Dr. Egerstedt in particular for introducing me to my advisers. Special thanks to Dr. Joseph Greathouse for believing in my abilities and bringing me to AMD Research.

My graduate studies have been generously supported by multiple sources and I would like to thank them profusely viz. NSF, Huawei and AMD. The internship at Huawei Technologies, that essentially decided my thesis topic, was made possible by my mentor Jun

Wang and manager Handong Ye. The constant encouragement given by Jun, Zongfang Lin and other colleagues at Huawei made me productive and insightful discussions lead to my very first patent applications.

During the six years at Georgia Tech, I had the privilege of interacting and sharing lab space with members from the CASL and GRITS Labs. I am very grateful to William Song, Minhaj Hasan, Eric Anger, Jin Wang, Xinwei Chen, Chad Kersey, Tine Blaise, Hugh Xiao, Matt Hale, Thiagarajan Ramachandran, Maria Santos, Tina Setter, Swamit Tannu, Gururaj Saileshwar and many others. The course homeworks, research collaborations and many discussions over coffees have helped me stay afloat. I look forward to maintaining the same level of communication in the future as well.

A heartfelt thank you to all my teachers who helped shape my career, especially K. Subrahmanya Bhat, Mrs. Shahin Masood, Dr. Kumara Shama, Dr. U.Sripathi, Dr. M.S.Bhat, Chirag Jain, Dr. Kadagattur Srinidhi, Dr. Ed Coyle and many others. I would be remiss if I do not thank Sundar, Garima and Dhawal, who believed in me right from our undergraduate days at Manipal. To Megha Sandesh, Ayush Sharma and many other friends at Georgia Tech, thank you for all the philosophical exchanges. Words are insufficient to express my gratitude to "Dhani" Shishir Kolathaya, who held me up during tough times. I am indebted to him. To my partners in crime, my roommates at one time or the other, my brothers, Kaivalya Bakshi and Srinivas Hanasoge. Our "conferences" on every little topic under the sun, shall never be forgotten. The joys, sorrows, frustrations and successes we shared will undoubtedly make for a good laugh later on.

My parents Jyoti Umesh and Govindarao Umesh, my sister Deepa, my grandmother Puttalakshmi and my foster parents in the US, Mr. Bhaskar and Dr. Jayanthi Srinivasiah have been my pillars of moral support since the very beginning. Without their encouraging words during tough times, I do not think I would have made it thus far. My father, a professor himself, deserves a special mention especially since he helped me edit this thesis. I aspired to be like him since my childhood and I hope I have made him proud.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Energy efficiency is a key design goal for future computing systems. With diverse components interacting with each other on the System-on-Chip (SoC), dynamically managing performance, energy and temperature is a challenge in 2D architectures and more so in a 3D stacked environment. Temperature has emerged as the parameter of primary concern. Heuristics based schemes have been employed so far to address these issues. Looking ahead into the future, complex multiphysics interactions between performance, energy and temperature reveal the limitations of such approaches. Therefore in this thesis, first, a comprehensive characterization of existing methods is carried out to identify causes for their inefficiency. Managing different components in an independent and isolated fashion using heuristics is seen to be the primary drawback. Following this, techniques based on feedback control theory to optimize the energy efficiency of the processor and memory in a coordinated fashion are developed. They are evaluated on a real physical system and a cycle-level simulator demonstrating significant improvements over prior schemes. The two main messages of this thesis are, (i) coordination between multiple components is paramount for next generation computing systems and (ii) temperature ought to be treated as a resource like compute or memory cycles.

CHAPTER 1
INTRODUCTION

The ever-increasing demand for better computational performance has driven several innovations at the hardware, software and microarchitectural levels. In tune with Moore's law, the microprocessor has undergone an exponential growth in performance since its advent in 1971. Simultaneously, supply voltage scaling governed by the tenets of Dennard scaling has enabled this increase in performance by keeping power consumption under a reasonable envelope. However, the bandwidth, latency and energy consumption of off-chip memories have not kept pace with their processor counterparts. The energy consumed by a DRAM access is more than twice that of a double precision floating point operation [1, 2]. The US Department of Energy estimates that 70% of the total power of a 100,000 node exascale system will be consumed by the DRAM accesses alone (See [3] and references therein). In the age of Cloud Computing, Big Data processing and the Internet of Things, applications require moving large amounts of data to and from off-chip memories. The performance of a computing system is, therefore, not a function of the processor alone. The memory system is more often than not, the bottleneck in deciding the overall performance while consuming a significant portion of the input power.

Addressing the so-called "Memory Wall", 3D packaging of silicon dice, enabled by advances such as Through-Silicon Via (TSV) technology [4], has led to the integration of memory and logic into a single package. Their footprint is small and they promise significant reductions in data movement latency and energy. Further, the 3D package provides an order of magnitude increase in memory bandwidth. For example, commercial standards like DDR3-1333 [5], DDR4-2667 [6], HBM2 [7] and HMC2 [8] realize 10.66 GB/s, 21.34 GB/s, 256 GB/s and 320 GB/s, respectively. To effectively exploit the high bandwidth provided by 3D die-stacked DRAM, multiple efforts have explored moving compute logic inside the package as part of the die-stack, revisiting the early efforts at architecting Processing-In-Memory (PIM) designs [9, 10, 11, 12, 13, 14, 15].

Alongside high performance server computers, the world of mobile System-on-Chips (SoC) is also rapidly expanding. It is predicted that by 2020 there will be over 20 billion mobile-connected devices [16]. The mobile CPU itself has improved 10$\times$ in performance between 2009 and 2015 [17]. Furthermore, it has evolved from a single core (2009) to multiple homogeneous cores (2011) to multiple heterogeneous cores (2013). With mobile applications becoming more memory intensive, the effects of the "memory wall" are being observed on mobile SoCs as well. Complementing the mobile CPU, newer low power mobile memories such as LPDDR [18] and Wide I/O [19] have been introduced. The mobile SoCs of today are more diverse and capable with advanced GPUs, DSPs, audio/video decoders, WiFi, 3G and 4G LTE modules etc. Despite this, the two pillars defining performance of such systems continue to be the mobile CPU and the memory.

As transistor sizes get smaller, effects due to leakage currents and subthreshold voltages, which are not accounted for in Dennard scaling rules, start to to play a dominant role. Since the breakdown of Dennard scaling in 2006, microprocessor performance scaling has demanded a more detailed understanding of the multi-physics interactions between performance, energy and temperature of the computing environment. While 3D stacking of memory and logic die delivers an order of magnitude improvement in available memory bandwidth, the price paid, however, is tighter thermal constraints. Due to thermal shielding, the logic die temperatures can reach unsustainable levels, thereby placing an upper limit on clock frequencies (and hence performance). Furthermore, large localized temperature variations across different layers of DRAM and logic die can lead to detrimental effects on the performance and reliability. It is therefore natural to ask the following questions: How should one go about managing such computing environments? Is it beneficial to expect a 'balance' between achievable performance and the physics of the device?

Attempting to answer these questions inevitably leads to a related, yet important, issue:

the choice of the optimization metric. *Power efficiency* and *energy efficiency* are two commonly used metrics. Power efficiency is measured as the ratio of Watts of power dissipated for a given number of operations executed, whereas energy efficiency is measured as the ratio of Joules of energy consumed for a given number of operations executed. In most of the research efforts in the past, the third parameter, temperature, is included in the problem definition only as a constraint to be observed. Minimizing power or energy consumed subject to performance constraints, a problem definition which falls in the same categories, have also been considered. The choice of the metric is strongly influenced by the platform i.e. mobile SoC or server class processor, primarily because the end-user demands vary widely. While a mobile phone user requires consistent performance and a long battery life, a server processor typically demands the best performance.

Researchers have proposed and implemented innovative techniques to manage processor and memory system power at various levels of the hardware and the software stack. For the processor, hardware-level power management methods such as clock gating, aggressive CPU idling and Dynamic Voltage and Frequency Scaling (DVFS) help in reigning in excess power consumption. Similarly on the software end, optimized application code, per-module governors implementing DVFS and software enabled low-power modes help to save power. Power management of memory systems involve dynamic frequency scaling, dynamic bandwidth throttling and multiple power-down states. Since the 3D stacked memory is a relatively new technology, limited studies on dynamic power and thermal management have been reported (see [20, 21, 22]). However, as discussed in CHAPTER 2, there is a need for detailed investigations to achieve higher efficiency levels.

1.1 Thesis Statement

While the power/energy/thermal management methods proposed so far work well in general, more often than not, these methods implement their respective policies for distinct elements in an independent and isolated manner. A few studies ([23, 24, 25]) have shown that independent power/performance control strategies can lead to conflicting policies causing performance and/or power losses. Addressing this problem, researchers have investigated the coordinated control of different hardware subsystems on servers [23], System-on-Chips (SoC) [26, 27, 28], heterogeneous architectures [24] and embedded systems ([29] and references therein). Looking ahead into the future, computing platforms such as servers are expected to process exabytes of data using (a) heterogeneous cores, (b) specialized and dedicated hardware (accelerators) and (c) high-bandwidth low-latency memories. Similarly, mobile devices powered by increasingly heterogeneous SoCs are expected to be highly energy efficient while providing the best end-user experience. Techniques proposed until now either, (a) optimize only power consumption, (b) are not amenable to expanding the scope of the solution or (c) are based on heuristics. Heuristics based solutions have been the state-of-the-art *and* state-of-the-practice primarily because of their simplicity and practical implementation constraints. However, the tight physical and functional coupling between the major components (processor and memory) require much better power, energy, performance and thermal management strategies that go past the achievable boundaries set by heuristics. These challenges lead to the main theme of this thesis: *Interactions between (a) thermal behaviors (b) compute and memory microarchitecture and (c) application workloads necessitate the management of performance, energy and temperature in a coordinated fashion. Control theory inspired approaches to coordinate the power management of the processor and memory can dynamically trade-off performance, energy and temperature, both leading to better energy efficiency.*

In order to go beyond heuristics based approaches, this thesis makes a case for exploiting the rich set of tools available from fields such as mathematical optimization and control theory. Feedback controllers have been employed to regulate power, throughput and temperature of the processor [30, 31, 32, 33]. There have also been research efforts that sought to optimize power, performance-per-watt, performance-per-joule etc. using more complex tools such as Model Predictive Control (MPC) [34, 35, 36]. The approach presented in this thesis is along the lines of the aforementioned works. However, the results reported here emphasize the need to coordinate power management between the processor and the memory by exploiting control theoretic tools. The notion of coordination between controllers used in this thesis is as follows. Suppose there are two controllers A and B, one for the processor (A) and another for the memory (B). Coordination between the two controllers A and B, monitoring a system output means that the decision taken by controller A for the current control cycle depends on the decision taken by B in the previous cycle and vice versa. Controllers implementing DVFS on the processor and memory have implications on the performance, energy and temperature of the entire computing system. It is demonstrated that control decisions made for the processor and memory in an uncoordinated fashion leads to energy inefficiency. This aspect is elucidated in the subsequent chapters.

A recurring theme throughout this thesis is the pursuit of simple, effective and robust feedback controllers that can be implemented on a real physical system with minimal overheads. The quest for achieving the aforementioned 'balance' is what drives this thesis towards unravelling key observations leading to simple and effective solutions. The vehicles of exploration in this thesis are the following two computing platforms: (1) A mobile SoC processor and (2) a server-class processor. The exploration *algorithm* is divided into three steps: (1) Characterization (2) Insight identification and (3) Solution development.

Figure 1.1 pictorially describes the exploration *algorithm*. The end goal is the design of TRINITY, a control theory based technique that dynamically balances the three parameters of interest: performance, energy and temperature.

Characterization: The first contribution of this thesis is the characterization of various power/energy management methods implemented in different hardware and software levels of the SoC stack, using a smartphone as an example of a mobile device. Finding the right balance between performance and energy consumption in an SoC, when application complexity varies widely, has generally been influenced by heuristics, end user demands

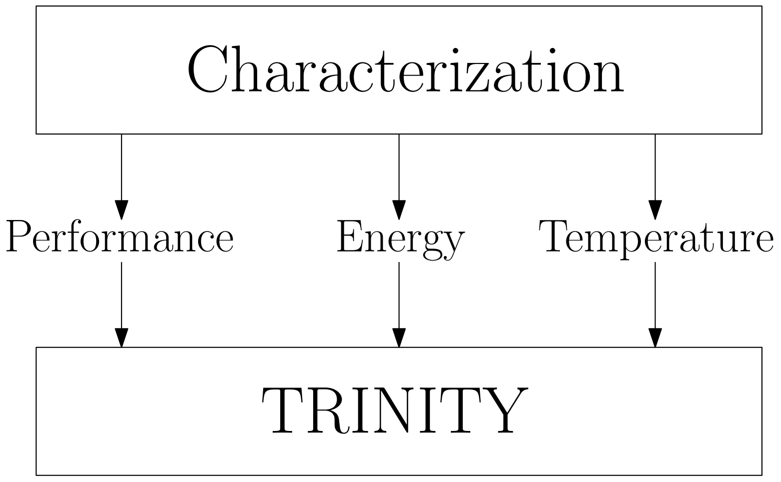


Figure 1.1: Thesis Overview: Characterization of *three* parameters leading to *TRINITY*, a control theory based solution for *3D* stacked architectures.

etc. Understanding the behavior of the SoC and its components under a variety of operating conditions helps in identifying areas of energy inefficiency. The impact on power, performance and energy by varying a multitude of design parameters and governors are analyzed individually. The investigation of interactions *between* individual governors reveals that governors implementing their policies in an isolated manner are energy inefficient, thus laying the groundwork for coordinated control of the processor and memory.

A second set of characterization experiments are conducted on a 3D stacked processor-memory architecture. A comprehensive characterization of the multi-physics interaction between (a) thermal behaviors (b) compute and memory microarchitecture and (c) application workloads is presented. Insights from this exploration reveal the need to manage performance, energy and temperature in a coordinated fashion. Furthermore, the concept of "effective heat capacity", the heat generated beyond which no further gains in performance are observed with increases in voltage and frequency in the compute logic, is established as a useful metric. This characterization study also opens up multiple directions for further research.

Optimization: In light of the insights generated from the SoC characterization, this thesis demonstrates a software controller designed to maintain application performance while minimizing energy consumption on a Nexus 6 smartphone [37]. The controller is based on the work presented in [38] and exploits (a) techniques from feedback control theory and mathematical optimization and (b) the in-depth understanding of the SoC components and its governors. It chooses a combination of CPU frequency and memory bandwidth *simultaneously*, thus making it an *energy-efficient coordinated control* scheme. The novelty of this work is that it, (1) successfully demonstrates a coordinated controller, (2) is generic enough to incorporate other SoC components into the ambit of the problem and (3) indicates the viability of application-specific controllers on SoC platforms.

Since the dependence on application-specific offline data limits the extent and effectiveness of such a controller, an improvement/extension is explored as well. Simple analytic models for the performance and power of the processor and memory are developed which are subsequently used by the feedback controller at run-time to optimize energy efficiency. Using the parameter (ops/byte) measured at run-time, the controller classifies the application as compute bound or memory bound or a mix of the two and selects the processor and memory frequencies that minimize the energy-delay-product. A similar approach is explored for a multi-core multi-memory-controller system.

Neither of the works described in the previous two paragraphs consider temperature in the problem formulation. This thesis explores Dynamic Thermal Management (DTM) on Chip MultiProcessors (CMP)s using feedback controllers. An adjustable gain integral controller is demonstrated in [39] to regulate the temperature of each core in a CMP to a fixed value by performing per-core DVFS. The controller is tested on a cycle-level simulator running PARSEC and Splash2 benchmarks. An important insight gained from this work is the connection between workload characteristics and thermal behavior of the core. Specifically, the wastage of energy in the form of heat is clearly observed during memory intensive phases of the workload. The concept of "effective heat capacity" is strongly motivated by this phenomenon.

The final part of this thesis tackles this very issue in a 3D stacked processor-memory

architecture where its effects are exacerbated. As opposed to prior works which consider temperature as a constraint to be met, this thesis advocates using temperature as a *resource* just like compute or memory cycles. The characterization experiments provide ground rules for the design of a real-time, numerical optimization based, application agnostic controller, TRINITY, for intelligently managing performance, energy *and* temperature. TRINITY achieves up to 11% improvement in energy-delay-product (EDP) over heuristic schemes. On an average, TRINITY reduces temperature by 4 Kelvin for similar EDP. An analysis of device reliability shows an increase by up to 26% on account of reduced temperature.
In summary, the contributions of this thesis are as follows:

- Performance characterization of current power management strategies in SoCs demonstrating the scope for coordinated power management.
- A control-theoretic solution to the coordinated management of the core and the memory to minimize energy consumption for a target performance level.
- Extending the coordinated control framework to multi-core multi-memory-controller systems to improve energy efficiency.
- A distributed feedback controller to regulate core temperatures in a 2D multi-core processor.
- A comprehensive characterization of a 3D stacked processor-memory architecture.
- A distributed coordinated control framework for performance, energy and thermal management in a 3D stacked processor-memory architecture.

1.2 Thesis Organization

Apart from CHAPTER 1, this thesis comprises of the following 9 chapters:

CHAPTER 2 begins with a brief discussion of the evolution of microarchitecture from planar 2D to 2.5D to 3D stacked architectures. The issues associated with each architectural design and the limitations of the solutions adopted emerge out of a detailed literature survey. The state-of-the-art power, energy and thermal management techniques in the domain of SoCs and traditional server-class processors are described in detail. Their associated shortcomings are highlighted which forms the basis of this thesis. It concludes with a list of challenges that are addressed in the subsequent chapters.

CHAPTER 3 presents characterization of performance, power and energy consumption in an SoC using a commercially available mobile phone as a platform for exploration. It describes in detail, the interaction between a multitude of software and hardware optimizations on a wide range of target applications. It also highlights the need to coordinate the power management between the processor and memory. This, and more insights are summarized in the end forming the basis for the next chapter.

In CHAPTER 4, a feedback controller is developed which minimizes energy consumption while maintaining a performance target. Experimental results obtained from a mobile phone are discussed in detail. The results confirm the need for coordinating processor and memory clock frequencies to trade-off energy and performance. The chapter concludes with a summary of results and shortcomings of the presented approach.

CHAPTER 5 extends the feedback controller framework presented in CHAPTER 4. This chapter proposes an application agnostic energy optimization approach for a more generic architecture. Starting from a single-core single-memory controller model, the concept is extended to explore a multi-core multi-memory-controller system. The chapter concludes with a discussion of the results and also prospects for future implementation on a real system.

CHAPTER 6 focuses on the role of temperature, the third key parameter in this thesis. Using a cycle-level simulator coupled with power and thermal calculations, a distributed feedback controller designed to regulate temperature in a 2D multi-core processor environment is described. Experimental results are presented and key observations regarding the potential inefficiency of such an approach in thermally constrained 3D stacked processor-memory architectures are also mentioned.

CHAPTER 7 analyzes the interaction between performance, processor-memory interactions and thermal coupling in a 3D stacked architecture. Insights from the previous chapter motivate the development of the concept of “effective heat capacity”. This chapter lists key insights which open up many possibilities for future research.

CHAPTER 8 presents details of a feedback controller (TRINITY) which manages performance, energy and temperature of a 3D stacked processor-memory architecture. The power, performance and temperature models used in the controller are described in detail. An in-depth analysis of experimental results shows the benefits of TRINITY over existing state-of-the-art techniques.

Finally, CHAPTER 9 summarizes the main contributions of this thesis and discusses potential future research directions.

Collectively, through the different contributions presented in this thesis, the following aspects are substantiated: (1) The need to manage the processor and memory in a coordinated fashion for current and future architectures, (2) The benefits of control theoretic approaches over heuristics, and (3) The importance of managing temperature as a resource similar to compute or memory cycles. Furthermore, this thesis identifies a set of future research directions pointing towards to innovative inter-disciplinary solutions.

CHAPTER 2

PROBLEM FORMULATION AND RELATED WORK

This chapter begins by describing the "Memory Wall" problem and provides an overview of related research efforts that address this problem from architectural, software and hardware perspectives. A second important aspect, discussed in detail is the dynamic management of performance, power, energy and temperature of processors and memory. Finally, this chapter concludes with a list of open problems that stress the need to optimize energy efficiency for compute and memory in a coordinated fashion.

2.1 The Memory Wall

The rate of increase in microprocessor speed and memory speed have been following an exponential curve. However, the exponent for the microprocessor has been larger than that for the DRAM (See Figure 2.1). The *memory wall* is described as a situation where the rate of increase of processor speed as compared with that of the DRAM will eventually lead to processor speed improvements being masked by the DRAM. Quoting an example from [40], consider the following equation:

$$t_{avg} = p \times t_c + (1 - p) \times t_m \tag{2.1}$$

where p is the probability of a cache hit, t_c is the cache access time, t_m is the DRAM access time and t_{avg} is the average time to access the memory. The above equation implies the following: Suppose 20% of the instructions reference the DRAM, if t_{avg} exceeds $5t_c$, the performance of the processor is entirely determined by t_m. Consequently, the queuing delays grow (t_{avg} continues to increase) and the processor performance hits a wall i.e. the *memory wall*.

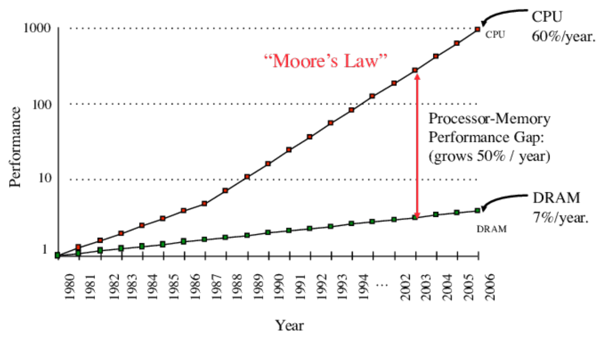


Figure 2.1: Memory Wall Problem [41]

Researchers have addressed this issue at multiple levels. The details can be found in the technical report [42] and references therein. Some of the efforts aimed at improving the DRAM are access latency reduction, increasing bandwidth, latency hiding via aggressive prefetching, non-blocking caches etc. Improving cache performance however, is not a very obvious method to handle the memory wall because the DRAM access speed will continue to be the critical bottleneck. Nevertheless, reducing the number of requests between the last level cache and the DRAM can potentially improve system performance. Optimizing cache size and cache associativity for reducing cache misses, memory compression etc. are a few techniques listed here among several others. Simultaneous Multi Threading (SMT) and Chip Multi Processors (CMP) try to hide the DRAM latency by executing available work on a thread while the other thread waits for the memory request. SMT can keep a single processor busy whereas a CMP can have better overall system throughput even though some cores remain idle.

The US Department of Energy predicts that for an exascale system with 100,000 nodes, the memory size, bandwidth and hierarchy are some of the important challenges [43]. The steep performance demands ($\geq$ 1 ExaFlop/s) are expected to be met subject to a cap on the maximum power consumed ($\leq$ 20MW). As mentioned in CHAPTER 1, the DRAM is projected to consume 70% of this total power. To address this issue, device manufacturers proposed a solution: 3D and 2.5D die stacking of compute logic and memory [44]. Accordingly, 3D die stacked memories such as HBM [45] from JEDEC and HMC [46] from Micron were introduced which offered an order of magnitude better bandwidth than traditional DDR technologies. These hardware and architectural innovations are primarily aimed at meeting the performance target. Dynamically managing the power consumed by the processor and the memory under a variety of workloads has been an active research topic for the past two decades.

2.2 Power, Energy and Thermal Management of Processors and Memory

Power and energy management strategies have been extensively explored ever since the advent of the very first series of computers. In the last decade, with the proliferation of cloud technologies, thermal management too is receiving a great deal of attention. A number of technological innovations have been proposed that span the entire hardware-software stack: from the logical circuit to complex applications. Many have become industry standards.

2.2.1 Dynamic Voltage and Frequency Scaling (DVFS)

At the circuit level, DVFS is a widely used technique which has been implemented with multiple design objectives. The basic principle of DVFS emerges from the dynamic power equation $P \propto V^2 \cdot f$, where V is the supply voltage and f is the clock frequency. Reducing voltage leads to quadratic power savings whereas lowering the frequency reduces the power linearly. Modern processors and even peripheral devices such as RAM, GPU etc., support DVFS. Linux provides OS-level support for CPU-DVFS through a subsystem known as *cpufreq* [47] and *devfreq* enables DVFS for other peripheral devices. The DVFS policies are called *governors*. In [48], different *cpufreq* governors provided by Linux are tested for a variety of applications. The power, performance and energy numbers are compared for each of the CPU governors. On mobile systems running the Android OS as well, DVFS policies for the CPU and other subsystems are inherited from the Linux kernel designed for servers. The two main objectives are: (1) energy/power saving and (2) meeting task deadlines. DVFS has benefits in reducing power and energy but it does so at the cost of performance. In reference [49], the authors investigate the effects of DVFS on power, energy and performance on a Pentium III and a PowerPC system. DVFS algorithms implemented on processors available in the market today are described in references [50, 51, 52, 53, 54]. The effects of DVFS are also evaluated on three generations of AMD processors in [55], wherein the authors claim that DVFS will have diminishing returns for future processors and memories. Nevertheless, reaffirming the efficacy of DVFS based techniques, the more recent survey article [56], reviews among other approaches, recent research works which perform DVFS to reduce power consumption on embedded processors.

Optimization based approaches aim to minimize (i) system wide energy consumption under performance bounds [57], (ii) energy-delay^2 [58], (iii) power-per-watt [59], (iv) power budgets with temperature constraints [60]. More exotic schemes use game theory for maximizing performance under power budgets [61], machine learning to efficiently share system resources for performance maximization [62], balance resource utilization and fairness in a CMP using market based strategies [63, 64] and [65] finds a Pareto optimal per-core configuration by integrating multiple power management techniques. On mobile systems, power/energy consumption is a top priority. Accordingly, a model-based DVFS governor for Android systems is presented in [66]. At first, offline profiling is performed on a set of benchmarks and for each benchmark the critical speed (CS), i.e., the energy-optimal CPU frequency is obtained along with the corresponding memory access rate (MAR), which in turn is obtained from the hardware performance monitoring unit. Statistical methods are then used to derive a model for CS with regard to MAR. This model is called the MAR-based CS Equation, or MAR-CSE. A DVFS governor is created that uses MAR-CSE to select the optimal CPU frequency based on the run-time MAR values. This approach is application-agnostic in the sense that it is independent of the running application. Furthermore, it is designed to optimize energy without considering performance. Another work in the same domain is the POET system [38]. It minimizes energy consumption of an application while attempting to meet its performance requirement. The system requires two inputs before it starts: a performance target, and performance and power data for different system configurations. At run-time, it repeatedly measures the actual performance, and uses feedback control and linear programming to select energy-optimal configurations that meet the performance target. POET consists of a C library and a run-time system, and is designed for traditional embedded systems with soft real-time constraints. Because of the diversity of such systems, one of the key design goals of POET is portability. The problem it tries to solve is to create an application and platform-independent resource allocation framework. The advantage of [38] over [66] is the inclusion of performance constraint within the ambit of the problem. CHAPTER 4 is based on [38] for this very reason.

As opposed to optimization, researchers have implemented simpler PID based techniques for power budgeting via DVFS [67, 68, 69]. Power regulation [32, 70] and instruction throughput regulation [71] using adaptive gain integral controllers has also been explored on traditional desktop/server class processors. Benefits of using adaptive gain feedback controllers versus fixed gain controllers are: (i) robustness to modeling errors and (ii) rapid convergence. Fixed gains can lead to sluggish response or worse, oscillations. While most approaches tend to be application-agnostic DVFS methods, the authors in [72] present a DVFS algorithm for a specific application: video decoding. Application specific techniques can outperform their agnostic counterparts but at the added cost of switching between multiple policies depending on the application type.

Modern manufacturing technologies, while allowing for faster processing, have the unavoidable effect of increase in static power dissipation as well. Researchers have proposed Dynamic Thermal Management (DTM) strategies for the CPU and chip packages based on (i) power gating, (ii) reducing instruction fetch rate, (iii) thread migration, (iv) DVFS and (v) external cooling. What began as heuristic approaches eventually gave way to more formal control-theoretic solutions. References [31, 30] use PI and PID controls to slow down the rate of the instruction-fetch unit whenever the temperature exceeds a given upper bound. Stressing the importance of managing temperature effectively, the work in [73] points out that minimizing thermal impact extends the sustainability of desired Quality-of-Service levels on mobile devices.

In contrast to power gating, DVFS allows for reducing the temperature by simply reducing the dynamic power dissipated. Reference [74] applies DVFS for temperature control when the temperature hits an upper bound. Capitalizing on the drawbacks of threshold based control, researchers have proposed more advanced and rigorous formulations for DTM using techniques from the domain of optimal control and numerical optimization [35, 34, 36, 75]. These works however, assume linear and time-invariant plant-models for their respective control systems; [34, 75] updates the model on-line while [35, 36] do not. DTM under soft and hard real-time constraints has been investigated in [76] and [77], respectively.

Power and thermal management are not limited to the CPU alone. The survey paper [78] classifies and describes several DRAM power and thermal management techniques. Recently, researchers have started to focus on the DTM of 3D stacked memories as well. Due to the 3D architecture, higher power densities lead to larger thermal gradients, localized heating and heat shielding. The state-of-the-art for the DTM of 3D stacked memories is evolving along the lines of CPU-DTM strategies. The approaches vary from architectural level changes [79] to applying numerical optimization to maximize instruction throughput under strict power and thermal constraints [80]. Some other approaches involve data compression at the memory controller [21], two level prefetching with throttling off-chip memory links [20], dynamic page allocation [81], DVFS [82], thread migration [83] and data block reallocation with heterogeneous memory architectures [84].

3D stacking of memories has been explored academically [12] and also by the industry [7, 19, 46] for performance improvements. Memory intensive applications can benefit greatly due to availability of increased bandwidth as predicted by [85]. PIM architectures are being proposed to further increase performance by offloading memory-intensive code to the logic die of the 3D memory. Some researchers have characterized the performance of 3D processor-memory systems [79, 86, 87] under a variety of benchmark applications. Reference [79] goes onto propose a new 3D memory architecture specifically suited for SoCs and compares it against 2D memory systems, whereas [87] explores the possibility of controlling CPU and 3D memories simultaneously. The authors in [3] explore the viability of using the execution unit of a General Purpose GPU (GPGPU) as the in-memory processing element. An important observation they make is that exascale computing workload may benefit from PIM; PIM itself does not require exascale workloads in order to be useful. In their experiments, they observe a tremendous reduction in energy-delay-product (85%) at marginal performance improvements (7%). However, they design their experiments such that power and thermal emergencies are not triggered. [88] proposes moving the hottest datapaths closer to the heat sink (thermal herding) for better heat dissipation. More recently, the use of thermal TSVs to extract heat from the different layers has been proposed [22], wherein the authors propose boosting the performance of applications by exploiting the improved cooling efficiency.

2.2.2 CPU Idling

To further optimize power consumption, when certain components in the processor or the SoC are not being used, it is a common strategy to either power down (clock gating) or enter different levels of sleep modes. Linux calls it *cpuidle* and different processor manufacturers provide drivers which implement idling policies. Reference [48] explores (i) the benefits of the stock *cpuidle* manager and (ii) the effect of *cpuidle* on compiler optimizations and CPU governors. They find that idling helps in saving up to 19% energy when combined with other mechanisms such as CPU DVFS. CPU idling has been inherited by Android systems with `msm_idle` as the default driver on Qualcomm chipsets. CHAPTER 3 discusses the interaction between CPU DVFS and CPU idling.

2.2.3 Microarchitecture Optimizations

Moving up the hardware stack, at the architecture and microarchitecture levels, there has been a clear shift of design goals from being performance-centric to power-aware. Many designs at these levels target power efficiency of major hardware components such as memory [89], cache [90], etc. Other methods include clustered DVFS [91], heterogeneous computing that incorporates CPUs and accelerators such as DSPs and GPUs [92], asymmetric cores such as the big.LITTLE architecture from ARM [93] and many more. For the work presented in this thesis, a specific microarchitecture is assumed. The control techniques presented in the subsequent chapters are agnostic to the changes in the microarchitecture.

2.2.4 OS, Compiler Optimizations

Commensurate software improvements have complemented hardware optimizations. An important problem at the OS level that has been extensively studied is power-aware thread/task scheduling, not only in a single core [94] but also across cores [95]. [96, 97] schedule threads (computations) from hot cores to cooler cores in an effort to maintain a balanced thermal field. More recently, Linaro [98] announced an Energy Aware Scheduler which aims to improve power management on ARM processors (homogeneous and heterogeneous like big.LITTLE). They schedule tasks intelligently on to cores so as to minimize energy consumption. Above the OS level, compiler optimizations however have focused on performance improvement [99]. Improved performance generally leads to better energy efficiency due to shorter execution times as noted in [100]. Furthermore, power-aware compiler optimizations have also been studied [101]. To truly understand the ramifications, [102] explores the design space of the embedded `gcc` compiler to identify the optimization options that offer maximum reduction in energy. The authors of reference [48] compare the power, performance and energy trends of different optimization levels of the `gcc` compiler on a server system. Multithreaded programs can exploit inherent parallelisms in applications and [103] compares energy efficiency of two parallelism technologies: parallelization and vectorization. At the programming language level, various aspects for writing energy-efficient programs including programming framework, language extensions and so on have seen some research activities [104, 105].

Except for [87, 62, 64], rest of the works listed in this section either implement independent strategies for the CPU and memory or control only CPU parameters such as clock frequency, L1 cache banks, Re-Order Buffer size etc. CPU-only methods do not consider memory boundedness of applications whereas, memory-only schemes are concerned with meeting bandwidth or latency requirements. Some DTM schemes consider static power dissipation as a constant and do not take into account heating from neighboring components. Software optimizations at the application layer focus on performance and ignore power/energy effects in the lower layers of the hardware stack.

2.3 Coordinated Management of Processor and Memory

The idea of controlling multiple components simultaneously has been explored extensively for embedded systems. In [23] the authors point out that independent control policies may run into conflicts with each other leading to oscillations, thereby significantly reducing the effectiveness in achieving energy savings. They demonstrate "CoScale" on a server that performs coordinated control of CPU and memory, where the goal is to minimize energy consumption while staying within a user defined performance degradation bound. Using performance and power models and also aided by hardware counters, CoScale uses a gradient-descent-like algorithm to select the optimal configuration. For a mobile device platform, [26] evaluates energy-performance trade-offs when implementing DVFS on the CPU and the memory bandwidth. They run SPEC CPU2006 benchmarks on a `gem5` simulator and conclude that capturing the complex interplay between the CPU and memory subsystems is essential and that coordinated strategies can deliver higher energy efficiencies. On an Android phone, [27] demonstrates a coordinated control of the CPU and GPU for saving power. Based on power and performance models generated offline, they design an algorithm to keep the performance within a predefined range while consuming the least amount of power. They achieve up to 58% power savings by performing CPU-GPU DVFS simultaneously. The authors in [106, 107] demonstrate heuristic schemes to balance performance and power between the CPU and GPU on AMD's Accelerated Processing Units. In [24], the authors propose "Harmonia", which balances computation and memory power on a high performance GPGPU. This is a heuristic based algorithm and is designed for server workloads, which generally tend to be memory intensive. MemScale [108] proposes applying DVFS to the memory controller and memory channels in order to reduce energy consumption while staying within performance loss bounds. Finally, the authors in [109] present a hardware control system implemented on an FPGA for GPUs that manages, (1) number of streaming multiprocessors (SM), (2) number of warps/wavefronts, (3) frequency of the SM and, (4) the DRAM frequency.

2.4 Summary

This chapter discussed previous research efforts related to the dynamic management of power, temperature, energy and performance of a computing system. Most prior research has focused on processor management. As the memory power consumption started becoming comparable to their processor counterparts, multiple techniques were developed to increase the efficiency of the memory subsystem. However, state-of-the-art processor and memory power management techniques were, and continue to be, independent of each other. Coordinated control of multiple components has started receiving interest only recently. However, the techniques developed thus far are either platform centric or application-specific. Furthermore, almost all approaches are based on heuristics. Control theoretic schemes are slowly gaining steam. Additionally, since the industry is moving towards 3D stacking of silicon die, there is a need to fundamentally understand the multi-physics interactions between performance, energy and temperature. Heuristics based methods fail to satisfy all the necessary requirements. This thesis substantiates the lacuna in heuristics-based approaches and develops control theory based solutions to manage processors and memories in a coordinated fashion.

CHAPTER 3
PERFORMANCE, POWER AND ENERGY CHARACTERIZATION: MOBILE DEVICES

Energy-efficient high performance mobile computing is receiving increasing attention in recent years. Mobile device manufacturers are continuously striving to provide the best user experience at the lowest energy consumption. Multiple, yet independent research efforts, have focused on individual layers of the hardware-software stack (operating systems, compilers, microarchitecture etc.). While providing better performance individually, energy/power saving mechanisms may sometimes not lead to complementary benefits when employed in tandem. A global understanding of interactions between these techniques can help in developing strategies at different layers that supplement each other. In this chapter, a collection of energy management schemes are evaluated on a Nexus 6 and a Nexus 5X phone at both the hardware and software levels. Their effects on the following three parameters: 1) Average performance 2) Average power and 3) Total energy, are compared. The goal is to help designers better understand interactions of these methods for more innovative solutions.

3.1 Overview

The performance of smartphones has been constantly improving. However, high performance comes at the cost of high power consumption and faster battery discharge, which can negatively impact user satisfaction. To tackle this issue, numerous studies, on both the hardware and software sides, have been conducted in an effort to make smart devices more power/energy efficient. However, research efforts in this direction have more or less investigated each issue independently. There is rarely a common platform, in either hardware or software, and there is a lack of comparative studies which consider effects of a new technique on existing technologies. The problem of multiple platforms is exacerbated by the continuously changing landscape in the smartphone industry. Furthermore, there are variations in the reported results; while some report average power, others report average energy. Comparing two methods which have the same goal yet very different experimental platforms is a futile exercise.

To better understand the benefits of a particular power/energy management method, one must carefully analyze the cross effects between different layers of the stack. A newly developed method tested in isolation, might offer high power savings but deliver little gain when working on a full system with other kinds of mechanisms. This type of survey has been performed on a server [48], where the authors put the system under a series of rigorous tests to determine which mechanisms work well while some schemes fail. Drawing inspiration from their work, the work presented in this chapter performs extensive testing of a collection of energy/power management techniques on a smartphone and compares the effects on (1) Average Performance (2) Average Power and (3) Total Energy. The goal is to help a hardware/software designer understand the nature of interactions between different layers of the stack so that they can develop even better energy/power management strategies.

A set of widely used optimization techniques is tested, some designed explicitly for energy efficiency while others created mainly to improve performance but with side effects on energy consumption. Specifically, these include compiler optimizations, multithreading, CPU idle states, dynamic voltage and frequency scaling (DVFS) on CPU and GPU, and memory bandwidth scaling. In particular, two scenarios are evaluated that each involve two optimization techniques to understand their interactions: (1) CPU DVFS and memory bandwidth scaling and (2) CPU DVFS and CPU idle states. The experiments are performed on two Android devices: a Nexus 6 and a Nexus 5X, and the test programs include 5 popular Android apps such as AngryBirds, and 4 prominent Android benchmarking applications such as AnTuTu. For further insights and to highlight differences from servers, single and multi-threaded benchmarks from SPEC CPU2006, PARSEC, respectively, are also tested. In total, the test suite includes 367 test cases.

Some of the important findings of this work are listed below:

1. The default DVFS governors can be optimized for better energy efficiency.
    (a) CPU governors must include energy consumption and not just performance or power in their design.
    (b) A DVFS mechanism which can adapt to the behavior of individual apps can possibly provide higher energy efficiency with the same performance.
    (c) CPU and memory bandwidth governors working in collaboration can lead to better energy savings.
2. Smartphone vendors may want to consider customizing certain aspects of power management.
    (a) The *performance* CPU and GPU governor consumes excessive power and results in severe thermal throttling. Vendors should carefully decide if it should be avoided.
    (b) CPU idling, at least on the Nexus 5X, does not seem to have much effect on the power and energy consumption for real-world Android apps.
3. Developers may want to note that increasing the number of threads for a program while improving performance does not save energy, contrary to the trend on a server.

3.2 Methodology

This section describes the experimental platform, the software benchmarks and the different power/energy management techniques tested. It also explains the rationale for the benchmarks and options chosen along with the base configurations against which the results from different options are compared.

3.2.1 Experimental Testbed

The experiments are performed on the Motorola Nexus 6 and LG Nexus 5X hereafter referred to as *N6* and *N5X* respectively. The *N6* runs on a Qualcomm Snapdragon 805 chipset which has a quad-core Krait 450 CPU and Adreno 420 GPU with 3GB of RAM. In all the experiments conducted, the *N6* has Android 6.0 Marshmallow operating system with a Linux kernel v3.10. *N5X* comes with a Qualcomm Snapdragon 808 SoC comprising of a quad-core ARM Cortex-A53 and a dual-core ARM Cortex-A57, Adreno 418 GPU and 2GB of RAM. The *N5X* runs Android 6.0 Marshmallow operating system with a Linux kernel v3.10. Section 3.3 describes experiments on the following: CPU governor, memory bandwidth governor, GPU governor, CPU idle, threads and parallelization and compiler optimization. The *userdebug* build on *N6* and *N5X* is required to enable changes to the *cpufreq*, *devfreq* and *cpuidle* modules in the Linux kernel. The Monsoon Power Monitor [110] is used which measures power consumption of the whole phone i.e. power values are inclusive of the CPU and other modules as well. Power samples are collected every 2ms and to communicate with the phone, Android Debugging Bridge [111] (`adb`) runs on a host machine. Since the default setting on the phone is to start charging when connected to a computer via USB, USB charging is turned OFF to avoid bias in the power measurements.

Table 3.1 lists all the 25 benchmark applications tested in this work. They are into 4 groups: (G1) Android benchmarks (G2) Android apps (G3) PARSEC and (G4) SPEC CPU2006. Each benchmark in the experimental survey is chosen with a specific intent and the intention is to touch as many features of the SoC as possible.

First of all, a few commonly used Android benchmark applications are chosen which test the performance of the entire SoC. The set includes AnTuTu v6 [112], Geekbench3 v3.4.1 [113], Vellamo v3.2 [114], and 3DMark [115]. AnTuTu performs tests on CPU, GPU, RAM and I/O. GeekBench tests the CPU and RAM. Vellamo offers web browser, single core and multicore benchmarks. Only the browser feature of this benchmark app is tested in order to prevent repetitions of single and multicore benchmarks. Finally, 3DMark

Table 3.1: List of benchmarks suites.

Benchmark suite		Applications/Tests
G1	AnTuTu	CPU, GPU, I/O, UX,
	Geekbench3	Single and multi-core CPU benchmarks, STREAM memory benchmarks
	Vellamo	HTML5 browser benchmarks
	3DMark	Sling Shot using OpenGL ES 3.1 (Physics and Graphics tests)
G2	Android Apps	Facebook, AngryBirds, Chrome Browser, MX Player, Spotify
G3	PARSEC 3.0	`blackscholes, facesim ,ferret, freqmine, fluidanimate, streamcluster, swaptions`
G4	SPEC CPU2006	`bzip2, h264ref, hmmer, mcf, perlbench, xalancbmk, namd, calculix, sphinx3`

is a specialized benchmark that runs physics tests and 3D graphics tests for benchmarking the CPU and GPU respectively. For experimental rigor, each of the benchmark apps mentioned in this paragraph is run 5 times in every configuration.

Next, there are 5 popular Android applications: Facebook, AngryBirds, MX Player, Spotify and Chrome browser. The Facebook app is tested with the help of RERAN [116] for repeatability. Although RERAN is an established record-and-replay tool for Android, timing issues were observed when changing CPU governors and hence it could only be used on Facebook. A 30-second sequence of actions like scrolling, changing tabs etc. is recorded and the same sequence is replayed during experimental runs. AngryBirds is a video game, and to test it, the game is manually played for a duration of 60 seconds. MX Player, a video player, is tested by playing a 137-second long HD video. Spotify is a music streaming app, and it is tested by playing songs from a playlist for 60 seconds while changing songs manually every 10 seconds. Finally, the Chrome browser is tested with the Mobilebench Browser Benchmark (MBB) [117] by loading a series of web pages and performing automatic zooming and scrolling actions to mimic a real user. Every configuration is tested 5 times for the apps except for Facebook which requires 10 repetitions.

Additionally, programs from the traditional PARSEC [118] and SPEC CPU2006 [119] benchmark suites is also included. While these are not the ideal benchmarks for a mobile platform, one of the goals with this work is to understand the key differences between SoCs and servers. Also, these programs have very different characteristics from the Android programs. Testing them allows the tests presented here to cover a wider range of behaviors and to gain more insights. The PARSEC suite consists of multithreaded programs, and is a standard test suite for parallel computing. A subset of the benchmarks is chosen with the 'simlarge' input set and each benchmark application is run 10 times in each configuration. SPEC CPU2006, on the other hand, consists of single-threaded programs. They have been used to test mobile CPUs as seen in [17] and references therein. A set of 9 benchmarks are selected of which 6 are integer and 3 are floating point programs. Each of these 9 benchmarks is executed 5 times for every configuration and the input size is set to 'train'. The data set sizes are so chosen to reduce duration of each experiment and also to prevent thermal throttling.

3.2.2 Hardware and Software Testing Options

Table 3.2: Table listing different options tested. ✓indicates the tested option for the benchmark group. *Except 3DMark.

Testing Options	Android Bench. (G1)	Android Apps (G2)	PARSEC (G3)	SPEC (G4)
Compiler Opt. (O0-O3)			✓	✓
Threads (1,4,16)			✓	
cpufreq governor	✓*	✓	✓	✓
devfreq CPU-DDR BW gov.	✓*	✓	✓	✓
devfreq GPU governor	3DMark			
CPU-BW cross effect		✓	✓	✓
cpufreq-cpuidle cross effect		✓	✓	✓

On the Android platform, the software and hardware stack supports various power/energy management options. Attempting to study the effect of each setting on one another will result in a large set of combinations and will not be amenable to interpretations. Therefore, the number of tested configurations is reduced by forming 6 groups. Table 3.2 lists the hardware and software options tested in this survey.

Compiler Optimizations: Both PARSEC and SPEC CPU2006 benchmarks are cross-compiled for ARM with optimization levels `O0-O3`.

Threads: Of all the benchmarks, only PARSEC explicitly supports changing the number of threads. The tests include $1, 4$ and 16 threads for the PARSEC benchmarks listed in Table 3.1.

The only hardware options tested are the ones that are exposed through the software, i.e., the *cpufreq, devfreq, cpuidle* modules.

cpufreq: This module in the Linux kernel enables DVFS for the CPU [120]. In the Linux kernel v3.10, *cpufreq* supports the following governors:

1. `performance`: It sets the CPU frequency to the highest supported value.
2. `powersave`: It sets the CPU frequency to the lowest supported value.
3. `ondemand`: This *cpufreq* governor implements DVFS by tracking the CPU load and ramps up to the maximum frequency when the load crosses a certain threshold.
4. `interactive`: This governor is similar to `ondemand` but is designed to be more aggressive in ramping up the CPU clock frequency to be more responsive. This is the default CPU governor on both *N6* and *N5X*.
5. `userspace`: This governor allows the root user to set the CPU clock frequency to a specific value.

All the above governors are tested except `userspace`, because `performance` and `powersave` can be considered two special cases of `userspace`.

devfreq: This kernel module enables DVFS in devices other than the CPU. For example: In *N6*, *devfreq* supports GPU clock frequency scaling and CPU-DDR bandwidth scaling. The *devfreq* governors for the GPU are `msm-adreno-tz` (the default), `performance`, `powersave` and `userspace`. The `msm-adreno-tz` governor is very similar to the `ondemand` CPU governor but is tuned towards performance. Experiments with the `performance` governor resulted in current surges leading to the mobile phone turning OFF. Therefore, results are presented for only `msm_adreno_tz` and `powersave`.

Memory bandwidth governor: The *devfreq* module also allows the memory bandwidth to be varied and the governors supported are: `cpubw_hwmon` (the default), `performance`, `powersave` and `userspace`. The `cpubw_hwmon` monitors the hardware counters and implements an exponential back-off algorithm when the memory activity starts to decrease. The governors tested here are `cpubw_hwmon`, `performance` and `powersave`.

cpuidle: This module manages the CPU idle states or C-states [121]. There can be multiple idling states which get triggered based on CPU (in)activity. This module is **not** supported on the *N6* but is available on the *N5X*. Hence **only** the results related to *cpuidle* on the *N5X* are presented. The default *cpuidle* driver is called `msm_idle`. The efficacy of this driver is evaluated by comparing the results for when it is enabled and disabled.

3.2.3 Other Factors

Background Interference: During the experiments, GPS and screen-rotation are turned OFF. The phone does not have a SIM card and is laid flat on a table left undisturbed. This helps in preventing extraneous peripheral devices from drawing power during the experiments. Benchmark apps and Android apps require a screen touch to begin the test and hence the screen is left ON throughout the benchmark duration with the lowest (fixed) brightness setting. Furthermore, some apps require WiFi to be ON and hence WiFi is left turned ON for all the Android and benchmark apps. For PARSEC and SPEC however, the screen and WiFi are turned OFF during the experiments.

Performance Metric: The notion of performance varies across the set of benchmarks. Performance of PARSEC and SPEC is measured in seconds (execution time) whereas performance for each benchmark app is derived from the figures reported on the app.

Performance metrics for Android apps are as follows: AngryBirds and Facebook: Seconds-Per-Frame (SPF, i.e., FPS^{-1}), MX Player: seconds (execution time), Spotify: (Giga-Instructions-Per-Second)$^{-1}$ i.e. GIPS^{-1}, Chrome: seconds (warm page load time).

3.3 Results

The results are divided into different categories and the trends of performance, power and energy for each benchmark group are explained. For all the plots, lower the value, better is the performance, power and energy.

The *N5X* is used **only** during CPUIdle experiments whereas *N6* is used for the rest. The results presented are in general normalized. The averages are therefore simply arithmetic means of the normalized values.

3.3.1 Compiler Optimizations and Thread Level Parallelism

Reference [48] presents a detailed study of compiler optimization options in relation to energy consumption on a server. In this work, the effect of compiler optimizations on program performance, power, and energy on a smartphone is examined. Using PARSEC and SPEC benchmarks, the `gcc` compiler's optimization levels `O0` through `O3` is tested with all the other system settings such as CPU governor set to the default. PARSEC benchmarks are run with 4 threads. Figure 3.1 shows the performance, power and energy numbers obtained for different optimization levels for PARSEC and SPEC. All results are normalized with respect to `O3`.

As expected, there is a big performance difference between `O0` and `O1`. Compared with `O0`, `O1` produces an average performance (execution time) improvement of $1.5\times$ and $2.2\times$ for PARSEC and SPEC respectively. However, beyond `O1`, the performance difference is very small for both multithreaded PARSEC and single-threaded SPEC.

The second observation is that optimization levels make very little difference in power consumption. Although `O0` generally leads to the highest power consumption, the average difference between `O0` and `O3` is only $0.039\times$ and $-0.03\times$ for PARSEC and SPEC respectively. As a result, the energy difference as shown in Figure 3.1 is mostly due to the performance difference. These trends are very similar to those found on servers as reported in [48].

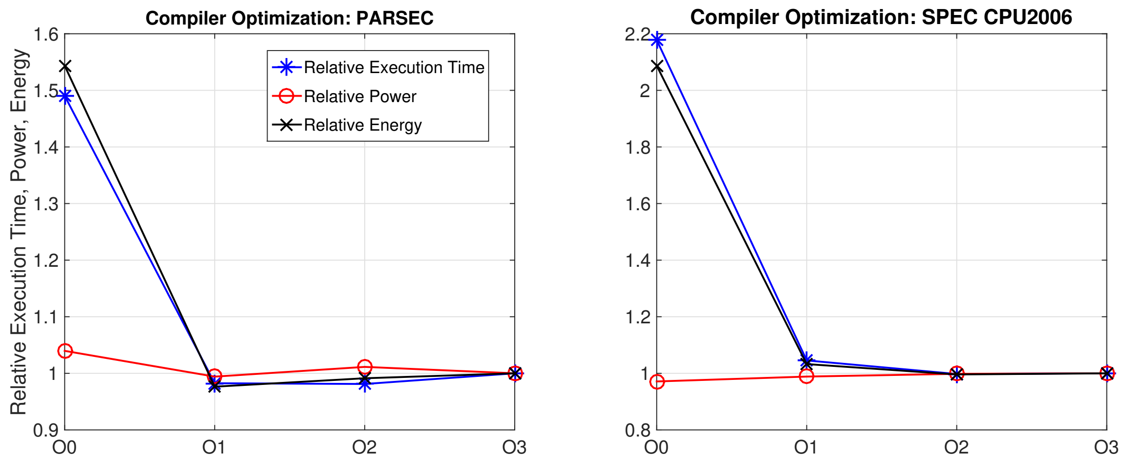


Figure 3.1: Effect of compiler optimization on performance, power, and energy on *N6*. Normalized w.r.t `O3`.

In the domain of mobile devices, $4, 6$ and 8-core SoCs are becoming the norm. While parallelization improves performance, due to increased CPU activity, the side effects on power and energy have to be analyzed carefully.

Figure 3.2 shows the results obtained when the effects of parallelization is evaluated by varying the number of threads from 1 to 4 to 16 for 7 PARSEC benchmarks as listed in Table 3.1. Results are normalized to 4 threads.

When the number of threads is increased from 1 to 4, five of the benchmarks show an average speedup of 3.62, while `freqmine`, a memory-intensive benchmark, has a speedup of 1.71, and `ferret` has a speedup of 1.99 due to small-sized L2 cache. Increasing the number of threads from 4 to 16, however, does not improve the speedup by a large margin because the *N6* has a quad core processor.

In [48], increasing the level of parallelization from $1 \rightarrow 4 \rightarrow 16$ threads reduces energy consumption by 45% and 59% respectively, because the magnitude of reduction in runtime

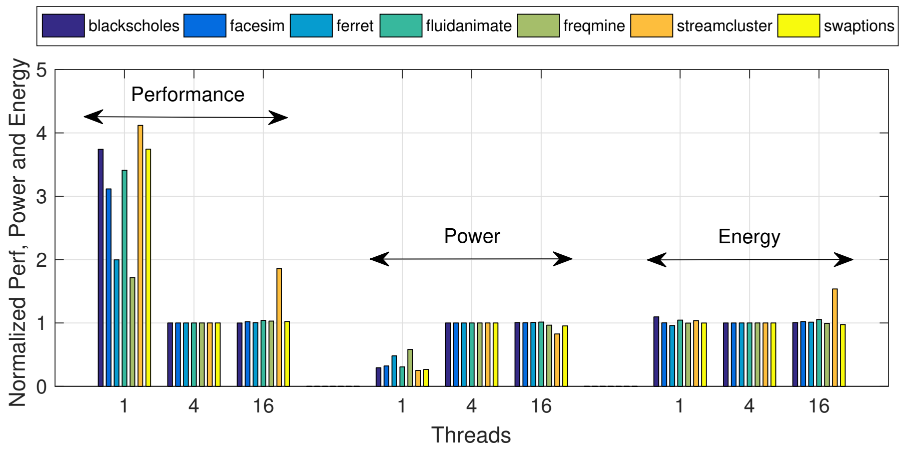


Figure 3.2: PARSEC Threads: Performance, Power and Energy on *N6*. Normalized w.r.t 4 threads.

surpasses increase in power. Experiments here show that the trends on mobile devices are quite different. The results reveal that increasing although the number of threads from $1 \rightarrow 4 \rightarrow 16$ threads, energy remains almost constant. The only exception is the 16-thread `streamcluster`, which has a 53% increase in energy, mainly due to the significant drop in performance (85%). As shown in [122], Android applications in general have relatively low TLP (< 3). However since Android 7.0 multi-window support allows for displaying more than one app on the screen. Through the experiments it is observed that programs which contain highly parallelizable CPU intensive code when allowed to run for long duration, can cause thermal throttling, which reduces performance significantly. If high CPU activity persists, it eventually leads to device shutdown. In conclusion,

OBSERVATION 1: It appears that, on both servers and Android mobile devices, the optimizations implemented by `gcc` *have little impact on a program's power consumption. Unlike on servers, multithreading on smartphones does not reduce energy consumption. Additionally, quad-core Android mobile devices are not designed to run applications with high thread-level parallelism (TLP), and developers should avoid writing such applica-*

tions.

3.3.2 CPU Governors

CPU DVFS is one of the most important power management techniques on smartphones. Four CPU governors shipped with the phone: `interactive`, `ondemand`, `powersave`, and `performance` are evaluated in this study. Due to thermal throttling issues, the `performance` governor is only tested on the Android apps. Results shown in Figures 3.3 (a), (b) and (c) are all normalized to `interactive`.

For PARSEC and SPEC, as expected, the `powersave` governor has the lowest performance – on average, the performance is more than 7$\times$ worse than `interactive`. However, it is interesting to note the very different energy results with the two benchmark sets. In every single case of the multithreaded PARSEC programs, `powersave` results in net energy savings compared with `interactive`, ranging 15.2% - 35.3% with an average of 28.5%. SPEC programs show opposite results. In every single case, `powersave` consumes more energy than `interactive` – 59.6% more on average. The `ondemand` governor, on the other hand, has almost identical behavior as `interactive` in terms of performance, power, or energy.

For the Android benchmarks, AnTuTu, GeekBench and Vellamo, the performance degradation of `powersave` is less marked than PARSEC and SPEC, but is still significant – 4.46$\times$ on average. Energy consumption results are mixed. With AnTuTu, `powersave` consumes about 40.7% less energy, while with GeekBench and Vellamo, it consumes 49.1% and 17.2% more compared with `interactive`. The `ondemand` governor produces mixed results compared with `interactive`. For example, under `ondemand`, while the performance of AnTuTu is 24% worse, that of Vellamo is about 20% better. Neither of the two governors shows a clear advantage over the other.

Finally, for the Android apps, `powersave` also has the lowest performance, about 2.51$\times$ lower than `interactive`. However, its power reduction is significant, resulting

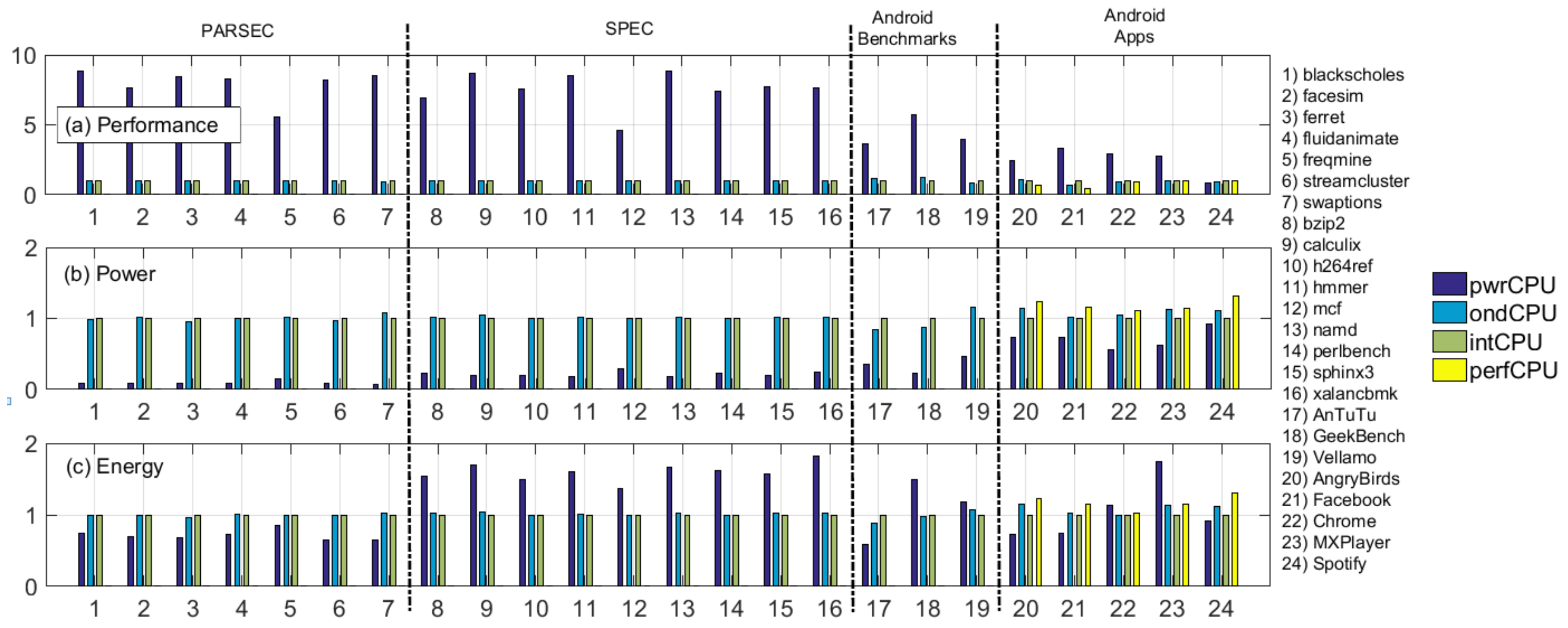


Figure 3.3: CPU Governors: (a) Performance (b) Power and (c) Energy on *N6*. Normalized w.r.t `interactive` governor.

in net energy savings in 3 out of the 5 cases. The `performance` governor, as mentioned in Section 3.2, fixes the CPU frequency to its maximum value. Consequently, in all 5 cases, it causes the most power consumption. However, with MX Player, it does not produce better performance than `interactive`. In the case of Spotify, the performance is about the same, but the power consumption is 31% more. These are typical cases where the higher CPU frequencies are *not* required, and, if used, simply waste energy.

With the tested Android apps, the `ondemand` governor generally has similar performance compared with `interactive`. However, its power consumption is more in all 5 cases. In terms of energy, the default `interactive` appears to be a better choice, with `ondemand` consuming on average 9% more energy.

Two conclusions can be drawn from this part of the tests. First, given that the CPU utilization by Android apps is relatively low, and that sustained use of the highest CPU frequency can cause thermal throttling, the usage of *the `performance` governor should be carefully evaluated to decide if it should be supported.* Second, the energy results indicate that the low end of the CPU frequencies can produce net energy savings compared with `interactive`. This seems to suggest that `interactive` and `ondemand` may be too aggressive in ramping up the CPU frequency, leaving room for improvements. In addition, since different apps show different requirements, the following observation can be made for balancing energy and performance:

OBSERVATION 2: A DVFS mechanism which can adapt itself to the behavior of individual apps can provide higher energy efficiency without sacrificing performance.

3.3.3 Memory Bandwidth Governors

The performance of processors does not entirely depend on the CPU clock frequency. The memory subsystem also plays a key part. Data transfer between the last level cache and the DRAM takes place over a dedicated bus. To improve power efficiency, hardware manufacturers have introduced flexible memory bandwidth and the *devfreq* subsystem in the Linux kernel enables the control of this bandwidth. Nevertheless, reference [123] reports that the memory bus takes up a significant share (about 23%) of the total system power in servers. It is therefore important to understand the behavior of the memory bandwidth governors on current mobile devices.

The *N6*, which supports 13 memory bandwidths (BW) between 762 and 16250MBps, comes with 4 BW governors (Section 3.2). The `powersave` and `performance` governors are compared against the default `cpubw_hwmon`, while keeping the CPU governor fixed (`interactive`). The results are shown in Figures 3.4 (a), (b) and (c). Overall, the default governor performs rather well. However, the difference from the other two governors is in general relatively small, although there are noticeable exception cases.

Excluding 3 exception cases (`mcf`, GeekBench, Spotify), `powersave` has, on average, almost identical performance, power, and energy as the default governor. This suggests that the default governor mostly uses the lower end of the bandwidths which is verified by observing the histogram of the memory BWs chosen. This behavior is in-fact consistent with the findings in [37] which suggest using the lowest BW is often sufficient. For Spotify in 3.4 (b), power consumption for the `powersave` BW governor is higher than the default. On closer examination of the power trace, it is observed that for each of the governors, peak power *during* the data download is the same. But lowering the memory BW increases the download *time* thereby increasing the net power (and also energy) consumed.

The `performance` governor does have the best performance in almost all cases. However, its performance advantage over the default governor is very limited – less than 4% on average. Only `mcf` and Facebook show notably better performance – by 17 and 13% respectively. In terms of energy, the averaged result of PARSEC and SPEC combined is 6% more than the default governor, while the Android benchmarks and apps shows bigger difference – about 13% more compared with the default.

<u>OBSERVATION 3:</u> Compared with the CPU governor, the memory bandwidth governor's impact on performance, power, and energy is limited but not negligible. Compared with

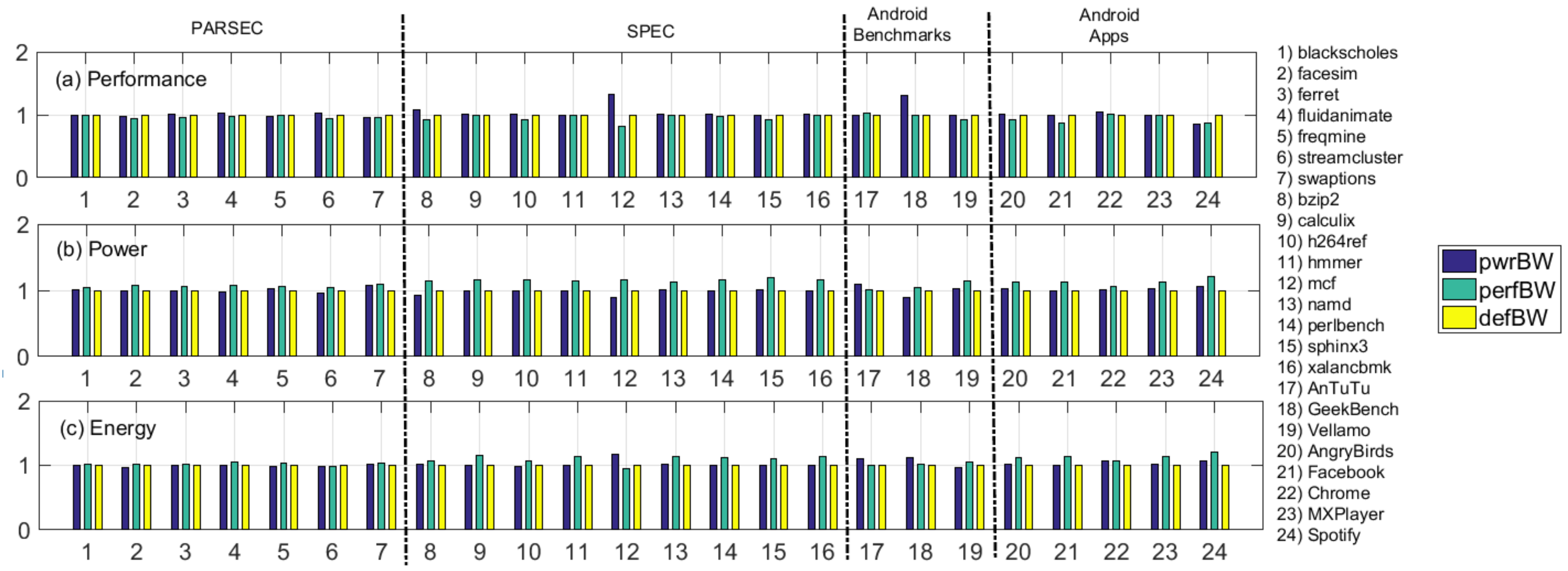


Figure 3.4: Memory BW Governors: (a) Performance (b) Power and (c) Energy on *N6*. Normalized w.r.t `cpubw_hwmon` BW governor.

the static governors, the default `cpubw_hwmon` *is probably the best. This is particularly true energy-wise with the Android benchmarks and apps.*

Although the observations 2 and 3 show that default governors perform quite well, it is shown in the sequel that there is significant room for improvement when governors *coordinate* rather than operate independently.

3.3.4 GPU Governors

Like the CPU, GPU too supports DVFS and the *devfreq* module in the Linux kernel provides a means to change the clock frequencies.

Table 3.3: 3DMark score for different GPU governors on *N6*. CPU governor fixed to `interactive`.

	Default	Powersave	Difference
Overall Score	1339	643	$-2.082\times$
Graphics Score	1338	560	$-2.388\times$
Physics Score	1394	1347	$-1.035\times$
Graphics FPS	8.3	3.5	$-2.371\times$
Physics FPS	29.4	30.2	$1.027\times$

On the *N6*, the GPU has 5 frequencies: 600, 500, 389, 300, 240MHz and the *devfreq* module supports 4 GPU governors. The default governor `msm_adreno_tz` and the `powersave` governors are evaluated. 3DMark benchmark which uses OpenGL 3.1 are run with the CPU governor set to `interactive`. Table 3.3 compares the performance and FPS of the two GPU governors as reported by the app. In the graphics test, the `powersave` governor shows a significant reduction in performance (2.38×) and FPS (2.37×). The physics tests involve CPU activity as well and since the CPU governor is the same, it is seen that the two governors produce close performance and FPS results. On the other hand, the `powersave` GPU governor produces significant reductions in power and energy when compared to the default configuration, $1.71\times$ and $1.69\times$ respectively. It is also noted that the default GPU governor is aggressive and chooses the highest GPU frequency most of the time. Furthermore, reference [27] presents a control strategy which performs DVFS for the CPU *and* GPU simultaneously, to achieve up to 26% reduction in power consumption for comparable performance. During the runtime of an application, the CPU and GPU get stressed dynamically. Hence independent DVFS governors for CPU and GPU tend to be energy inefficient.

OBSERVATION 4: *Coordinated control of the CPU and GPU DVFS states can lead to better power/energy efficiency.*

3.3.5 Cross Effects

With several power/energy management methods implemented at various levels of the stack, it is not unexpected to observe one technique working against another. While reference [48] reports such a phenomenon on server machines, in the experiments performed in this study however, it is observed that the different governors *do not* seem to work *against* each other. Nevertheless, a few scenarios are observed where the default governors do not contribute to their intended purpose and can be designed in a much better way.

CPU Governor vs BW Governor

The interactions of CPU governors and memory bandwidth governors are tested on all the 5 Android apps, as well as 3 PARSEC benchmarks, `blackscholes`, `facesim`, `freqmine`, and 3 SPEC benchmarks, `bzip2`, `mcf`, `perlbench`. The subset of PARSEC and SPEC benchmarks were chosen so as to stress the CPU and memory BW. The performance results are listed in Tables 3.4 and 3.5 and power results are listed in Tables 3.6 and 3.7.

Table 3.4 shows the performance loss data when the CPU governor is changed from `interactive` to `powersave`, while the BW governor is fixed. It is clear that, in this interaction scenario, the CPU governor is the determining factor for performance. The data are average numbers among each groups of benchmarks. It can be seen that changing the CPU governor from `interactive` to `powersave` has a big impact on performance, regardless of the BW governor. The Android apps see the least amount of performance change, but the loss is still significant: 124% to 151% or 2.2 - 2.5× slowdown.

Table 3.4: Performance difference when changing CPU governor from `interactive` to `powersave`, with fixed BW governor. All numbers compared against *intCPU-defBW* on *N6*.

BW Gov	Apps	PARSEC	SPEC
Default	-151.4%	-634.4%	-513.3%
Powersave	-125.6%	-642.9%	-468.4%
Performance	-124.1%	-650.7%	-557.2%
Baseline	InteractiveCPU-DefBW		

On the other hand, when the CPU governor is fixed, changing the BW governor produces much smaller performance changes. Table 3.5 shows the performance variation when the BW governor is changed from the default to `powersave` and `performance`, with the CPU governor fixed. Data for `interactive` and `powersave` CPU governors are relative to *intCPU-defBW* and *pwrCPU-defBW* respectively. The results for individual Android apps are listed, but only the average for the PARSEC and SPEC programs for brevity. The difference in general is less than 15% with a few exceptions, and the maximum is 41.8% difference as seen with AngryBirds. However, with `powersave` CPU governor the game-play is no longer smooth. Similarly for MXPlayer with the `powersave` CPU governor, the 137s video takes over 300s to complete regardless of the memory BW governor.

As for power, the behavior is a little more complex. Although the CPU governor still has greater impact, across the board, the effect of the BW governor increases, particularly when the CPU runs at low frequencies. Table 3.6 shows the change in power consumption when the CPU governor changes from `interactive` to `powersave` with fixed BW governors. The data are average numbers for each benchmark groups. The CPU governor's impact on power is obviously very significant, although much less so with the Android apps. Table 3.7 shows the change in power when the CPU governor is fixed, and the BW governor is changed from the default to `powersave` and `performance`.

Table 3.5: Performance difference when changing BW governor from default to `powersave` or `performance`, with fixed CPU governor. Numbers compared against *intCPU-defBW* and *powsavCPU-defBW*, respectively, on *N6*.

	InteractiveCPU		PowersaveCPU	
	pwrsavBW	perfBW	pwrsavBW	perfBW
AngryBirds	-1.4%	7.8%	27.8%	41.8%
Facebook	0.0%	13.5%	4.8%	8.5%
Chrome	-4.9%	1.0%	-2.6%	8.2%
MXPlayer	0.0%	0.0%	0.0%	1.3%
Spotify	17.5%	14.5%	-19.0%	-19.7%
PARSEC avg.	1.7%	3.0%	0.5%	1.0%
SPEC avg.	-9.3%	12.4%	0.2%	5.2%
Baseline	InteractiveCPU-DefBW		PowersaveCPU-DefBW	

Table 3.6: Power difference when changing CPU governor from `interactive` to `powersave`, with fixed BW governor. All numbers compared against *intCPU-defBW* on *N6*.

BW Gov	Apps	PARSEC	SPEC
Default	-28.6%	-89.1%	-75.5%
Powersave	-38.1%	-89.3%	-74.8%
Performance	-23.3%	-84.6%	-66.1%
Baseline	InteractiveCPU-DefBW		

Table 3.7: Power difference caused by changing BW governor from default to `powersave` or `performance`, with fixed CPU governors. Numbers compared against *intCPU-defBW* and *pwrsavCPU-defBW*, respectively, on *N6*.

	InteractiveCPU		PowersaveCPU	
	pwrsavBW	perfBW	pwrsavBW	perfBW
AngryBirds	2.0%	12.0%	-2.7%	25.0%
Facebook	-0.2%	13.3%	0.0%	27.6%
Chrome	0.3%	5.8%	-2.1%	21.4%
MXPlayer	2.2%	13.1%	-0.5%	15.3%
Spotify	6.1%	13.1%	-0.5%	19.7%
PARSEC avg.	0.5%	5.7%	-1.4%	47.6%
SPEC avg.	-6.3%	15.0%	-4.2%	59.6%
Baseline	InteractiveCPU-DefBW		PowersaveCPU-DefBW	

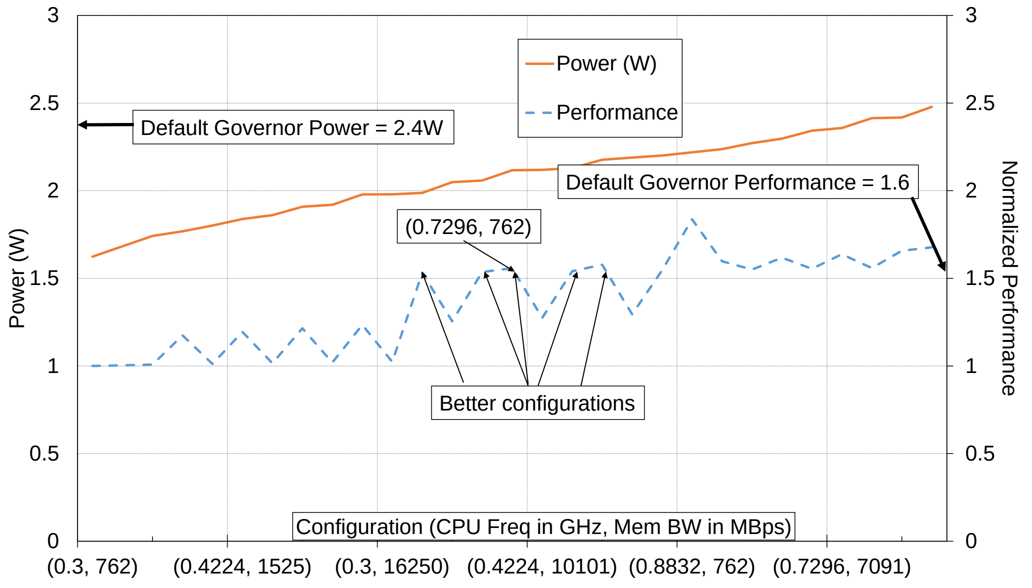
Two observations can be made from Table 3.7. First, the `powersave` BW governor, i.e., setting the bandwidth to its lowest, generally has limited effect on power under either CPU governors. The SPEC program `mcf` is the only exception with a 17% difference under the `interactive` CPU governor. This again indicates that the bandwidth is not saturated, and most of the time lower bandwidths are used, as pointed out in Sec. 3.3.3.

Second, the `performance` BW governor, i.e., setting the bandwidth to its largest, has a big impact on power consumption. This is particularly noticeable when the `powersave` CPU governor is used, which fixes the CPU frequency at its lowest. This behavior is expected – when running at a low frequency, the CPU's contribution to the overall power is lower, and correspondingly, the BW's portion increases. However, it is noted that, as far as the Android apps are concerned, under the `interactive` CPU governor, the BW governor's impact on performance and power does not show a clear trend, and appears to be dependent on the individual apps. To test this hypothesis further, the Android apps are run at different fixed CPU frequencies and memory BWs, and found that performance similar to the default governors can be obtained at much lower energies. For example, running AngryBirds at CPU frequency of 0.729GHz and memory BW of 762MBps consumes 12.11% lesser power for the same performance(See Fig. 3.5a). It is also noted that for apps like MX Player which use dedicated hardware codecs, tuning CPU frequency and memory BW doesn't show significant energy reduction 5% as compared to default governors (See Figure 3.5b). On current Android mobile devices, the *cpufreq* and *devfreq* subsystems work independently of each other. This issue is addressed in the next chapter where a feedback controller minimizes energy consumption of an SoC by performing DVFS of both the CPU *and* the memory bandwidth simultaneously.

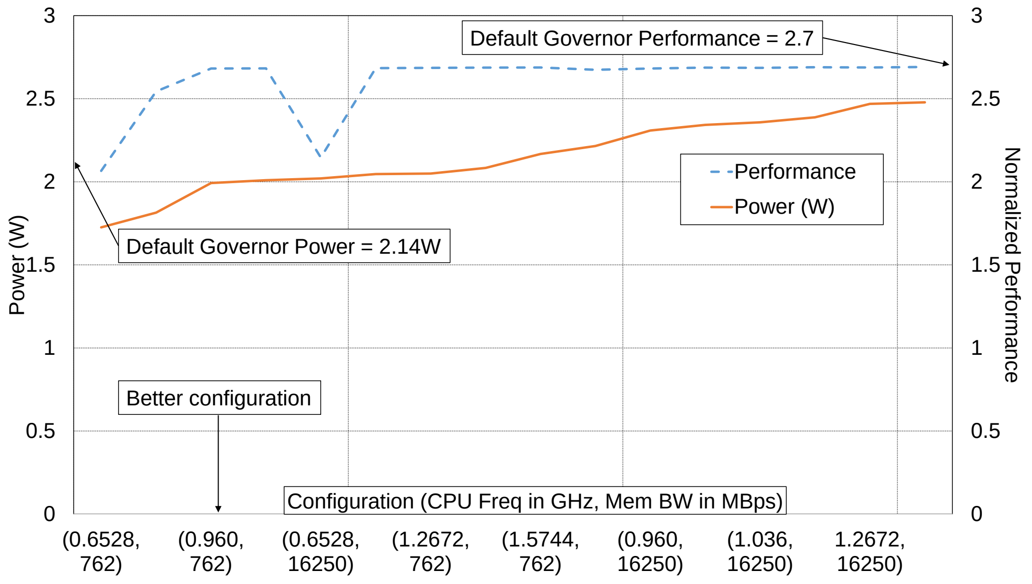
<u>OBSERVATION 5:</u> A more global and coordinated control of multiple components (CPU, GPU and memory BW) should be further investigated for better energy efficiency.

<u>CPU Governor vs CPU Idle</u>

The interactions of CPU governors and *cpuidle* is evaluated by changing the CPU governors between `interactive` and `powersave` for *cpuidle* turned ON and OFF. This is performed on the *N5X* with all 5 Android apps and a subset of benchmarks from PARSEC



(a)



(b)

Figure 3.5: (a)AngryBirds and (b)MXPlayer: Performance and Power trends on *N6* for different CPU frequencies and mem BW combinations. Performance normalized to configuration (0.3GHz, 762MBps).

and SPEC CPU2006. The PARSEC benchmarks are run with 4 threads. Table 3.8 lists the performance, power and energy difference as compared to the default setting on the *N5X*.

Table 3.8: Performance, Power and Energy difference of different CPU and idle configurations as compared to default *intCPU-idleON* on *N5X*.

	InteractiveCPU	PowerSaveCPU	
Perf.	intCPU-idleOFF	pwrCPU-idleON	pwrCPU-idleOFF
PARSEC	-1.00%	-357.2%	-356.7%
SPEC	-0.06%	-450.3%	-447.9%
Apps	-6.37%	-116.00%	-132.9%
Power	intCPU-idleOFF	pwrCPU-idleON	pwrCPU-idleOFF
PARSEC	1.2%	-88.7%	-85.4%
SPEC	21.7%	-85.1%	-76.8%
Apps	3.8%	-15.9%	-15.6%
Energy	intCPU-idleOFF	pwrCPU-idleON	pwrCPU-idleOFF
PARSEC	1.9%	-48.4%	-34.1%
SPEC	21.8%	-21.2%	23.2%
Apps	3.5%	-5.6%	-4.4%

The results when CPU governor is fixed to `interactive` is discussed first. The data show that *cpuidle* has virtually no impact on program performance for PARSEC and SPEC. The 6% difference seen in the apps is dominated by AngryBirds (discussed below). In terms of power and energy, it is seen that *cpuidle* has very limited effect with PARSEC and the apps. This is because (1) All 6 cores of the *N5X* are used when running the PARSEC programs, regardless of *cpuidle* and (2) For Android apps, the screen dominates the power consumption and hence masks any gains from the idling of unused cores. When *cpuidle* is OFF, only state C0 (highest power state) is used on the 6 cores. However, turning ON *cpuidle* shows that the 6 cores continue to be in the C0 state most of the time as well. The only application that appears to not conform to this trend is AngryBirds. Analyzing the raw data, it is observed that the median SPF is 7.7ms for *interactiveCPU-idleON* and 9.3ms with *interactiveCPU-idleOFF*, both well below the 16ms-per-frame threshold which is recommended for a smooth user experience. Since the end user cannot perceive this minor change, the following observation is arrived at:

<u>*OBSERVATION 6:*</u> *cpuidle does not improve performance for real world apps at least on Nexus 5X.*

SPEC benchmarks, on the other hand, consume an average of 21.7% more power when *cpuidle* is turned OFF. This is because these are single threaded programs and only one core is used. With the screen and WiFi being turned OFF, extraneous power consumption is eliminated. If *cpuidle* is OFF, the otherwise idle cores are prevented from going to sleep, thus causing a significant power increase.

The data trends are similar when the CPU governor is changed to `powersave`. However, the degree of impact of *cpuidle* shows some difference. In terms of performance, *cpuidle* still has almost no effect on PARSEC and SPEC. Apps, however, have greater performance difference. Power- and energy-wise, for PARSEC and SPEC, it is seen that, under the `powersave` governor, turning *cpuidle* OFF makes much greater difference than under `interactive`, suggesting an influence of CPU frequency on *cpuidle*. The trend for apps, however, is quite different. Under `powersave`, the average power and energy differences are 0.4% and 1.3%, respectively, when *cpuidle* is turned OFF. With the CPU frequency at its lowest, the power drawn by the screen overshadows CPU power even more, thus concealing any positive effects due to idling.

<u>*OBSERVATION 7:*</u> *The data trend indicates that cpuidle is effective only when the CPU is the dominant power consumer in the mobile device. Coordination between CPU idling and CPU governors can conserve more power and hence more energy.*

3.4 Summary

In this work a characterization of performance, power and energy of various energy management schemes on Android smartphones is performed. A total of 367 tests is conducted which include 5 popular Android apps, 4 prominent Android benchmarking apps, and benchmarks from PARSEC and SPEC CPU2006. While the analysis is certainly not exhaustive, the findings are presented with the intention of helping software and hardware developers better understand the interactions between different layers of the stack so as to design effective algorithms for mobile platforms. To conclude this chapter, Table 3.9 summarizes all the tested options and the main findings.

Table 3.9: Summary of tests and findings.

• Compiler optimizations	Compiler optimizations of `gcc` have little impact on power or energy.
• Multithreading	1) Unlike on servers, multithreading does not save energy. 2) Current mobile devices are not designed to run applications with high TLP.
• CPU governors	1) CPU governor can achieve higher energy efficiency by adapting to behavior of individual apps. 2) Vendors should carefully evaluate the `performance` governor.
• BW governors	1) Compared with CPU governors, BW governor's impact on performance, power, and energy is limited but not negligible. 2) Lower bandwidths are often sufficient for performance.
• GPU governors	Default GPU governor is aggressive and must coordinate with CPU governor.
• CPU-BW cross effect	Coordinated control of multiple components (CPU, BW, GPU) may lead to better performance-power balance.
• CPU DVFS-Idle cross effect	1) CPUIdle does not seem to be very effective on N5X for Android apps. 2) Coordination of DVFS and CPUIdle may be a better solution.

CHAPTER 4
COORDINATED CONTROL: MOBILE DEVICES

Energy management is a key issue for mobile devices. As observed in CHAPTER 3, power management for various hardware components relies heavily on OS modules known as governors. The governors implement algorithms that attempt to balance performance and power consumption. This chapter establishes that the existing governors are: (1) general-purpose by nature, (2) focused on power reduction, and (3) not energy-optimal for many applications. The need for an application-specific approach is established that could overcome these drawbacks and provide higher energy efficiency for a class of applications. It is also shown that existing methods manage power and performance in an independent and isolated fashion, and that, *coordinated control* of multiple components is more energy efficient. In addition, it is also noted that on mobile devices, energy savings at the expense of performance is not desirable. Consequently, a solution is proposed that minimizes energy consumption of specific applications while maintaining a user-specified performance target. The solution consists of two stages: (1) offline profiling and (2) online controlling. Utilizing the offline profiling data of the target application, the control theory based online controller dynamically selects the optimal system configuration (in this work, a combination of CPU frequency and memory bandwidth) for the application, while it is running. The energy management solution is tested on a Nexus 6 smartphone with 6 real-world applications. Energy savings in the range of $4-31\%$ is achieved as compared to default governors with a worst case performance loss of $< 1\%$.

4.1 Overview

System-on-Chips (SoC) for mobile devices have seen continued improvements in performance due to diversified functionalities provided by GPUs, DSPs etc. The processor performance has experienced a boost over the last few generations due to commensurate advancements in memory technologies. On the software end, the emergence of a variety of applications utilizing the hardware diversity has increased the popularity of mobile devices. Battery technology however, has not kept pace, thereby making battery life one of the top concerns of end users.

In the interest of prolonging battery life, modules in the latest SoCs are equipped with power/energy management solutions. Greedily entering low power states and Dynamic Voltage and Frequency Scaling (DVFS) are the most commonly used techniques. For example, the Linux kernel on Android devices has subsystems called *cpufreq* and *devfreq* to manage power consumption of CPU and other DVFS-capable components, respectively. To further improve performance, the Linux kernel for Android supports a "touch boost" feature to ramp up the CPU frequency when required. Within these subsystems, modules known as *governors* implement algorithms that determine clock frequencies to be used under different conditions. The governors attempt to strike a balance between performance and power dissipation. For instance, the *interactive* governor, the current default CPU governor on Android devices, will quickly ramp up the frequency when user interactions are detected and will reduce the frequency when there are no interactions.

The stock governors are designed for general purpose usage. Consequently the key observation is that, in the process of improving performance, stock governors result in higher energies for some applications, including popular ones like AngryBirds. This suggests the relevance of *application-specific controllers* in such scenarios. Additionally, current state-of-the-art governors on Android mobile devices are tailored for power optimization. However, as observed in [26], governors for mobile devices must be designed for minimizing energy with performance constraints and not power because energy consumption is strongly correlated with battery life. Correspondingly, the problem statement addressed in this chapter is the following:

Problem: *Choose the minimum energy system configuration while maintaining the performance target.*

Maintaining performance while minimizing energy under dynamic run-time conditions is a complex problem. However, as is elaborated in the later sections, the problem statement can be divided into two parts: (P1) Maintain the performance target and (P2) Minimize the energy consumption. The solution adopted is implemented in two stages. The application is profiled offline (Stage 1) and the feedback controller is implemented on-line (Stage 2) utilizing the profiled data to minimize energy while maintaining performance. In Stage 2,

1) To maintain performance (P1), a performance regulator is used. The regulator computes a signal based on the measured and target performance.
2) To minimize energy (P2), an optimizer uses the output from the regulator and chooses a hardware system configuration from the offline profiled data in order to minimize the energy.

An important observation to be made at this juncture is that *the software governors work independently of each other*. A few studies ([23, 24, 25]) have shown that independent power/performance control strategies can lead to conflicting policies causing performance and/or power losses. Addressing this problem, researchers have investigated the coordinated control of different hardware subsystems on servers [23], SoCs [26] and embedded systems ([29] and references therein).

Proposals which offer energy minimization with performance maintenance by the control of multiple subsystems simultaneously, are either implemented on (1) simulators designed for servers [23] or (2) real physical devices with CPU-only DVFS [38]. Reference [26] discusses CPU and memory DVFS trade-offs for mobile devices using a `Gem5` simulator and SPEC CPU2006 benchmarks as their test cases.

The highlights of the work reported in this chapter are the following:

1. For some popular applications, the stock power manager causes excessive energy consumption on a modern Android mobile device.

2. A software controller is implemented to minimize the energy consumption of such applications while maintaining a user-specified performance target.

3. The controller achieves $4 - 31\%$ energy savings on 6 real-world applications with a worst case performance loss of less than 1%.

4. Unlike default governors that are independent for each subsystem, the control strategy adopted in this chapter is the *coordinated control* of CPU frequencies and memory bandwidth. Compared to a CPU-only energy minimization scheme, energy savings improves by 53%.

5. The control strategy can be readily extended to include GPU frequencies, GPU memory bandwidth, network packet rate, etc. Furthermore, it can be implemented on any mobile device capable of DVFS.

4.2 Motivation

This section is an extension to the observations made in Section 3.3.5. The crucial point is the coordination among different components for the purposes of power management. The drawbacks of the existing DVFS governors are evident from previous studies as well as experiments performed in CHAPTER 3. This provides the motivation for the work in the present chapter.

A potential source of ineffectiveness for the existing governors is the lack of coordination among different components. Take the *ondemand* CPU governor mentioned in 3.2.2 as an example. Its actions are solely based on the CPU load and is oblivious to the state of other components such as the memory. In [23] the authors point out that independent control policies could lead to conflicts and, subsequently, to oscillations, thereby greatly reducing the effectiveness in achieving energy savings. They further demonstrate that co-ordinated control of CPU and memory DVFS is a better strategy.

The current *cpufreq* and *devfreq* governors work well in some cases. However, they do have their limitations.

In [124] the authors compare power consumption of a mobile platform at a set of fixed CPU frequencies and when using two different governors. Four typical usage scenarios are tested: 3G, WiFi, voice call, and ebook reading. They find that the optimal CPU frequency in terms of power consumption is dependent on the use case. In addition, in two of the four cases, the *ondemand* governor consumes more power than *most* of the fixed frequencies. This suggests that for some applications at least, an application-specific DVFS strategy may be a better solution.

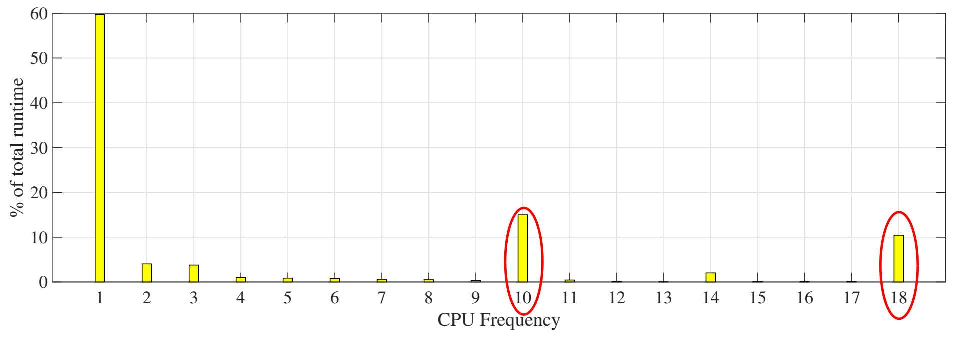


Figure 4.1: Histogram of CPU frequencies for eBook application. The numerals indicated on the x-axis stand for the choice of 18 discrete frequencies (in the range $0.3 - 2.65$GHz). See Table 4.2 for details.

This study too evaluates the behavior of the default governors for some applications. Fig. 4.1 shows the histogram of CPU frequencies chosen by the default CPU governor on a Nexus 6 smartphone for an e-book reader application when there is no user interaction such as scrolling or zooming, i.e., when the user is just reading the page. The screen brightness is fixed at the lowest level, WiFi is turned ON and there are no applications running in the background. The x-axis of the figure shows the CPU frequencies from low to high and the y-axis indicates the percentage of time spent in a given frequency during the test period. It can be seen that, even though there are no user interactions, the CPUs, under the control of the governor, spend over 10% of the time in the highest frequency, and about 15% of time in a middle frequency (No. 10), as highlighted in the figure. Running at a higher-than-necessary clock frequency results in energy wastage. It is clearly seen that the default DVFS governors on the current Android devices, in the process of providing better performance, are not energy-optimal for many applications. Thus it becomes necessary to investigate whether an application-specific approach that can set system configurations, e.g., DVFS, based on the characteristics of specific applications, can lead to higher energy efficiency. In the course of searching for a better solution however, one must keep in mind that performance is a top priority for end users. As a general principle, *energy savings should not be achieved at the expense of performance degradation*.

This work exploits the ineffectiveness of default governors and presents a strategy motivated by (1) energy minimization in contrast to power minimization, (2) meeting performance requirements and (3) coordinated control of multiple components.

4.3 Controller Design

This section presents the design of the application-specific performance-aware energy optimization solution. As mentioned in Section 4.1, the solution consists of two stages: offline profiling and online controlling. Both stages are discussed in detail below.

4.3.1 Offline Profiling

The application-specific aspect of the solution relies primarily on the run-time utilization of offline profiled data of the target application. Prior to online controlling, in the offline profiling stage, the performance and power of a target application is measured under different *system configurations*. For each such configuration, the power and performance data are averaged over three runs for every application tested.

The term *system configuration* means hardware or software settings and combinations thereof, that could impact the performance of applications. Examples include CPU frequency, memory bandwidth, storage parameters, network packet transfer rate, thread scheduling policy and so on. In the context of the present chapter, system configuration is used to mean the combination of CPU frequency and memory bandwidth. It must be emphasized that the solution is not limited to controlling this particular configuration and can be extended to include other configurations mentioned above.

Profiling data of an application are organized in a table, an example of which is shown in Table 4.1. The performance data are normalized with respect to the value corresponding to the lowest system configuration and is termed *speedup*. The lowest system configuration in this work refers to the lowest CPU frequency and lowest memory bandwidth of the SoC. Power data, obtained with a Monsoon power monitor [110], are the average power consumption of the entire device during the test period.
There are two issues associated with offline profiling:

1. The number of configurations that require profiling could be rather large in practice.

2. There could be discrepancy between the controller's runtime environment and the profiling environment.

Since the tuple (CPU frequency, memory bandwidth) is considered as the system configuration, exhaustive offline profiling involves running the application for every combination of supported CPU frequency and memory bandwidth. On a Nexus 6 smartphone for example, there are $18 \times 13 = 234$ combinations (18 CPU frequencies and 13 memory bandwidths). While a large number of system configurations profiled gives fine-grained data, it also increases the profiling time-span as well as the online controller's run-time overhead due to the larger search space. Addressing the issue of space explosion, a maximum of $9 \times 2 = 18$ configurations are chosen, i.e., for each alternate CPU frequency with the lowest and the highest memory bandwidths. For each profiled CPU frequency, linear interpolation is used to get the intermediate data for the rest of the memory bandwidths. Interpolations are not performed along the CPU frequency dimension because it is observed that in general, performance and power do not change by a large margin for neighboring CPU frequencies. Although this approach introduces quantization and modeling errors, practical results show that the controller is robust enough to handle it.

Table 4.1: Sample table with performance and power data profiled offline for AngryBirds application

	Config (GHz,MBps)	Speedup, $\mathbb{S}$	Power, $\mathbb{P}$ (mW)
1	$(0.3, 762)$	1.0	1623.57
2	$(0.3, 1525)$	1.0038	1682.83
3	$(0.3, 3051)$	1.0077	1742.09
$\vdots$	$\vdots$	$\vdots$	$\vdots$
31	$(0.8832, 762)$	1.837	2219.22

On mobile devices at any given point in time, many applications run in the background albeit most of them are in the "sleep-state". Applications such as e-mail clients perform synchronizations periodically while some applications like Spotify or similar music players continue to run even when they are minimized. Offline data collected for an application under a given background load can be rendered unusable at run-time when the load conditions differ by a large margin. A straightforward method is to profile the application under different background loads. However, the drawback of such an approach is that the profiling overhead increases significantly. The approach used to tackling this issue is to profile the application with a background load, i.e., WiFi ON, e-mail synchronization enabled and Spotify running in the background. The controller performance is then tested for heavier and lower background load conditions. The results show that while there is input data dependence, this issue is mitigated in part by the feedback controller. Overall, the approach performs rather well for a range of typical load conditions.

Furthermore, the performance ($\mathbb{R}_{def}$), running time ($\mathbb{T}_{def}$) and average power ($\mathbb{P}_{def}$) of the application is measured under the default governors. The default energy consumption ($\mathbb{E}_{def}$) of the device while the application is running is simply $\mathbb{P}_{def} \times \mathbb{T}_{def}$. The default performance, $\mathbb{R}_{def}$, serves as the basis for the *target performance*, which is an input to the online controller. The energy consumption of the device under the proposed control scheme is compared with the default energy $\mathbb{E}_{def}$.

4.3.2 Online Controller

The offline profiled data are used by the online controller to run the application in an energy-efficient fashion while at the same time meeting a user-specified performance target. The online controller is based on the work presented in [38] and is a feedback control loop as shown in Fig. 4.2. In accordance with splitting the problem statement into two parts P1 and P2 as described in Section 4.1, the online controller K has two parts: a performance regulator and an energy optimizer.

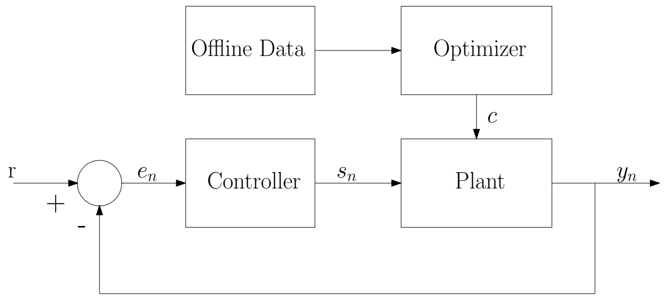


Figure 4.2: Block diagram of feedback controller

Let $\mathbb{C} = \{c_1, c_2, \ldots, c_N\}$ be the set of N system configurations. Each $c_i \in \mathbb{C}$ is an ordered pair $\{f_{(CPU,i)}, f_{(BW,i)}\}$ where $i = \{1, 2, \ldots, N\}$. The term *coordinated control* in this context refers to the ordered pair c_i. Instead of choosing $f_{(CPU,i)}$ and $f_{(BW,i)}$ independently of each other, augmenting $f_{(BW,i)}$ with $f_{(CPU,i)}$ is the meaning of *coordination*. Choosing the best values for $f_{(BW,i)}$ and $f_{(CPU,i)}$ depends on the performance and energy implications of the whole system and not just the individual components. With reference to Fig. 4.2, the feedback controller is implemented as follows:

1. Given a target performance of the application $r \in \mathbb{R}$ and the measured performance of the system $y_n \in \mathbb{R}$, the error is computed as $e_n = r - y_n$. The subscript 'n' represents the control cycle index.

2. Based on e_n, the controller computes $s_n \in \mathbb{R}$ to meet the performance target r. The control cycle duration is $\mathbb{T}$ seconds.

3. The optimizer then applies the system configurations c_i to the phone for an appropriate duration of time represented by the vector $u_n \in \mathbb{R}^N$ so as to minimize energy consumed while meeting the performance target.

4. The performance y_n is measured at the end of the control cycle.

Performance Metric

Performance of an application can be quantified in many ways. Execution time is perhaps the most commonly used metric. Frames per second can be used for video playing applications. Other metrics include number of jobs completed per unit time, task latency, and so on. Android applications are distributed in a compressed format (`apk`) which contains partially to completely obfuscated code. Requiring the developer to implement modifications in the source code so as to report application performance periodically is in-feasible. In this work, the objective is to obtain information about the progress of the application without any source code modifications. Fortunately, modern micro-processors, including SoCs used in Android mobile devices, generally possess a performance monitoring unit (PMU). In this work, Giga-Instructions-Per-Second (GIPS) obtained from the PMU is used as the performance metric. GIPS is considered as a good metric because it is highly correlated with the execution time. It is also to be noted that GIPS has been used as a performance metric in earlier works as well (see [125]).

Controller

The controller, also referred to as the performance regulator, computes a required speedup s_n so as to track the performance target r. Specifically, the goal of a performance regulator is to reduce the error between the target performance and the measured performance to zero, i.e., $e_n = (r - y_n) \to 0$. The performance of the system is modeled as

$$y_n = s_{n-1} \cdot b_{n-1} \tag{4.1}$$

where b_n is the base speed of the application and s_n is the *speedup* with respect to b_n. Base speed b_n is defined as the speed of the application when the least amount of system resources are consumed. Feedback controllers can be designed with fixed gains or adaptive gains. This work chooses an adaptive gain integral controller in order to accommodate run-time variations, inaccuracies in measurement, modeling errors etc. References [33, 39, 126] have shown the practical feasibility of using adaptive gain integral controllers in a variety of computing environments. Equation 4.3 describes the performance regulator mathematically.

Optimizer

The optimizer computes a system configuration $c_i \in \mathbb{C}$ and applies it to the plant i.e. the phone. Specifically, it computes a vector $u_n^T = [\tau_{c_1}, \tau_{c_2}, \ldots, \tau_{c_N}]$ where τ_{c_i} represents the time duration for which configuration c_i is applied. Let $\mathbb{S}, \mathbb{P} \in \mathbb{R}^N$ denote the average speedup and power vectors generated via offline profiling (See Table 4.1). The i^{th} element in $\mathbb{S}$ and $\mathbb{P}$ denote the average speedup and power of configuration c_i.

Encoding the actions of the performance regulator and the optimizer mathematically,

u_n is generated by the following equations invoked at the end of every control cycle:

$$e_n = r - y_n \tag{4.2}$$

$$s_n = s_{n-1} + \frac{e_{n-1}}{b_{n-1}} \tag{4.3}$$

$$\min_{u_n} \quad {u_n}^T \cdot \mathbb{P} \tag{4.4}$$

$$s.t. \quad \mathbb{S}^T \cdot u_n = s_n \cdot T \tag{4.5}$$

$$\mathbb{1}^T \cdot u_n = T \tag{4.6}$$

$$0 \preceq u_n \preceq T \tag{4.7}$$

Eqn. (4.3), as mentioned earlier, represents the performance regulator. The adaptive gain is encoded by the term $1/b_{n-1}$. The *required speedup* s_n is computed based on the history of e_n which is why Eqn. (4.3) is called an "integrator". One may refer to [32] for details on derivation and stability proofs. Different applications can have different base speeds. For example, on the Nexus 6 smartphone whose lowest possible configuration is $(300MHz, 762MBps)$, the base speed of AngryBirds is 0.129GIPS whereas for a Video Converter application the base speed is 0.471GIPS. To ensure that the controller can track these changes automatically, based on the work in [38], a Kalman filter [127] is used to continuously estimate the application base speed b_n.

Equations (4.4) - (4.7) represent the energy optimizer which is a linear program. Here Eqn. (4.4) encodes the energy minimization objective, Eqn. (4.5) is the performance constraint and Eqns. (4.6) and (4.7) are timing constraints. The solution to the linear program generates the vector u_n. It can be shown that an optimal solution exists with *at most* two non-zero values for τ_{c_i}. The energy optimizer therefore selects at most 2 configurations c_l and c_h such that $\mathbb{S}(l) \leq s_n < \mathbb{S}(h)$ and $\tau_{c_l} + \tau_{c_h} = \mathbb{T}$. The subscripts l and h represent "lower than" and "higher than" the required speedup respectively. This is pictorially described in Fig. 4.3. Since there are at most N configurations, the run-time complexity of the energy minimization is $O(N^2)$.

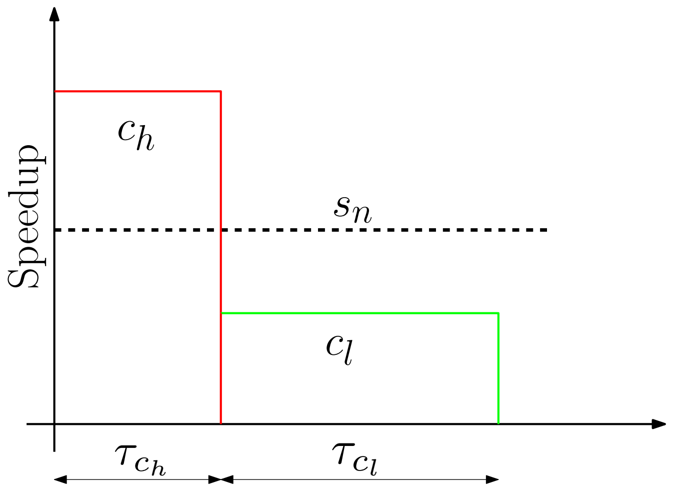


Figure 4.3: Pictorial representation of the energy optimization

$c_l, c_h \in \mathbb{C}$ are applied on the phone for a duration τ_{c_l} and τ_{c_h}. Note that since $\tau_{c_i} = 0$ for $i \neq l, h$, those system configurations are not applied on the phone.

Table 4.2: List of CPU frequencies and memory bandwidths on Nexus 6

CPU Frequency (GHz)				Mem Bandwidth (MBps)			
1	0.3000	10	1.4976	1	762	10	8056
2	0.4224	11	1.5744	2	1144	11	10101
3	0.6528	12	1.7280	3	1525	12	12145
4	0.7296	13	1.9584	4	2288	13	16250
5	0.8832	14	2.2656	5	3051		
6	0.9600	15	2.4576	6	3952		
7	1.0368	16	2.4960	7	4684		
8	1.1904	17	2.5728	8	5996		
9	1.2672	18	2.6496	9	7019		

4.3.3 Implementation Challenges

In contrast to previous works, this work evaluates the controller on a physical device with real applications and run-time conditions. A list of challenges faced during the course of this work and the subsequent solutions adopted is described next.

Previous work ([38]) required source code modifications to enable the controller to monitor the application performance. Specifically, the “application being controlled” reports its performance periodically to the controller. This work uses GIPS derived from a PMU counter as mentioned above which does not require application developers to modify their code.

However, a commercial phone does not come pre-loaded with the `perf` tool, neither does it provide root access to the Linux kernel. Therefore, the *userdebug* version of Android Marshmallow 6.0 is built along with `perf`. This enables (1) measuring performance at run-time and (2) changing CPU frequency and memory bandwidth. The `perf` tool on the *N6* has the lowest sampling period of 100ms. Furthermore, the computation overhead at this sampling period is 40%. Therefore, a control cycle duration of 2 seconds is chosen for all the experiments, i.e., with reference to Eqn. (4.4), $\mathbb{T} = 2$. The overhead of `perf` and the controller is discussed in Section 4.4.1.

Unlike commercial Intel and AMD processors, the Snapdragon 805 SoC does not support hardware power and energy counters. Moreover, the setup used for this work allows recording the power consumption of the entire device only. Although the control algorithm ideally requires only the power consumed by the CPU and the memory, the robustness of the controller is shown to be able to handle these modeling inaccuracies.

A typical desktop/server class processor supports multiple processes running in parallel. While ARM based SoCs do support the same, the process consuming most of the resources in an Android device corresponds to the application being currently displayed on the screen. Background application threads are in the "sleep state" in general, woken up periodically depending on the nature of the application. Following suit, the strategy is to control the application *only* while it is running in the foreground.

4.3.4 Applications

A set of 6 real world applications is chosen where each application demonstrates unique characteristics. The applications are individually described below.

VidCon is a video converter application which uses the `FFmpeg` library [128] to convert videos to different formats. For the experiments in this work, a fixed size `mp4` HD video is chosen and the default conversion settings are used.

MobileBench [117] is an established browser benchmark based on BBench [129]. The benchmark loads a collection of websites whose content is available in the phone memory. It offers automatic horizontal and vertical zooming and scrolling as well. The Chrome browser application on the phone is used for running the tests.

AngryBirds is a gaming application with over 100 million downloads on Android alone. This is chosen as a representative gaming application to test the controller performance. The game is manually played for 200 seconds during the experiments.

WeChat is an Internet based text messaging, voice communication and video conferencing application. With over 700 million active users, it is among the most downloaded applications in the communication application segment. The video conferencing feature of this application is chosen and a 100-second long video call is initiated for the experiments.

MX Player is a video player application which can play videos encoded in a variety of formats and has over 100 million downloads. It also supports hardware accelerated decoding and high speed rendering for ARM NEON compliant processors. The controller performance is tested when playing a 137-second long HD video.

Spotify is an audio, podcast and video streaming application with over 100 million subscribers. Using a premium version of the application which avoids advertisements between songs, this application is tested for 100 seconds with songs being changed every 20 seconds.

4.4 Evaluation

This section presents test results of the energy management scheme described in Section 4.3 against the default settings on the *N6*. A detailed analysis of the results is provided with a discussion on few important issues, including (1) application scope, (2) the effect of varying background application loads on the controller performance, and (3) comparison with a CPU-only DVFS strategy.

4.4.1 Results and Analysis

Table 4.3: Summary of performance difference and energy savings obtained by the controller

Application Name	Performance	Energy
VidCon	-0.4%	25.3%
MobileBench	4.1%	15.3%
AngryBirds	0.6%	14.9%
WeChat Video Call	-0.4%	27.2%
MX Player	0.0%	4.2%
Spotify	9.3%	31.6%

Table 4.3 summarizes the performance and energy savings achieved by the controller as compared with the default governors. Each number is the average of three runs. The background load used for the results in Table 4.3 is the same as discussed in Section 4.3.1. In what follows, this background load is referred to as *baseline load*. VidCon, MobileBench browser benchmark and MX player are deadline critical. Even though the controller measures performance for these applications in GIPS, performance numbers in Table 4.3 are based on execution time. For the rest of the applications, performance in Table 4.3 is measured in GIPS.

It can be seen that for all the applications tested, the controller is able to save energy while meeting the performance target. A worst case performance degradation of 0.4% is observed with this control technique. At the same time, compared to the default, $14.9-31.6\%$ of energy is saved with 5 out of the 6 applications and 4.2% with MX Player. The results in Table 4.3 clearly demonstrate the effectiveness of the application-specific approach in achieving substantial energy savings while maintaining performance.

To a large extent, the effectiveness of this approach is due to the coordinated control strategy. The default CPU governor `interactive`, changes the CPU frequency based on the CPU load. The default memory bandwidth governor `cpubw_hwmon`, on the other

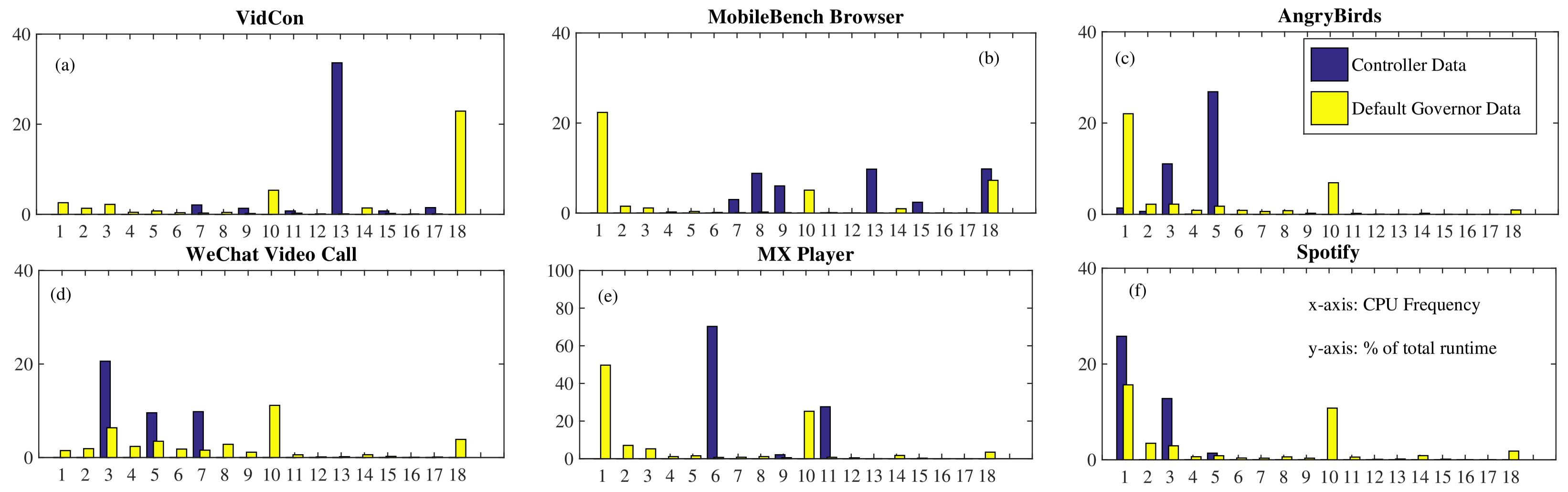


Figure 4.4: Histogram of CPU frequencies: controller vs. default

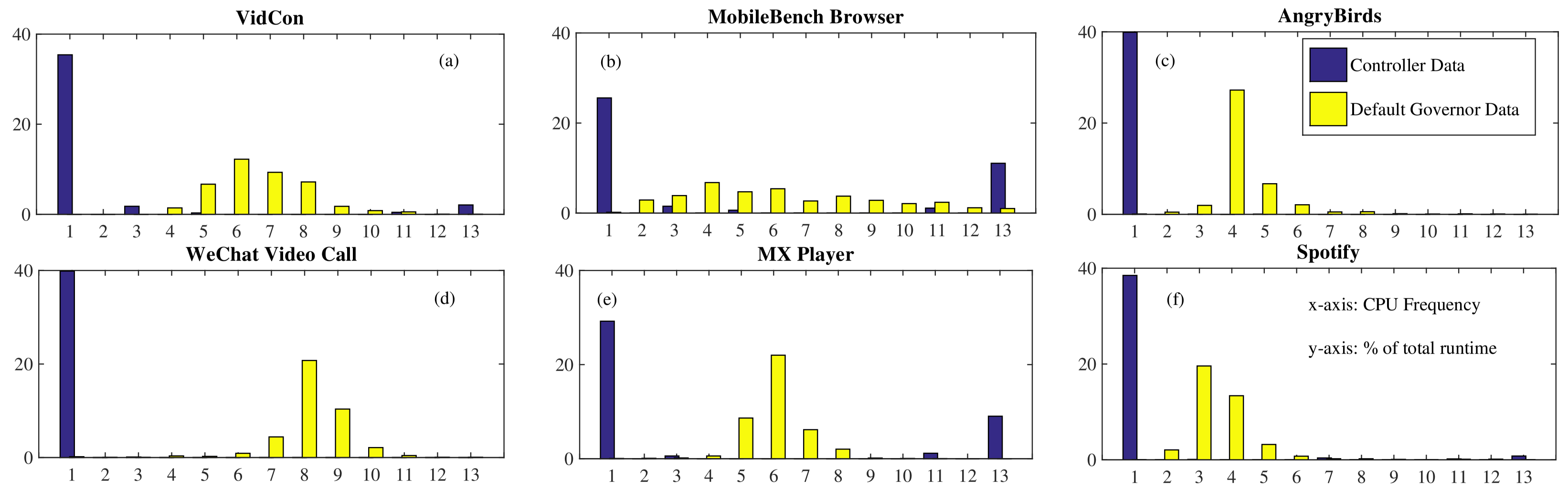


Figure 4.5: Histogram of memory bandwidths: controller vs. default

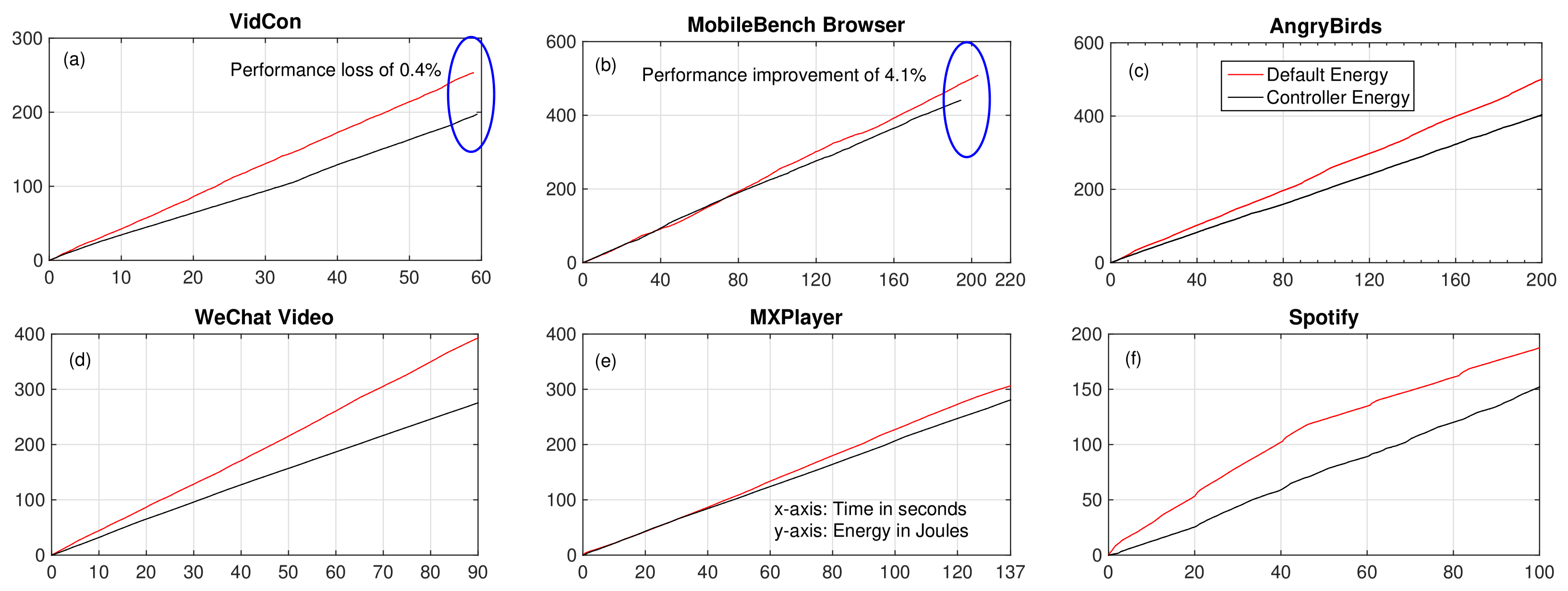


Figure 4.6: Energy Consumption: controller vs. default

hand, monitors the L2 cache read and write events to decide the required bandwidth. Both governors work independently and the results we obtain demonstrate the drawbacks of such an approach.

To help analyze and understand the experimental results, in Figs. 4.4 and 4.5 the percentage of time spent in each of the 18 CPU frequencies and 13 memory bandwidths during the application execution is computed. The choices made by the default governor and the proposed controller is compared as well. Fig. 4.4 shows some of the key characteristics of the default CPU governor. Firstly, in all 6 cases, it spends a considerable amount of time ($12.7 - 27.9\%$) at CPU frequency 10 (1.4976 GHz). Fig. 4.5 illustrates the characteristic behavior of the default bandwidth governor which implements an exponential back-off algorithm while reducing the bandwidth. The offline profiled performance data for AngryBirds, MX Player and Spotify show an improvement of less than 5% for frequencies between 5 and 10 whereas power increases by more than 36%. Secondly, in 3 out of the 6 cases, the highest frequency is used for a significant amount of time ($9.7 - 57.3\%$). With the approach proposed by this work, in 5 out of the 6 cases, the high frequencies are not included in the profiling table supplied to the controller, based on the performance/power characteristics of the profiled data.

It is observed that in Fig. 4.4 (b), (c), and (e), with the default governor, the CPUs are at frequency 1 for the largest amount of time, whereas the proposed controller selects higher frequencies. Intuitively, this should lead to higher energy consumption by this controller. But the results in Table 4.3 show that energy consumption with the controller is lower than default in all 3 cases for the same performance. This phenomenon is a result of the following: (1) The controller is designed to *maintain* a performance target (2) The controller trades higher CPU frequencies against increasing the bandwidth (see Fig. 4.5) and (3) In the solution proposed, the smallest duration for the CPUs to stay at any given frequency is 200ms. Choosing frequency No. 1 even for a duration of 200ms impacts the performance heavily. In fact, for MobileBench and MX Player, the lower frequencies are not even included in profiling data provided to the controller. In Fig. 4.4 (b), (c), and (e), even though it appears that with the default governor the CPUs spend most of the time in the lowest frequency, it should be noted that this is an accumulated time. The CPUs spend short durations (of the order of 10s of milli-seconds) in this frequency before moving on to higher frequencies. The conclusion that can be drawn is that lower CPU frequencies may reduce power consumption but it does not translate to lower energy. Similar or better performance with lower energy can be attained by choosing higher CPU frequencies. To understand this better, Fig.4.6 shows the running energy consumption for each of the benchmarks. The slope of the curve for the controller is smaller than the default governor which clearly shows that choosing CPU frequency and memory bandwidth in a coordinated fashion does lead to better overall energy. Now the 6 applications are discussed individually.

VidCon has a uniform power and performance profile during its execution. Fig. 4.4 (a) shows that the default governor spends nearly 60% of the time in the highest CPU frequency and takes 59 seconds to convert a sample video. The controller, however, chooses a much lower frequency (No. 13) for 80% of the time and is able to convert the same video with 25.3% less energy. The time it takes to convert the video is only 0.4% or about $0.24s$ longer, hardly noticeable by a human user. For this application CPU frequencies 7-18 are used because frequencies below No. 7 resulted in a performance drop of over 50%.

MobileBench browser benchmark, unlike VidCon, has a varying power and execution profile. For a fixed CPU frequency, an average increase of 7% in the relative speedup is noticed between the lowest and highest memory bandwidths. Due to the zooming and scrolling actions, the performance in GIPS too shows a steady increase as CPU frequencies are increased. However, the data used by the online optimizer is restricted between CPU frequency 7 and 18 (See Fig. 4.4). The justification is similar to VidCon in that, when the CPU frequency is fixed at No. 7, the performance is 30% worse than the default. Any lower frequency would incur a larger performance loss resulting in a lower user experience. The controller chooses CPU frequency 18 for a duration longer than the default governor, yet achieves a 15.3% improvement in energy. Although this seems counter-intuitive, the reason for this phenomenon is that the objectives of the default governor and the controller proposed in this chapter are orthogonal. While the default governor tries to *maximize* performance whenever possible, the aim of the proposed controller is to *maintain* a fixed performance. The default governor chooses to assign a higher CPU frequency when it senses a load increase whereas the controller only assigns a higher frequency when it senses a performance drop. As shown in Fig. 4.6 (b), the energy time-line graphs for the default governor and the proposed controller are very similar. But the energy savings are achieved on account of a shorter run-time.

With AngryBirds, the controller is provided with a smaller CPU frequency range because the offline profiling data shows that performance (in GIPS) does not improve beyond CPU frequency No. 5 but power consumption increases steadily for higher frequencies. Compared with the default governor, which spends nearly 20% of the time in frequency No. 10 and some amount of time in the highest frequency, the controller selects frequencies 3 and 5, as shown in Fig. 4.4 (c). The end result is a performance (in GIPS) that is slightly (0.6%) better with an energy saving of 19.3%. AngryBirds involves the GPU for image rendering, but, despite the fact that GPU frequency is not part of the controlled system configuration, no change in the game experience is observed when the controller is deployed. Moreover, the default governor chooses a higher frequency when advertisements get loaded between individual levels, resulting in higher power consumption[1].

When profiling WeChat video call, it is found that for CPU frequencies 1 and 2, the camera fails to record and transmit video reliably and hence they are excluded from the power and speedup table. Additionally, the performance (in GIPS and subsequently video quality) does not show significant improvement beyond CPU frequency 7. However Fig. 4.4 (d) shows that frequencies 10 and 18 get chosen for close to 40% of the time by the default governor. The controller is able to provide comparable performance by choosing

[1]Advertisements consume close to 0.5W of power and an application with several ads will result in rapid battery discharge.

lower CPU frequencies (3, 5, and 7) with No. 3 being used for over 50% of the time. This results in a significant energy saving of 27.2% compared with the default governor.

MX Player is not CPU intensive because it performs video decoding using a hardware decoder and bypasses the GPU to render the image on the screen. MX player has a performance vs. CPU frequency profile similar to WeChat in that, beyond frequency 5, the performance varies very little (0.4%). Furthermore for frequencies between 1 and 4, the video does not play smoothly regardless of the memory bandwidth chosen. Hence CPU frequencies 1 - 4 are not included in the offline profiling table. Due to the nature of the application and the fact that the controller can only manipulate CPU and memory bandwidths, only 5% energy can be saved. The implication is that the default governor indeed does a good job for this application.

Spotify is another case where a limited range of CPU frequencies is included in the profiling table. In fact, only 3 frequencies on the low end are used: frequencies 1, 3, and 5. It is to be noted that even when the CPU frequency is fixed at the lowest, the audio quality does not degrade. However, the default governor, as shown in Fig. 4.4 (f), spends a considerable amount time in the much higher frequencies 10 (27%) and 18 (4.6%). In contrast, the controller spends 64.5% of time in the lowest frequency and 32% in frequency No. 3. Compared with the default governors, the controller saves 31.6% energy with a minor performance loss in GIPS of 0.4%.

Controller Overhead

As mentioned in Section 4.3 the controller consists of three parts: (1) measurement (2) performance regulation and energy optimization and finally (3) actuation. Accordingly presented are the overheads for each part of the controller. The controller measures performance twice in each control cycle. On an average, the measurement is done every 1s when the control cycle duration is 2s. The `perf` tool takes 1.04s on average, i.e., a 4% computation overhead, to report the measurement. The power consumption overhead for `perf` at a sampling period of 1s is 15mW, a relatively negligible number. The execution time of the performance regulator and the energy optimizer together is less than 10ms per control cycle with an average power consumption of 25mW. Changing the CPU frequency and memory bandwidth requires writing into the appropriate `sysfs` files. The CPU frequency transition latency is of the order of micro-seconds whereas the shortest duration between frequency changes in the controller is 200ms. Finally, the power overhead for changing CPU frequencies is 14mW. In summary, the implementation overhead for the controller is negligible even when the number of system configurations is large.

4.4.2 Application Scope

Not all applications are amenable to the solution in its current form. Two types of applications are identified that are not well suited for the current strategy.

The first type includes applications for which the default CPU governor either selects the lowest frequency most of the time due to low CPU requirements or the highest frequency most of the time due to CPU-intensive computations. For the former case it is hard to obtain additional energy savings through CPU DVFS and for the latter it is hard to save more energy without performance degradation. For such applications, other components of the system such as network packet transfer rate etc. should be explored to save energy. The controller framework, as mentioned in Section 4.3, is generic enough to be able to control other parameters.

The second type includes applications with multiple rapidly varying phases (e.g. MobileBench browser benchmark), i.e., the application has very different CPU, memory, or I/O characteristics at different points in time. These applications pose a few very challenging problems. Firstly, how should application phases be defined and identified? This problem has been studied earlier on desktops/servers [130] and for simulators [131]. For example, in [130] six phases were defined based on the ratio of "memory access / uop". A study investigating whether this kind of metric can be used to classify application phases on the target platform is yet to be done. A practical concern is the lack of OS and/or hardware support for PMU counters. More serious problems are caused by the fact that the duration of phases could be very short. In such situations, experiments show that PMU-based performance measurements could have large variations, which in turn could misguide the controller. Furthermore, the shorter the duration, the more difficult it is for the controller to catch up. Phase prediction, as proposed in [130], might help, but is only a small step towards addressing these problems.

4.4.3 Effect of Different Background Loads

Section 4.3.1 discusses the issue of discrepancy between controller run-time environment and the profiling environment. In what follows, the controller performance is evaluated under different loading scenarios. The controller is tested under two different run-time conditions: (1) No-Load (NL) and (2) Heavier-Load (HL) while utilizing the offline profiling data and target performance obtained under the baseline load (BL).

Table 4.4: Summary of performance difference and energy savings obtained for the tested applications under Baseline Load (BL), No Load (NL), Heavier Load (HL) conditions

App Name	Performance (%)			Energy (%)		
	BL	NL	HL	BL	NL	HL
VidCon	0.8	0.2	-8.0	25.3	28.0	11.4
MobileBench	4.0	-3.5	-2.0	15.3	-4.9	4.6
AngryBirds	0.6	1.0	-2.0	14.9	12.8	10.0
WeChat Video Call	-0.4	2.0	3.6	27.2	19.4	27.0
MX Player	0.0	0.0	0.0	5.0	2.9	5.0
Spotify	9.3	-1.7	-1.3	31.6	7.2	6.0

In the NL condition only the application being controlled runs on the phone. In HL, a few more applications as compared to BL are opened but minimized. The background applications are: Gallery, eBook Reader, Chrome browser, FaceBook, e-Mail client, MX player and Spotify. WiFi is turned ON for both loading scenarios. It is noted that the most significant difference among the different loads is the memory usage. The amount of free memory is 500 MB, 1 GB, and 134 MB, for BL, NL and HL respectively. In contrast, the corresponding CPU loads as indicated by the file `/proc/loadavg` are similar: 6.3, 6.7, and 6.6 respectively.

Table 4.4 shows the controller's performance and energy results in the three different loading conditions. In 4 cases, i.e., VidCon, AngryBirds, WeChat, MX Player, the controller performs relatively well in terms of energy savings when running under an environment different from the profiled environment. VidCon under HL test condition experiences a performance loss of 8% but still achieves 11.4% energy savings. The controller performs the best for WeChat in NL and HL, saving 19% and 27% energy respectively.

Spotify displays a significant decrease in energy savings in both NL and HL. On further analysis, it is found that in NL and HL, the default governor uses CPU frequency No. 10 less than 10% of the run-time as compared to 25% with the baseline load. This directly translates to lower overall power consumption of 1.43W in NL and HL, versus 1.7W with the baseline load. The average power consumed by Spotify with the controller is 1.3W for all the loading cases which results in the varying energy savings shown in Table 4.4.

MobileBench browser has rapidly varying GIPS and power data on account of multiple websites being loaded in quick succession. Due to the lower bound on the time taken to measure application performance (200ms), the controller is unable to respond to these rapid variations. While the average power in the NL case is similar to the average power consumed by the default governor, the performance loss of 3.5% leads to the excessive energy consumption by the controller.

Although in a majority of cases the profiling data obtained under baseline load can be used to achieve good results in different load conditions, it is observed that better results can be achieved if the profiling condition closely matches the run-time environment. As an example, MobileBench was re-profiled for the NL case and the controller is re-tested, this time with a new target performance obtained from the offline data. The controller now saves 11.1% energy with no performance loss. A possible approach is to profile the application under a few different background loads and let the controller select the appropriate offline data by measuring the background load at run-time.

Note that the performance and power data for NL has the same trend as that for BL but with a small increase in the absolute value. A new method is envisioned which involves a power and performance model which uses the system load as the variable parameter. At run-time, the controller can track the background load and, using the models, generate power and performance data for different configurations. Such an approach would not require additional profiling thereby expanding the scope of the proposed method. These topics are elaborated on in the next chapter.

4.4.4 Comparison with CPU-only DVFS

To evaluate the effectiveness of coordinated control of CPU frequency and memory bandwidth, another version of the application-specific controller is created which controls only the CPU frequencies and allows the memory bandwidth to be controlled by the default governor, i.e., `cpubw_hwmon`. The controller does not communicate with the default memory bandwidth governor and hence takes decisions in an independent and isolated manner.

For this controller, the applications are re-profiled with CPU frequency set to fixed values while memory bandwidth is left in the control of the default governor. For each application, the same set of CPU frequencies as in the coordinated controller case is selected. Table 4.5 lists the energy savings and performance of the 6 tested applications when only

Table 4.5: Summary of performance difference and energy savings obtained by the CPU-only DVFS controller

Application Name	Performance	Energy
VidCon	2.8%	13.1%
MobileBench	-2.9%	7.6%
AngryBirds	-2.6%	9.6%
WeChat Video Call	4.7%	22.3%
MX Player	0.0%	0.4%
Spotify	3.3%	33.3%

the CPU frequencies are controlled. Excluding MX Player which practically does not save energy, on an average, a 53% increase in energy consumption is observed as compared to the coordinated control of CPU frequency and memory bandwidth. For WeChat and Spotify, the CPU frequencies chosen and their durations are similar. For other applications however, the default bandwidth governor selects a higher-than-necessary bandwidth for over 60% of the application run-time thus resulting in a higher power consumption. In AngryBirds, for example, the bandwidth governor increases the bandwidth to the highest whenever advertisements are loaded between game levels, which results in a peak power of 6W. In general, it is observed that CPU-BW DVFS controller trades higher CPU frequency over higher bandwidth at the same CPU frequency which is a direct consequence of the profiling table (see Fig.4.5). For example with Mobilebench, the average power and performance for the pair of CPU frequency and memory bandwidth $(7, 13)$ is $(2.128, 2.687)$ while the same parameters for the pair $(11, 1)$ are $(2.125, 2.9705)$. The controller chooses (11,1) rather than (7,13) because for the same power consumption the performance of (11,1) is much higher. This is exactly why the controller chooses the bandwidth No. 1 for over 60% in all 6 test cases.

4.5 Summary

In this chapter a key observation is that the default DVFS governors on current Android mobile devices are designed for general-purpose usage, focus on power savings, and are in general not energy-optimal for many applications. The need for investigating an application-specific energy optimization strategy is established and it is stressed that any energy optimizer should be mindful of performance impacts. Furthermore, the advantage of a coordinated control of different components such as CPU and memory is highlighted. A detailed description of the application-specific, performance-aware energy optimization solution targeting Android devices is then presented. The solution is implemented on a Nexus 6 smartphone and tested with 6 real-world applications, including highly popular ones. Energy savings in the range of $4-31\%$ is achieved with a worst-case performance loss of less than 1%.

CHAPTER 5
COORDINATED CONTROL: GENERALIZATION TO MULTI-CORE MULTI-MEMORY-CONTROLLER SYSTEMS

The feedback control framework described in CHAPTER 4 has its limitations. The gamut of potential applications is reduced due to the dependence on application-specific performance and power models. An ideal scenario is one where performance and power can be derived or estimated via online measurements. Working towards this goal, in this chapter, an improved feedback controller is developed. Starting from a simple single-core single-memory-controller architecture, models for performance and power are arrived at through regression. A software controller implementing DVFS on the core and the memory controller is tested on a cycle-level simulator. EDP for all possible combinations of core and memory controller frequencies is obtained offline. Results show that the coordinated controller chooses a voltage-frequency combination for the core and the memory controller that compares favorably against EDP values obtained offline.

The second half of this chapter deals with further extending the application agnostic feedback controller to a more generic multi-core multi-memory-controller architecture. Following the exploration *algorithm* (Fig. 1.1), a microbenchmark characterization of a two-core two-memory-controller system is conducted. This exploration reveals the dependence of performance, measured as MIPS, on: (1) the distribution of memory requests from a core, and (2) distance between a core and a memory controller. A per-core feedback controller minimizing EDP is tested on a cycle level simulator. Analogous to the evaluation method for the single-core single-memory-controller system, the coordinated controller compares well against EDP obtained offline. Finally, implementation of such a technique on a real physical system with several cores and memory controllers is discussed.

5.1 Overview

Although processors have historically dominated power consumption, the portion of total power that can be attributed to the memory system has gradually been increasing. In 2010, for a high performance server system, main memory accounted for 40% of the total consumed power [132]. To utilize idle low power modes, past research has explored energy minimization by creating idle periods via intelligent scheduling and batching of memory requests, layout transformation etc. [133, 134, 135, 136, 137, 138]. Apart from the self refresh, dynamic power is dissipated in the DRAM only when it is accessed. Therefore, reducing the total number of DRAM accesses itself, has also been explored [139, 140]. Some others have explored modifying the DRAM microarchitecture itself [141, 2]. DVFS based techniques, similar to algorithms implemented on a processor have also been explored. For example, [108] proposes a heuristic DVFS based scheme to minimize energy consumed by the memory controller, memory bus and the DRAM. The authors of this work allow the end user to set the performance penalty. Lower the penalty, higher is the energy saved. Another promising approach is to enter low power modes *actively* by trading performance [123]. A common thread for all the works listed here is that they focus only on DRAM power management.

The focus is slowly shifting towards cooperative or coordinated management of the processor, memory and other related components. The work in [23] utilizes ideas developed in [108] and demonstrates a heuristic approach to minimize energy consumed under performance constraints. A notable feature of the work in [23] is that the controller is centralized.

Processors have evolved from single core to multi-core. Similarly, computers of today have multiple memory controllers to manage memory requests from a large number of cores. The problem to be solved remains the same; minimizing energy subject to performance constraints or minimizing EDP. Working towards this goal, the work in [142] proposes a technique that implements per-memory-controller DVFS. By monitoring per application traffic across different memory controllers and estimating the bandwidth requirements, a linear program based approach is used to minimize memory system energy. Once again, this work focuses *only* on the memory system.

The goal of this chapter is to go two steps further than the state-of-the-art adopting the following two steps:

1. Make the controller distributed.

2. Control the processor and memory simultaneously.

Throughout this chapter, changing memory controller frequency implies changing the frequency of the memory controller, memory bus and the DRAM internal timings. Furthermore, memory frequency and memory controller frequency are used interchangeably. This chapter first develops a performance and power model for a single-core single-memory-controller system and then details the design of a feedback controller to minimize EDP for the same. Next, a microbenchmark characterization of performance of a two-core two-memory-controller system is discussed. This characterization helps the development of a simple per-core performance model. A distributed controller implemented on a cycle level simulator for a four-core two-memory-controller system is subsequently presented.

5.2 Memory Controller Configurations

A typical DRAM datasheet shows the different speed grades for a DRAM DIMM (Dual-Inline-Memory-Module). For example, DDR3 DIMMs can be operated at 400MHz, 533MHz, 667MHz and 800MHz. A higher speed grade DRAM can be operated at a lower speed grade i.e. a DIMM rated for 800MHz can be operated at 400MHz. For each operating frequency, the internal timing is very well defined. Table 5.1 lists all the relevant timing information for the 4 different DRAM frequencies. Changing the memory frequency linearly scales the available bandwidth and the power consumed by the DRAM [143]. Although the RAS, CAS delays etc. change for different memory frequencies, the wall-clock latency for a given request remains the same. The reader is referred to Section 2 in reference [108], for a detailed explanation of memory system performance and power consumption.

Table 5.1: DDR3 Timing parameters for different speed grades.

Parameter	800MHz	667MHz	533MHz	400MHz	Units
tCK	1.25	1.5	1.87	2.5	ns
CL	11	10	8	6	CLK
tRAS	28	24	20	15	CLK
tRCD	11	10	8	6	CLK
tRRD	5	5	5	5	CLK
tRC	39	34	28	21	CLK
tRP	11	10	8	6	CLK
tCCD	4	4	4	4	CLK
tRTP	6	5	5	4	CLK
tWTR	6	5	5	4	CLK
tWR	12	10	9	6	CLK
tRTRS	1	1	1	1	CLK
tRFC	88	74	59	44	CLK
tFAW	24	20	20	16	CLK
tCKE	4	4	3	3	CLK
tXP	5	4	3	3	CLK

5.3 Performance and Power Model for a Single-Core Single-Memory-Controller System

Figure 5.1 shows the variation of performance of a core (measured in MIPS) as a function of core frequency. Performance of compute intensive application scale almost linearly with core frequency where as memory bound workloads saturate. There are also applications that are a mix of compute and memory.

To show the effect of memory frequency variation on performance, a memory bound benchmark is run at 800MHz and 400MHz (See Figure 5.2). As much as 30% reduction in performance can be observed when changing memory frequency from 800MHz to 400MHz. Clearly, performance of a core is a function of the core clock frequency *and*

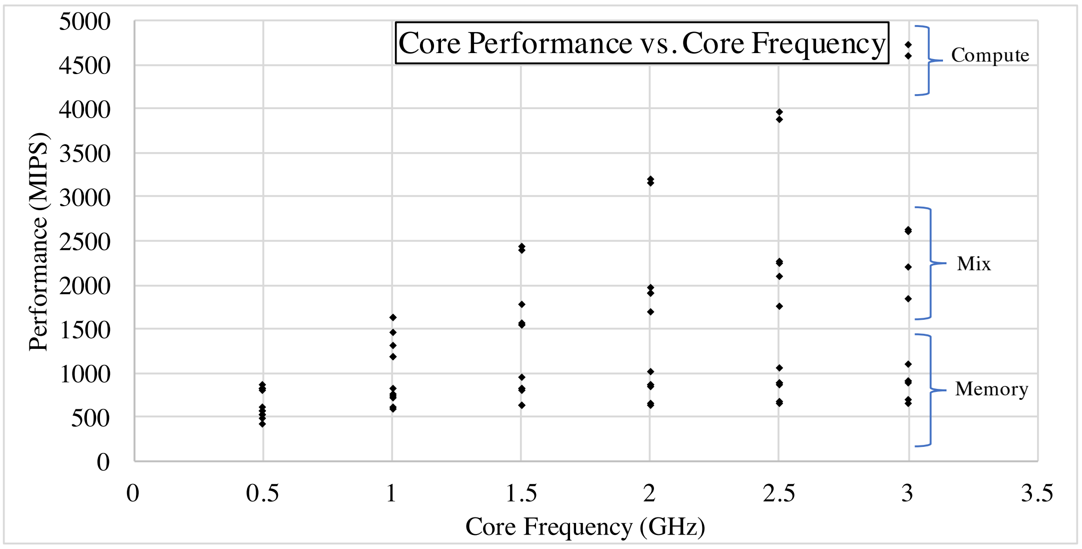


Figure 5.1: Performance model for a single-core single-memory-controller system.

the memory frequency. This aspect lays the groundwork for defining *coordinated control*. Using the Curve-Fitting toolbox in MATLAB, a regression based model for performance χ is derived and is described below:

$$\chi(f_{core}, f_{mem}) = \alpha f_{core}^{\beta} + \gamma f_{mem} \tag{5.1}$$

where α, β and γ are positive constants, f_{core} and f_{mem} are core and memory frequencies, respectively. As seen in Fig. 5.1, applications or regions within an application can have varied characteristics that are called “phases”. For example, a compute bound phase demands greater compute resources (higher clock frequency) whereas a memory bound phase demands greater memory resources (larger memory bandwidth). Performance of the application improves if the demand is satisfied appropriately. Application phases vary at run-time and prior research ([130] and references there-in) have investigated methods to track application phase change. Following the work in [130], this thesis classifies application phases into three categories at run-time using the ratio of number of retired instructions to the bytes of data transferred between the last level cache and main memory, also referred to as ops/byte. Therefore, three equations for performance are obtained, one for each cate-

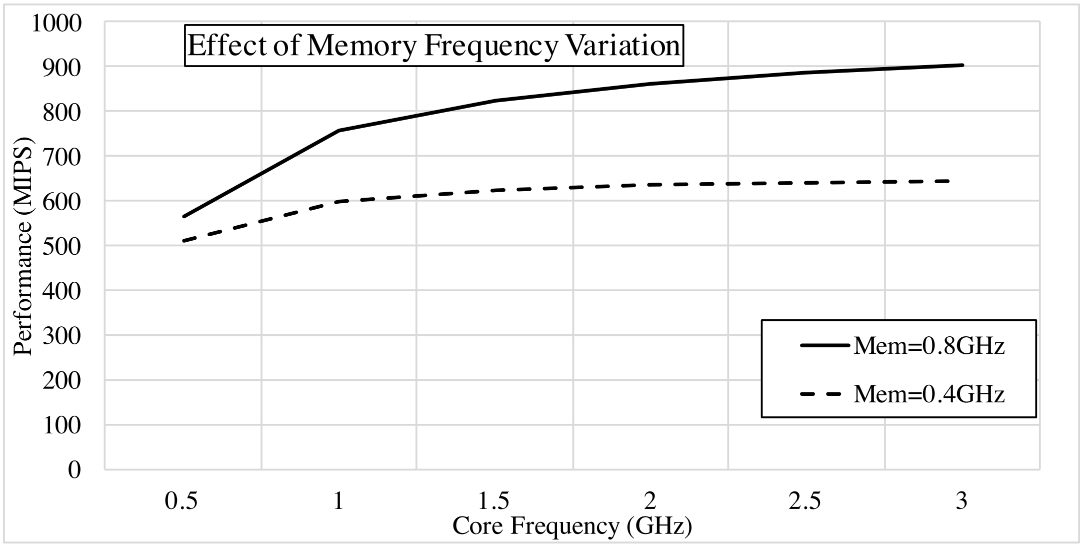


Figure 5.2: Performance of a memory bound workload for different memory controller frequencies.

gory. The parameters α, β and γ for the three categories are shown in Table 5.2. The range of ops/Byte used to classify the application phase is as follows: (i) $(.) \leq 1$ for Memory, (ii) $1 < (.) < 3$ for Mix and (iii) $3 < (.)$ for Compute.

Table 5.2: Performance model parameters

Parameter	Compute	Mix	Memory
α	1388	1005	422.3
β	0.85	0.52	0.34
γ	84.01	514.40	418.67

The model for power is split into two parts: (i) Core and (ii) Memory. The core power model follows the well known cubic relation $P \propto C(V^2 f)$ and is given by

$$P_{core}(f_{core}) = \alpha_1 f_{core}^3 + \beta_1 f_{core} + \gamma_1 N_{reqs} + \delta_1 \tag{5.2}$$

where N_{reqs} is the number of requests from the last level cache to the memory and α_1, β_1, γ_1 and δ_1 are positive constants. The L2 cache power is also considered as part of the core. Hence N_{reqs} is included in the core power model. The memory power model is proposed as follows

$$P_{mem}(f_{mem}) = \alpha_2 f_{mem} N_{reqs} + \beta_2 \tag{5.3}$$

where α_2 and β_2 are positive constants. It is to be noted that in both Eqns. 5.2 and 5.3, all the constants are obtained by offline characterization whereas f_{core}, f_{mem} and N_{reqs} can be obtained at run-time, thus making the optimization technique described in the next section, application-agnostic.

5.4 Optimization Problem: Single-Core Single-Memory-Controller

The metric chosen for optimization is Energy-Delay-Product (EDP). The objective is to minimize EDP by choosing a combination (f_{core}, f_{mem}) at run-time. Similar to the optimization problem in CHAPTER 4, coordinated control in this context refers to augmenting f_{mem} with f_{core}. When performance is measured in MIPS, minimizing EDP is equivalent to minimizing the ratio Power / Performance^2 i.e.

$$\begin{aligned} &\min_{(f_{core}, f_{mem})} \frac{P_{core}(f_{core}) + P_{mem}(f_{mem})}{\chi^2(f_{core}, f_{mem})} \\ &\text{subj. to} \quad \underline{f}_{core} \leq f_{core} \leq \overline{f}_{core} \\ &\qquad\qquad \underline{f}_{mem} \leq f_{mem} \leq \overline{f}_{mem} \end{aligned} \tag{5.4}$$

The underline and the bar indicate minimum and maximum values, respectively. The numerator and denominator are always positive and monotonically increasing with respect to both f_{core} and f_{mem}. Therefore, the cost is convex and the optimization problem is well posed. Various constraints on power, performance and temperature can very well be included but for the purposes of this thesis, only a simplified problem is considered.

5.5 Solution Strategy

The solution approach is described in Figure 5.3. Periodically [1], at application run-time, f_{core}, f_{mem}, χ and ops/Byte are measured. Based on the ops/Byte range, the application phase and therefore the appropriate performance equation is selected. Coupled with the power models, the cost can now be minimized. The EDP is evaluated for every possible combination of f_{core} and f_{mem}. On real physical systems since the domain of f_{core} and f_{mem} is discretized, this approach is feasible. The combination that gives the least EDP is then applied. It is noted that such a strategy becomes intractable if the number of combinations is too large.

In contrast, the authors in [130] compute the ideal system configuration for each application phase *apriori*. At run-time, depending on the phase detected, the appropriate configuration is applied. This process is repeated every 100 million instructions.

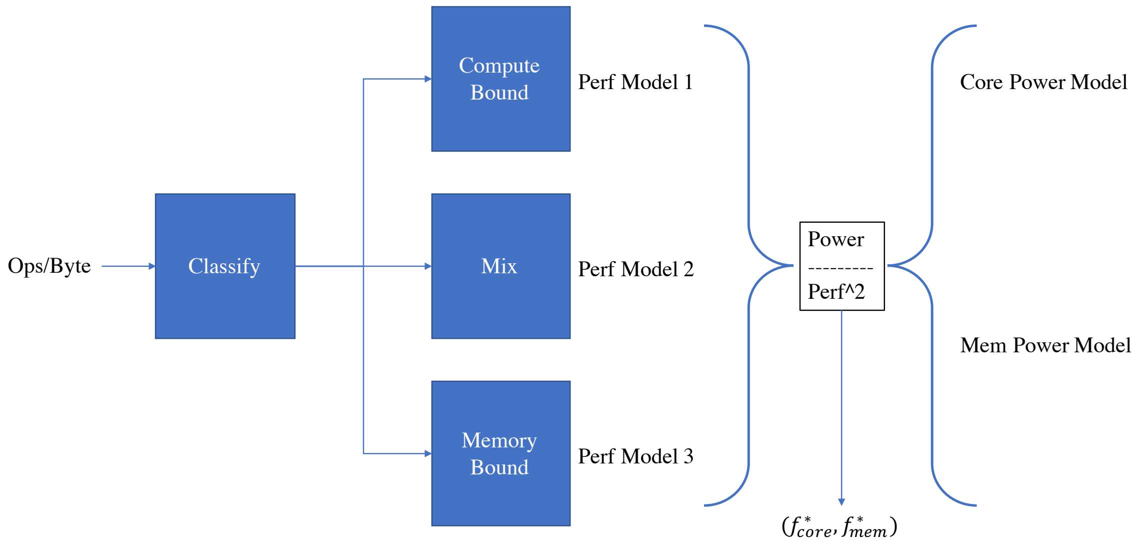


Figure 5.3: Solution strategy for optimizing EDP in a Single-Core Single-Memory-Controller system.

[1] For the experiments in this chapter, the period or control cycle is set to 1ms.

5.6 Results

A cycle level simulator is configured to simulate a single Out-of-Order with two level cache hierarchy connected to a 4GB DRAM DIMM via a single network router. DRAMSim2 [144] is modified to support multiple DRAM frequencies at run-time and also report power consumed in each rank. The DRAM uses an open-page policy and is configured to operate at 4 different frequencies. The core on the other hand is capable of operating in a frequency range of 0.5GHz to 3GHz, each frequency separated by 50MHz. Six benchmarks from the PARSEC, Splash2x and GraphBig [118], [145] are used for the experiments. To show that the controller indeed selects the core and memory frequency combination that minimizes EDP, each benchmark is executed at a fixed configuration and the corresponding EDP is calculated. The EDP obtained by the controller is compared with the values generated offline. The results are shown in Figure 5.4.

EDP is represented on the y-axis and the configuration (f_{core}, f_{mem}) in GHz is on the x-axis. EDP obtained by the controller is shown in red. The first observation is that every benchmark has a unique EDP signature. While `blackscholes` and `streamcluster` have the least EDP close to the configuration $(3.0, 0.4)$GHz, `kcore` has the least EDP close to $(1.5, 0.8)$GHz. The lowest configuration $(0.5, 0.4)$GHz gives the worst EDP for all the applications. Operating at a lower clock frequency definitely saves a lot of power but does so at the cost of performance. Applications running longer at low power result in higher EDP. In contrast, for all applications except `blackscholes` and `streamcluster`, the highest configuration $(3.0, 0.8)$GHz is also not ideal. In this case, although the application run-time is the shortest, the amount of work done is the same but at a much higher power consumption. Most of the power is lost in leakage therefore leading to a higher EDP. Through the performance and power models, the controller, cognizant of the application phases, tunes the core and memory clock frequencies appropriately and achieves up to 7% better EDP than a statically determined configuration. On an average, compared to

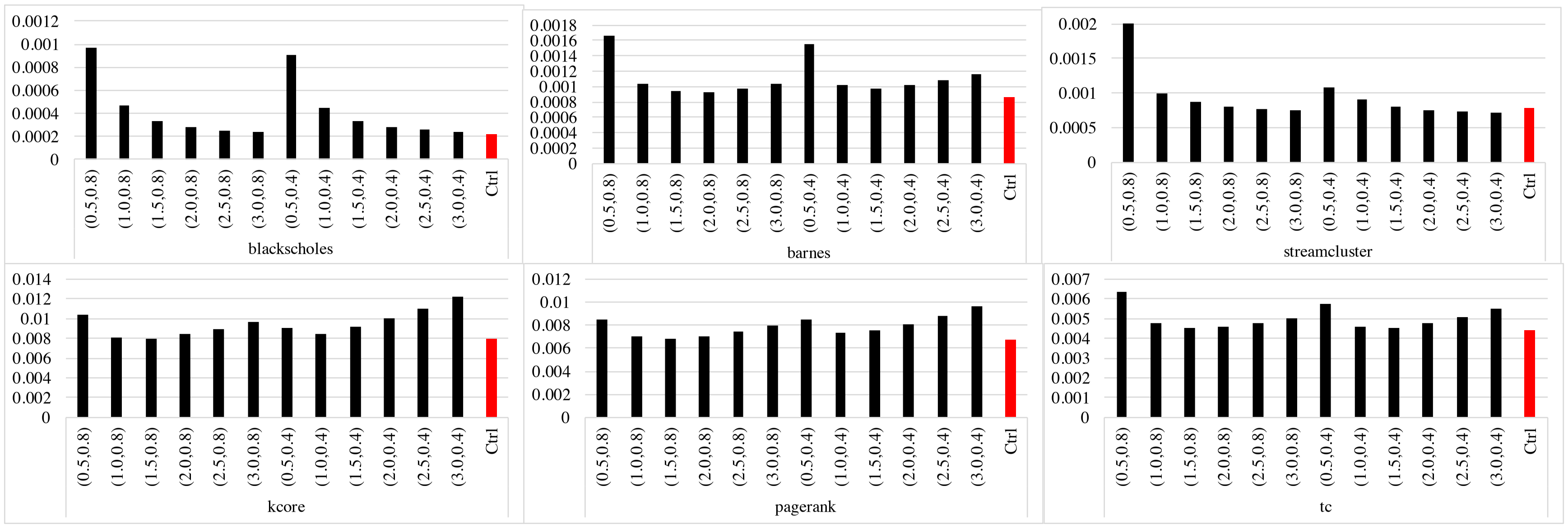


Figure 5.4: EDP Comparison: Single-Core Single-Memory-Controller

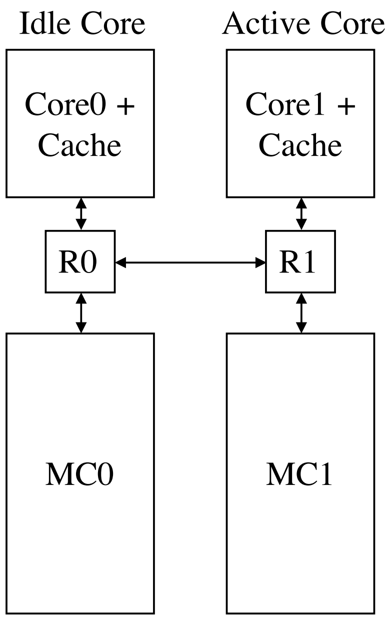


Figure 5.5: Two-core two-memory-controller system for microbenchmark characterization.

the highest configuration $(3.0, 0.8)$GHz, the controller performance is 13% lower. This, however, is not a limitation of the controller. It should be noted that since the objective is minimizing EDP, it comes almost always at the cost of reduced performance.

5.7 Microbenchmark Characterization: Two-Cores Two-Memory Controllers

Most prior works consider a single memory interface and consequently, all their approaches are aimed towards a single memory controller. However, as recent trends suggest, hardware manufacturers are moving towards Chip Multi Processors with multiple on-die memory controllers [146, 147, 148]. Furthermore, traffic patterns going to individual memory controllers are expected to be skewed with configurations such as multi-socket processors, heterogeneous architectures with multiple OoO cores and in-order cores alongside GPUs etc. These trends call for architecture-aware distributed energy/power management solutions.

As a first step in that direction, this section characterizes the performance of an application thread running on a two-core two-memory-controller system using targeted microbenchmarks (See Figure 5.5). The microbenchmark runs on Core$_1$ addressing memory controllers MC0 and MC1. Any memory request coming from Core$_1$ addressing MC0 goes through the following path $R1 \rightarrow R0 \rightarrow MC0$. The microbenchmark parameters varied are (i) ops/Byte and (ii) percentage of requests going to MC0 and MC1. The results are shown in Figure 5.6.

The y-axis in Fig. 5.6 represent performance measured in MIPS and the x-axis is grouped into different memory addressing patterns (MC0 : MC1). For example, Div 30:60 implies 30% of the requests from Core$_1$ are addressed to MC0 and 60% are addressed to MC1. The different colors of the bars represent different memory controller frequencies in GHz. For the results shown in Fig. 5.6, the Core$_1$ clock frequency is fixed at 3GHz. For memory intensive workloads, the effect of memory frequency variation on performance is more pronounced than for a compute intensive benchmark. Furthermore, the degree of performance variation increases as more requests are addressed to MC1 as opposed to MC0. The reason for this phenomenon may be traced to increased network delays. As mentioned earlier, requests from Core$_1$ to MC0 have to travel through 2 network routers. To understand this better, Figure 5.7 plots the sensitivity i.e. $\frac{dMIPS}{df_{mem}}$ as a function of ops/Byte and memory access patterns.

The x-axis is first grouped into different memory addressing patterns (MC0 : MC1). Each color in Fig. 5.7 represents a different ops/Byte ratio. The left y-axis is the following difference: Performance(f_{mem} = 800MHz) - Performance(f_{mem} = 400MHz). The right y-axis represents the standard deviation of these differences for each memory addressing pattern group. The important observations from this graph are the following:

1. Performance sensitivity to memory frequency reduces as distance between core and memory controller increases.

2. Performance sensitivity variation increases as a greater percentage of memory requests are addressed to a local memory controller.

Accordingly, the equation for performance of a Core$_i$, $i \in \{0, 1\}$ as a function of

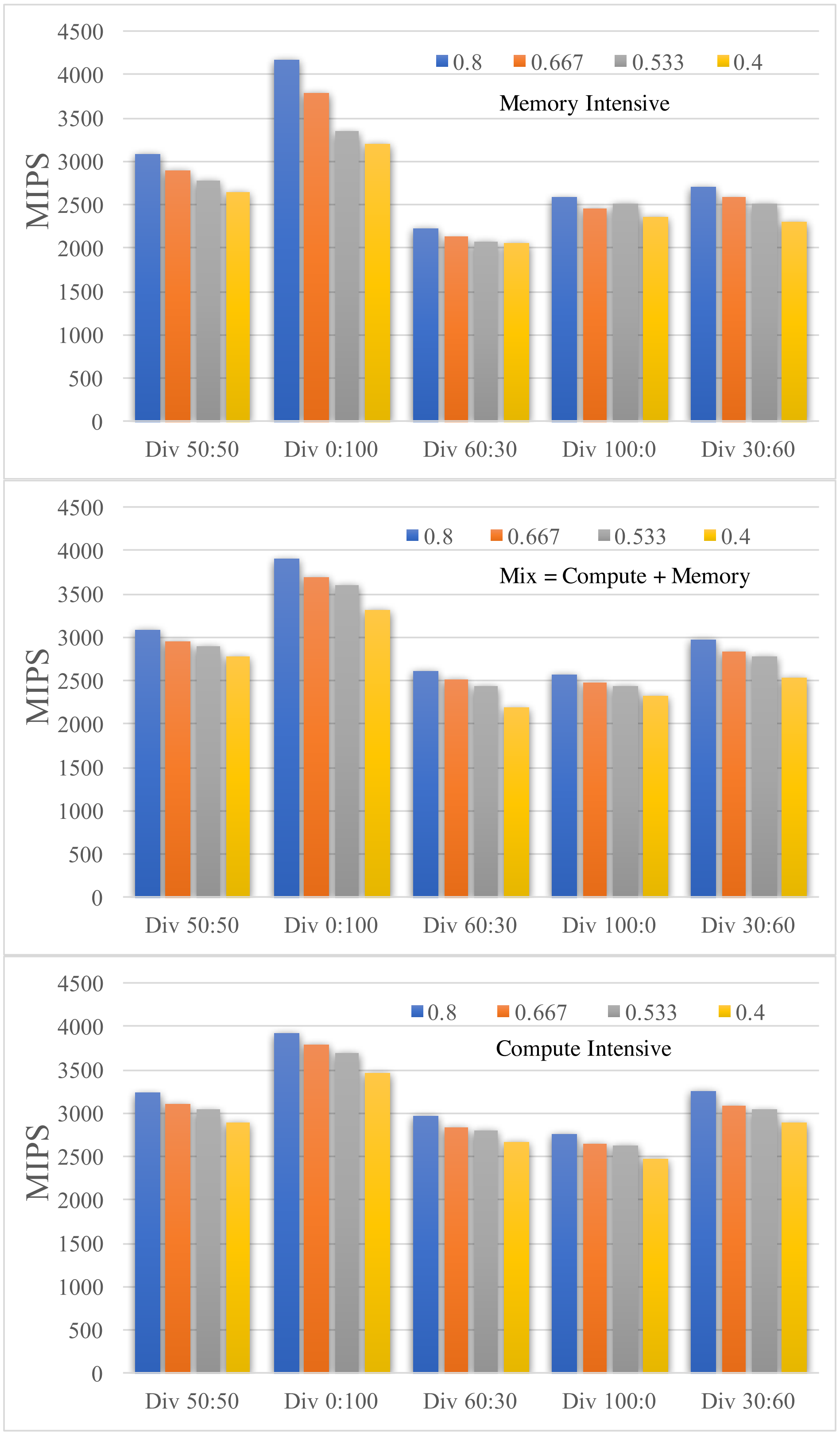


Figure 5.6: Performance of a microbenchmark for different ops/Byte and memory addressing patterns.

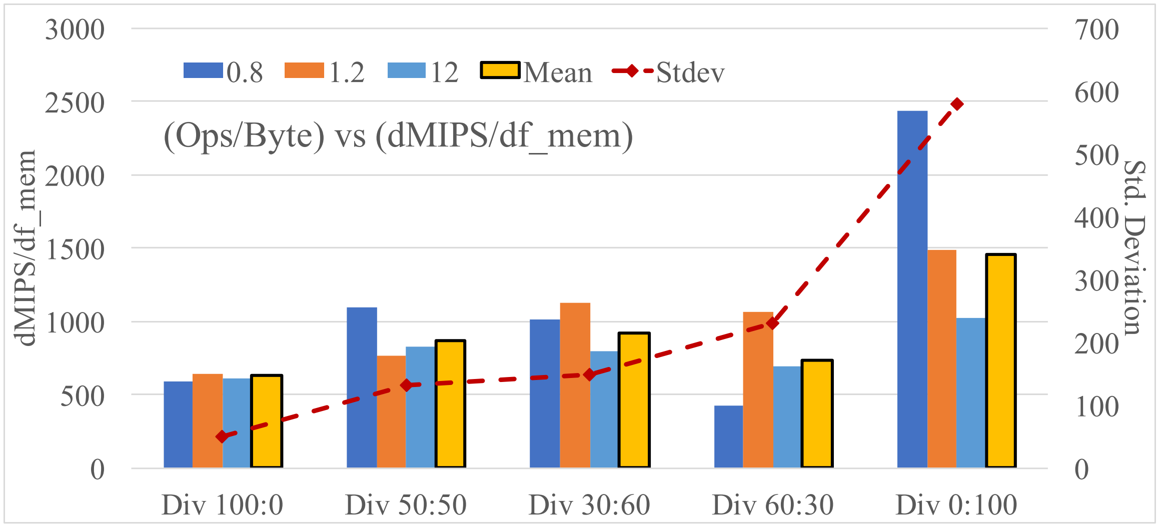


Figure 5.7: Performance sensitivity graph

memory controller frequencies is modified as follows:

$$\chi_i(f_{core_i}, f_{mem_0}, f_{mem_1}) = \alpha_{1_i} f_{core_i}^{\beta_{1_i}} + w_{0_i}\Gamma_{0_i}N_{0_i}f_{mem_0} + w_{1_i}\Gamma_{1_i}N_{1_i}f_{mem_1} \tag{5.5}$$

f_{mem_0} and f_{mem_1} are the memory frequencies of MC0 and MC1, respectively. w_{0_i} and w_{1_i} are parameters related to the number of network hops needed to reach a memory controller. For example, $w_{0_1} = 0.5$ and $w_{1_1} = 1$. This parameter captures point (1). N_{0_i} and N_{1_i} are simply the percentage of memory requests from Core$_i$ to MC0 and MC1 respectively. Note that $N_{0_i} + N_{1_i} = 1$ for $i \in \{0, 1\}$. This parameter captures point (2). The last parameter Γ_{0_i} is a curve fitting parameter. It is easily observed that Equation 5.5 is similar to Equation 5.4. The α_1 and β_1 parameters are sub-indexed with i because the ops/Byte of the each application thread can vary.

Thus, the per-core performance and power model and the power model for the memory controller are completely determined by parameters measured at run-time and parameters calculated by offline regression.

5.8 Optimization Problem: Four-Cores Two-Memory-Controllers

The two-core two-memory-controller microbenchmark analysis provides important observations that help in the development of a per-core performance model. Taking the optimization approach from Section 5.4 forward, a four-core two-memory-controller system is configured as shown in Figure 5.8.

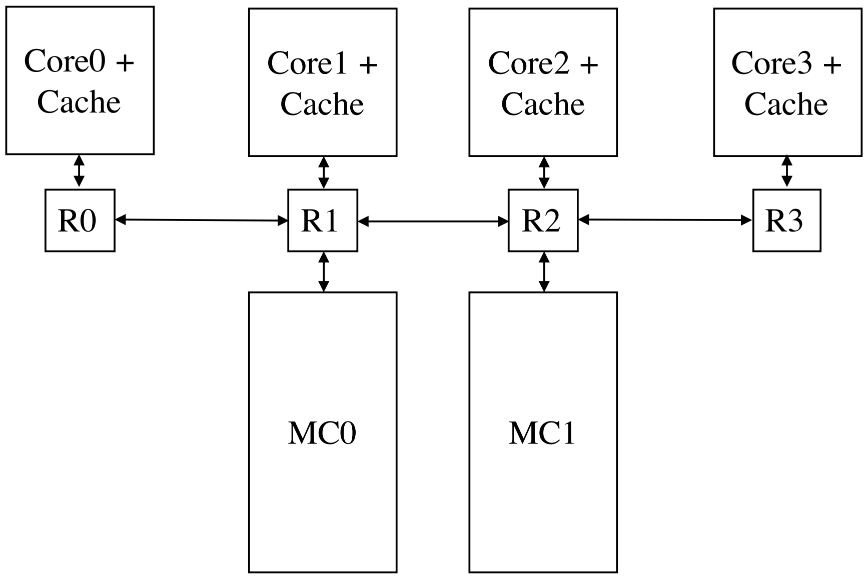


Figure 5.8: Four-core two-memory-controller system.

The objective is to minimize EDP for the entire system i.e. $\min \sum_{i=0}^{3} EDP_i$ where $i \in \{0, 1, 2, 3\}$ indexes the number of cores. Solving the optimization in a centralized fashion would result in an optimal configuration of core and memory controller frequencies. However, such a scheme will not scale. Therefore, a per-core EDP minimization is chosen. This could potentially be a sub-optimal approach but it is practically viable. The optimization problem for each Core$_i$ is defined as follows:

$$\min_{(f_{core_i}, f_{mem_0}, f_{mem_1})} \frac{P_{core_i}(f_{core_i}) + P_{mem_0}(f_{mem_0}) + P_{mem_1}(f_{mem_1})}{\chi^2(f_{core_i}, f_{mem_0}, f_{mem_1})} \tag{5.6}$$

$$\text{subj. to} \quad \underline{f}_{core} \leq f_{core_i} \leq \overline{f}_{core}$$

$$\underline{f}_{mem} \leq f_{mem_0} \leq \overline{f}_{mem}$$

$$\underline{f}_{mem} \leq f_{mem_1} \leq \overline{f}_{mem}$$

5.9 Solution Strategy

Each core solves the optimization problem in Equation 5.6 and arrives at the configuration $(f^*_{core_i}, f^*_{mem_0}, f^*_{mem_1})$. *Coordination* in this context refers to augmenting $f^*_{mem_0}$ and $f^*_{mem_1}$ with $f^*_{core_i}$. While f^*core_i is unique to the core, $f^*_{mem_0}$ and $f^*_{mem_1}$ is not. For this particular system with 4 cores and 2 memory controllers, each memory controller has to make a choice between the different frequencies demanded by each core. Each memory controller computes a weighted average of the memory frequencies demanded using the weight parameters w_{0_i} and w_{1_i}. The resulting weighted average is the clock frequency applied to the corresponding memory controller. The algorithm is described in Figure 5.9.

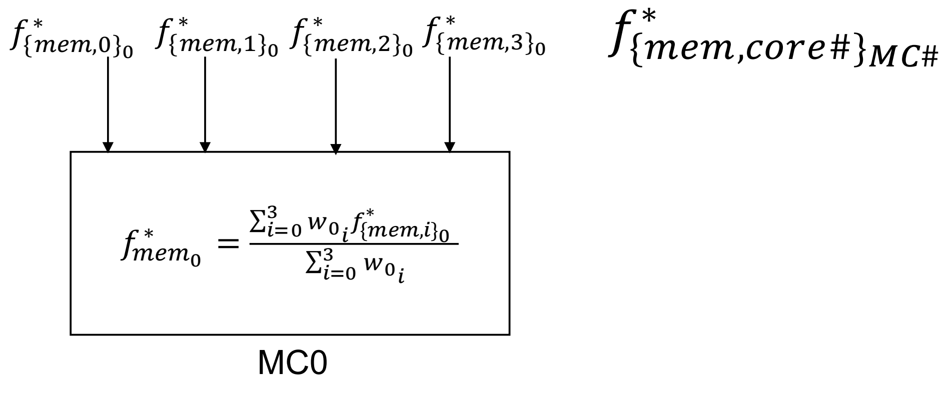

$f^*_{\{mem,0\}_0}$
$f^*_{\{mem,1\}_0}$
$f^*_{\{mem,2\}_0}$
$f^*_{\{mem,3\}_0}$
$f^*_{\{mem,core\#\}_{MC\#}}$
$f^*_{mem_0} = \frac{\sum_{i=0}^{3} w_{0_i} f^*_{\{mem,i\}_0}}{\sum_{i=0}^{3} w_{0_i}}$
MC0

Figure 5.9: Memory controller arbitration algorithm.

5.10 Results

A cycle level simulator is configured to simulate a system as shown in Figure 5.8 with 2 DRAM DIMMs of 2GB each. The same benchmarks mentioned in Section 5.6 are used here as well[2]. Each application is run with 4 threads, equal to the number of cores. It is observed that the memory traffic is equally spread out between MC0 and MC1 for all the benchmarks i.e. $N_{0_i} = 0.5$ and $N_{1_i} = 0.5$ for $i \in \{0, 1, 2, 3\}$. Nevertheless, the effect of w_{0_i} and w_{1_i} is observed distinctly. Performance of Cores 1 and 2 is up to 15% greater than Core 0 and 3 for memory intensive workloads. For compute intensive benchmarks however, this difference is within $\pm 5\%$ which is an expected behavior. Along the lines of the analysis conducted for a single-core single-memory-controller system, each benchmark is run at fixed core and memory frequencies and system wide EDP is calculated. The results obtained with the controller are compared against the fixed configuration EDP values in Figure 5.10.

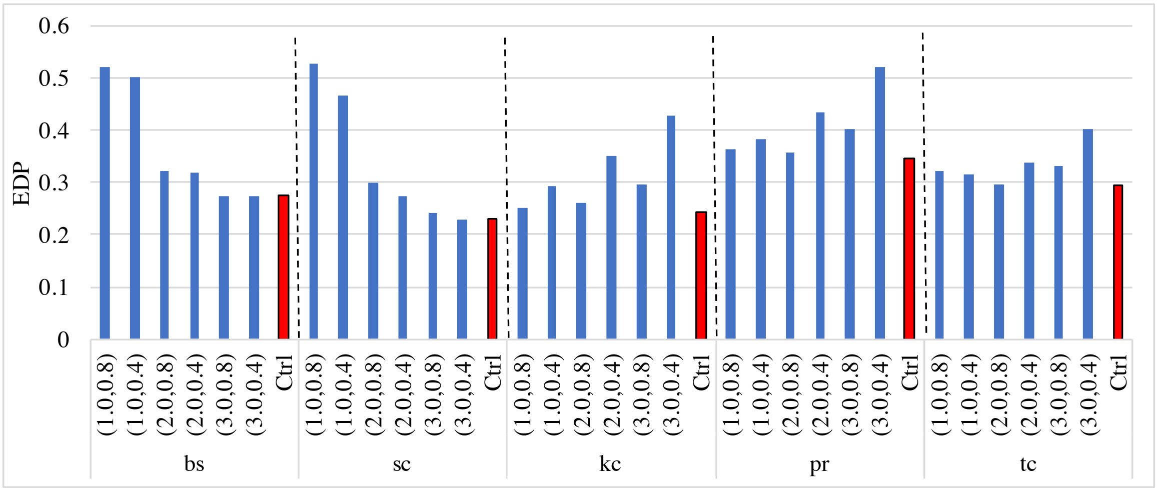


Figure 5.10: EDP Comparison: Four-Cores Two-Memory-Controller.

The EDP results are similar to what was observed in Section 5.6. For the set of benchmarks chosen, the arbitration at the memory controller turns out to be straightforward, in that, the per-core optimization demand the same memory frequency for both MC0 and

[2]`barnes` is excluded here due to inconsistent simulation behavior.

MC1.

5.11 Discussion

Consider the performance model for the core as described in Equation 5.1. At run-time, α_1, β_1 and γ_1 have to be selected based on the application phase which is parameterized by ops/Byte. Depending on the granularity required, more than 3 application phases can be chosen. Doing so would require more detailed offline characterization. Instead, a possible approach is to consider α_1, β_1 and γ_1 as functions of ops/Byte. This would make the online implementation simpler. The only values to be measured would be number of retired instructions and number of bytes transferred between the last level cache and the main memory. Remaining parameters can in-turn be inferred. This approach can also be extended to performance, as described in Eqn. 5.5.

There are many ways to solve the optimization problem described in Eqn. 5.4. Besides the approach explained in Section 5.4, the other possible options are

1. Gradient descent: This, however, involves many computations and could potentially take a long time to converge.

2. Pre-compute (f^*_{core}, f^*_{mem}) for each application phase and apply the appropriate combination at run-time. This approach is the fastest in terms of practical implementation since it is similar to a look-up table. Most of the computational burden is taken care of during the controller design phase.

Implementing the feedback controller on firmware requires further optimization of the controller code. For example, if the application phase does not change frequently, invoking the controller every X milliseconds or every Y million instructions will be redundant. Instead, determining the optimal DVFS states at the phase change boundaries is one option. Another approach is reducing the control cycle duration when application phases change rapidly and increasing the control cycle duration when the application resides in a phase for longer period of time. The former is ‘event-based’ control while the latter is a modified version of the traditional ’sample-based’ control. On low power mobile devices, such approaches certainly promise a reduction in the power consumed by the *always ON* microcontroller running the firmware. Detecting phase changes requires sampling performance counters at a high frequency. It follows from the Nyquist sampling criterion that there will always be some information that is missed between two successive sampling intervals. Furthermore, there is also a lower limit on how fast the controller can be invoked. This lower limit is decided by data acquisition delays, controller computation delays and DVFS actuation delays. Designing feedback controllers under such circumstances is a hard problem. The controllers described in this chapter are implemented considering the aforementioned practical constraints. For instance, at the beginning of each control cycle, if the application phase has not changed, the controller skips implementing the optimization algorithm thus saving both computation time and power. Dynamically expanding and contracting the control cycle duration will be considered in future work.

Next, consider the memory controller arbitration algorithm (Fig. 5.9). This algorithm does not scale well when the number of cores increase. The memory controller will have to wait to receive communication from each of the per-core controllers and then make a decision for itself. To tackle this issue, consider Fig. 5.7, where the influence of memory frequency on core performance is seen to be waning as the distance between said core and memory controller increases. Consequently, one possible solution is to consider only the group of cores within a 1-hop neighborhood of the memory controller. Using the insights from Fig. 5.7, hardware-software optimizations such as architecture and application phase aware thread migration can be considered for EDP minimization.

5.12 Summary

This chapter develops an application-agnostic EDP minimization technique. Regression based models for performance and power for a single-core single-memory controller system are constructed. Using parameters measured online such as ops/Byte, EDP for the whole system is minimized by selecting a combination of core and memory controller frequencies. Before extending the coordinated feedback controller to a multi-core multiple-memory-controller system, a microbenchmark characterization of performance is conducted. These experiments reveal the dependence of performance of a core on the (i) memory addressing patterns and (ii) distance between the core and memory controller. Per-core EDP minimization on a four-core two-memory-controller system shows that the controller is able to choose a combination of core and memory controller frequencies which gives the lowest EDP. A discussion on improving the performance model and memory controller arbitration is presented as well.

CHAPTER 6
THERMAL MANAGEMENT: 2D ARCHITECTURES

The previous two chapters analyzed the performance, power and energy consumption of a multi-core processor. Furthermore, CHAPTER 3 briefly mentions the problem of frequency throttling also referred to as ‘Thermal Throttling’ due to higher core temperatures. The end of Dennard scaling has led to increasing power densities on the processor die and consequently higher chip temperatures [149, 150]. Emerging and future processors are bound to be thermally limited and must operate within the cooling capacity of the chip package, which is typically represented by the maximum operating temperature. Dynamic Thermal Management (DTM) techniques have emerged to manage thermal behaviors but are challenged by a number of issues. In particular, the exponential dependence of static power on temperature limits the effectiveness of many existing DTM techniques. This coupling can also lead to thermal runaway that must be prevented by DTM to avoid damaging the chip. Furthermore, spatial and temporal variations in the thermal field degrade device reliability and accelerate chip failures. Similarly, rapid fluctuations in the thermal field referred to as thermal cycling, also cause thermal stresses that degrade device and hence chip reliability.

This chapter considers the problem of temperature *regulation* in multi-core processors via DVFS. A feedback law is proposed, that is based on an integral controller with adjustable gain, designed for fast tracking convergence in the face of model uncertainties, time-varying plants, and tight computing-time constraints. Moreover, unlike prior works, a nonlinear, time-varying plant model is considered that trades off precision for simple and efficient on-line computations. Cycle-level, full system simulator implementation and evaluation illustrates fast and accurate tracking of given temperature reference values, and compares favorably with fixed-gain controllers.

6.1 Overview

During early 2000s, as transistor count on microprocessors started to approach a billion, increased power densities started to push the thermal packaging limits. Higher performance was now achievable only with advanced cooling which further increased the total cost of the processor. To reign in the ‘temperature-problem’ a specific class of thermal regulation techniques were introduced. Activity management like instruction fetch throttling and clock gating [31, 30], thread migration (computations’ rescheduling) [96, 97], and core frequency scaling [151] were the initial efforts. In references [31, 30], PI and PID controllers have been proposed to slow down the rate of the instruction-fetch unit whenever the temperature exceeds a given upper bound, while in [96, 97], threads (computations) are scheduled from hot cores to cooler cores in an effort to maintain a balanced thermal field. Initial heuristic approaches started giving way to control-theoretic formalisms, with the aforementioned references [31, 30] providing the earliest examples. Subsequently, reference [74] considered a similar upper-bound regulation problem but used DVFS for temperature control. More recently [152] described a controller for regulating the fluid flow rates in a microfluidic heat sink based on the measured temperature as well as predicted temperature estimated from the projected power profile. Other works have investigated DTM under soft and hard real-time constraints [76, 77] seeking to satisfy thermal upper bounds while operating under scheduling constraints.

More recently, there emerged a number of new approaches, based on optimal control and optimization have been developed. Reference [35] minimizes a least-square difference between the working frequency and the frequency mandated by the operating system, subject to thermal and frequency constraints, by using model-predictive control (MPC). Reference [34] uses similar techniques to minimize the least-square difference between set power levels and actual power levels in a core. Reference [36] uses a combination of off-line convex optimization and on-line control to obtain uniform spatial temperature gradient across several cores in a processor. It is to be noted at this juncture that these references assume linear and time-invariant plant-models for their respective control systems; [34] updates the model on-line while [35, 36] do not. Finally, reference [75] minimizes energy consumption while preserving performance levels within a tolerable limit by employing separate Model Predictive Controllers for each core to ensure thermal safety, and updates, in real-time, the power-temperature model for the cores.

Besides the need to limit core and chip temperatures, there is a pressure to maintain temperatures close to package thermal capacity in order to maintain high levels of performance[1]. This is typically achieved by adjusting the rates of the processor cores as, for example, in Intel processors [51] and AMD processors [54]. Moreover, spatio-temporal variations in the thermal field generally impact device degradation and energy efficiency. For example, thermal gradients between adjacent cores on a die increase leakage power in the cooler cores, thereby increasing its temperature and reducing its energy efficiency (ops/joule) [106]. Further, the stresses introduced by the gradients reduce lifetime reliability by accelerating device degradation [153]. These affects are exacerbated in heterogeneous multi-core processors where cores of different complexities (and therefore thermal properties) are utilized to improve overall energy efficiency. Consequently, it has become necessary to be able to allocate and control the usage of thermal headroom in different regions of the die. Core-temperature regulation (and not only optimization) can provide an important means to this end.

The work in this chapter proposes an approach for regulating core temperatures by DVFS so as to track given reference temperature values (set points). The frequency is adjusted by an integral controller with adjustable gain, designed for fast tracking-convergence under changing program loads. Unlike the aforementioned references that are based on optimal control and optimization, this work considers a nonlinear, time-varying plant model that captures the exponential dependence of temperature on static power. The basic idea

[1]This aspect is challenged in the next chapter.

is to have the on-line computations of the integrator's gain be as simple and efficient as possible even at the expense of precision. This is made possible by a great degree of robustness of the tracking performance of the controller with respect to variations from the designed integrator's gain, which is observed from extensive simulations (see [32] for analysis and discussion). The efficacy of the proposed technique is verified by simulations on a full system, cycle level simulator executing industry standard benchmark programs. Rapid convergence is demonstrated despite the modeling errors and changing program loads.

The first application of the proposed approach was done in [32] for controlling the dynamic core power via DVFS. The problem considered here is more challenging for the following two reasons.

1. The underlying model required for this work is much more complicated. The authors in [32] considered static power as a constant which allowed them to use an established third-order polynomial formula for the dynamic power as a function of frequency. In contrast, the temperature's dependence on frequency has no explicit formula, but rather is described implicitly by a differential equation that models the heat flow. Furthermore, the temperature depends on the total power (sum of static and dynamic) while the static power depends on the temperature (and voltage). This nonlinear dependence was avoided in [32] by ignoring the static power.[2] For reasons discussed later, the duration of the control cycle is about 10ms, which requires fast computations in the loop. The main challenge in this regard is to find an approximate model yielding simple computations while preserving the aforementioned convergence properties of the control algorithm.

2. The temperature levels in different cores on a chip are inter-related due to the heat transfer between them, while their dissipated dynamic powers are not directly related to each other by such physical laws. Therefore it is natural for the dynamic-power

[2]In present-day technologies and applications the static power can be as high as the dynamic power and can no-longer be ignored.

control law in [32] to be distributed among the cores, while in this work the temperature control appears to have to be centralized. Nonetheless a distributed control law is argued for and its use is justified via analysis and simulation.

6.2 Regulation Technique

Consider the discrete-time, Single-Input-Single-Output (SISO) feedback system shown in Figure 6.1, whose input is a constant reference r, its output is denoted by y_n, the input to its controller is the error signal e_n, and the input to the plant is $u_n \in \mathbb{R}$. Suppose that the plant is a time-varying nonlinear system described via the relation

$$y_n = g_n(u_{n-1}), \tag{6.1}$$

where the function $g_n : \mathbb{R} \to \mathbb{R}$ is called the *plant function*.

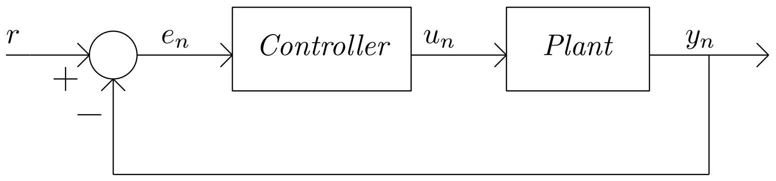


Figure 6.1: Control System Block Diagram

If the controller is an integrator having the transfer function $G_c(z) = Az^{-1}/(1 - z^{-1})$, for a constant $A > 0$, then in the time domain it is defined by the relation $u_n = u_{n-1} + Ae_{n-1}$. However, an adjustable (controlled) gain is considered, and hence the controller equation has the form

$$u_n = u_{n-1} + A_n e_{n-1}, \tag{6.2}$$

where the gain A_n is computed in a manner described below. The error signal has the form

$$e_n = r - y_n. \tag{6.3}$$

Suppose that the plant functions $g_n(u)$ are differentiable, and let "prime" denote their derivatives with respect to u. The gain A_n is defined as

$$A_n = \frac{1}{g_n'(u_{n-1})}. \tag{6.4}$$

The systems considered in the sequel have the following structure. Consider a SISO dynamical system having an input $\{u(t)\}$ and output $\{y(t)\}$, $t \geq 0$. Partition the time-horizon $\{t \geq 0\}$ into consecutive time-slots $[\tau_{n-1}, \tau_n)$, $n = 1, 2, \ldots$, with $\tau_0 := 0$ and $\tau_{n+1} > \tau_n \; \forall \; n = 1, \ldots$; define $C_n := [\tau_{n-1}, \tau_n)$ and call it the n^{th} control cycle. Suppose that the value of the input is changed only at the boundary points τ_n, and denote the value of the input $u(t)$ during C_n by u_{n-1}. Let y_n be a quantity of interest that is generated by the system during C_n from u_{n-1}, such as $y(\tau_n^-)$ or $\int_{C_n} y(t)dt$. y_n also depends on the initial condition $y(\tau_{n-1})$, but this is reflected in Equation 6.1 by the system's definition as time varying. Thus, 6.1 represents certain input-output properties of dynamical systems while hiding the details of the dynamics and appearing to have the form of a memoryless nonlinearity. Regarding the feedback system, it is assumed that u_{n-1}, y_{n-1}, and e_{n-1} are available to it at time τ_{n-1}, and it generates y_n by 6.1 and computes A_n during C_n via 6.4. The closed-loop system is defined by repeated applications of Equations $6.1 \rightarrow 6.4 \rightarrow 6.2 \rightarrow 6.3$.

To see the rationale behind the definition of the gain A_n in 6.4 consider the case where the plant is time invariant, namely $g_n(u) = g(u)$ for a function $g : \mathbb{R} \rightarrow \mathbb{R}$. Then this control law amounts to a realization of the Newton-Raphson method for solving the equation $g(u) = r$, whose convergence means that $\lim_{n \to \infty} e_n = 0$. Furthermore, if the derivative $g'(u_{n-1})$ cannot be computed exactly, convergence also is ensured under broad assumptions. For instance, suppose that Equation (6.4) is replaced by

$$A_n = \frac{1}{g'(u_{n-1}) + \xi_{n-1}}, \tag{6.5}$$

where the error term ξ_{n-1} is due to modeling uncertainties, noise, or computational errors.

If the function $g(u)$ is globally monotone increasing or monotone decreasing, and convex or concave throughout $\mathbb{R}$, and if the relative error term $|\xi_n|/|g'(u_n)|$ is upper-bounded by a constant $\alpha \in (0,1)$ for all $n = 1, 2, \ldots$, then convergence (in the sense that $\lim_{n\to\infty} e_n = 0$) is guaranteed for every starting point e_0 as long as $g^{-1}(r) \neq \emptyset$. If $g(u)$ is piecewise monotone and piecewise convex/concave then convergence is guaranteed for a local domain of attraction; namely, for every point $\hat{u} \in \mathbb{R}$ such that $g(\hat{u}) = r$ and $g'(\hat{u}) \neq 0$, there exists an open interval I containing $\hat{u}$ such that, for every $u_0 \in I$, $u_n \to \hat{u}$ and hence $e_n \to 0$ as $n \to \infty$. More specifically, there exists $\gamma \in (0,1)$ and $N \geq 0$ such that, for every $n \geq N$,

$$|e_n| \leq \gamma |e_{n-1}|. \tag{6.6}$$

These, and more extensive results concerning convergence of Newton-Raphson method for finding the zeros of a function can be found in [154].

In the general time-varying case where the plant function g_n is n-dependent (as in 6.1), it cannot be expected to have $e_n \to 0$. However, the term $\limsup_{n\to\infty} |e_n|$ has been shown to be bounded by quantified measures of the system's time-variability. For instance, [32] derived the following result under conditions of monotonicity and strict convexity of the functions g_n: For every $\varepsilon > 0$ there exist $\delta > 0$ such that, if $|g_{n-1}(u_{n-1}) - g_n(u_{n-1})| < \delta$ $\forall n = 1, 2, \ldots$, then $\limsup_{n\to\infty} |e_n| < \varepsilon$. Moreover, there exist $\eta > 0$ and $N \geq 0$ such that, for every $n \geq N$, Equation 6.6 holds true as long as $|e_{n-1}| > \eta$.

These results have had extensions to the multivariable case arising in Multi-Input-Multi-Output (MIMO) systems with the same number of outputs as inputs (e.g., [154, 155]). Accordingly, for a given $M \geq 1$, let $u \in \mathbb{R}^M$ and $y \in \mathbb{R}^M$ denote the input and output of the plant, respectively. Define the plant function by Equation 6.1 except that g_n is a function from $\mathbb{R}^M$ to $\mathbb{R}^M$, the feedback equation by 6.2 except that A_n is an $M \times M$ matrix, the error term via Equation 6.3, and the gain matrix A_n by the following extension of Equation

6.4,

$$A_n = \Big(\frac{\partial g_n}{\partial u}(u_{n-1})\Big)^{-1}. \tag{6.7}$$

In the time-invariant case where $g := g_n$ is independent of n, the system consisting of repetitive applications of Equations $6.1 \rightarrow 6.7 \rightarrow 6.2 \rightarrow 6.3$ comprises an implementation of Newton-Raphson method for solving the equation $g(u) = r$.

This work is concerned with the time-varying case where the plant function depends on n as in 6.1, and the Jacobian matrix $\frac{\partial g_n}{\partial u}(u_{n-1})$ is approximated rather than computed exactly. In this case Equation 6.7 is replaced by the following extension of 6.5,

$$A_n = \Big(\frac{\partial g_n}{\partial u}(u_{n-1}) + \xi_{n-1}\Big)^{-1}, \tag{6.8}$$

where the error term ξ_{n-1} is an $M \times M$ matrix. Define the relative error at the n^{th} step of the control algorithm by $\mathcal{E}_n := ||\xi_{n-1}||\big(||\frac{\partial g_n}{\partial u}(u_{n-1})||\big)^{-1}$. Various general results concerning the Newton-Raphson method guarantee local convergence of the control algorithm under the condition that $\mathcal{E}_n \leq \alpha$ for some $\alpha < 1$, for all $n = 1, 2, \ldots$; see, e.g., [154]. They typically state that $\lim_{n\rightarrow\infty} e_n = 0$ in the time-invariant case, and show upper bounds on $\limsup_{n\rightarrow\infty} ||e_n||$ in the case of time-varying systems.

The control law defined by Equations 6.8 and 6.2 updates all of the M components of u_n simultaneously and hence can be viewed as centralized. However, by ignoring the off-diagonal terms of $\frac{\partial g_n}{\partial u}(u_{n-1})$ a distributed controller is effectively obtained. Formally, define D_n to be the matrix comprised of the diagonal elements of $\frac{\partial g_n}{\partial u}(u_{n-1})$, and define $\xi_{n-1} := D_n - \frac{\partial g_n}{\partial u}(u_{n-1})$. Then Equation 6.8 can be computed in parallel by Equation 6.5 for each input-output coordinate. Thus the system comprised of repeated applications of Equations $6.1 \rightarrow 6.8 \rightarrow 6.2 \rightarrow 6.3$ can be viewed as a distributed system consisting of repeated runs of $6.1 \rightarrow 6.5 \rightarrow 6.2 \rightarrow 6.3$.

6.3 Temperature Control in Multi-Core Processors

This section describes an application of the control technique described in Section 6.2 to temperature regulation in computer cores by adjusting their frequencies. Unlike the case of regulating the dynamic power, described in [32], the frequency-to-temperature relationships are highly dynamic and complex, and moreover, the temperatures at various cores on a chip are inter-related. Nevertheless the objective here is to have a distributed controller whose required calculations are as simple as possible since, among other reasons, their complexity poses a lower bound on the duration of the control cycles.

To this end, approximations are considered that trade off precision with low computational complexity by leveraging the convergence robustness reflected in Equations 6.5 and 6.6. Therefore much of the developments in this section concern modeling approximations that yield simple computations. The resultant control law is tested in the next section.

The first part of the investigation concerns the frequency-to-temperature relations in a single core, formalized via the scalar-version of Equation 6.1. Suppose that the frequency applied to the core has a constant value during each control cycle and it is changed only at the cycle boundaries. Let ϕ denote the frequency applied to the core during a typical control cycle, and let $P := P(t)$ and $T := T(t)$ denote the resulting dissipated power and spatial average temperature during the cycle. The power has two main components: static power and dynamic power, respectively denoted by P_s and P_d. The static power is dissipated due to leakage currents in the transistors, and the dynamic power is dissipated when the transistors are switched between the *on* and *off* states. Figure 6.2 depicts the functional relations between these quantities, and it is to be noted that the dynamic power depends on the frequency, the temperature depends on the total power, and the static power depends on the frequency and temperature. The relationships between these quantities are indicated in the figure by the system-notation S_1, S_2, and S_3, and their models are described next in detail.

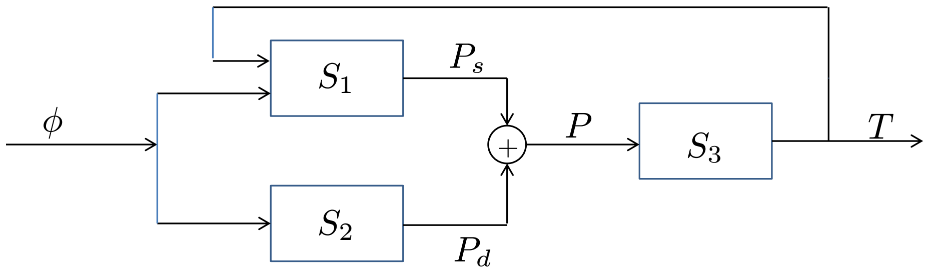


Figure 6.2: System Model

The core frequency is typically controlled by an applied voltage V, not shown in Fig. 6.2. The relationship between frequency and voltage can be modeled by the affine equation

$$V = m\phi + V_0, \tag{6.9}$$

[156, 157] whose slope m often can be obtained from the manufacturer.

As mentioned earlier, the total power is given by

$$P \;=\; P_s + P_d. \tag{6.10}$$

The system S_1 (Figure 6.2): An established physical model for the static power is described in [158], and it is given by the equation

$$\begin{aligned} P_s \;=\; & VNk_{\text{design}}I'_{so}e^{-(V_{\text{off}})q/(\eta kT)} \\ & \times 10^{-(V_T)q/(2.303\eta kT)}, \end{aligned} \tag{6.11}$$

where V is the applied voltage, N is the number of transistors in the core, k_{design} is a positive valued parameter depending on the core design, I'_{so} is a constant related to the subthreshold drain current, V_{off} is an empirically determined model parameter, $q = 1.6 \times 10^{-19}$C is the electron's charge, η is a technology-dependent parameter, $k = 1.38 \times 10^{-23}m^2kgs^{-2}K^{-1}$ is the Bolzmann's constant, T is the core temperature in Kelvin, and

V_T is the threshold voltage of the transistor. Grouping terms and defining

$$\beta = Nk_{\text{design}} I'_{so}$$

and

$$\gamma = q(V_{\text{off}} + V_T)/(2.303\eta k),$$

the following equation is obtained

$$P_s \;=\; V\beta \times 10^{-\gamma/T}, \tag{6.12}$$

where note that $\beta > 0$ and $\gamma > 0.$ Observe that P_s depends on V (and hence on ϕ via (9)) as well as on T.

The system S_2: An established model for the dynamic power [159] is described by the following equation,

$$P_d \;=\; \alpha(t)CV^2\phi, \tag{6.13}$$

where C is the lumped capacitance of the core, and $\alpha(t)$, called the *activity factor*, is a time-varying parameter related to the amount of switching activity of the logic gates at the core. Note that $\alpha(t)$ cannot be effectively computed or predicted in real time, but its evaluation is not needed for the control algorithm.

The system S_3: A detailed physical model for the power-to-temperature relationship is quite complex. However, the analysis is greatly simplified by treating the derivative term $\frac{dT}{dP}$ as approximately a constant that can be computed offline. In making this approximation, this work leverages the robustness of the tracking algorithm with respect to errors in the computation of $g'_n(u_{n-1})$ (see 6.5,6.6), as discussed in Section 6.2.

The power-to-temperature relationship in a core has had an effective model in [160], that is based on a linear and time-invariant system, and hence yields fast simulation-response as compared to physics-based models. The dimension of the system is the number of functional units in the core, typically in the 50 - 100 range. If the input u represents the vector of the dissipated power at each functional unit, and the state variable x is the temperature at each functional unit, then the state equation may be written as

$$\dot{x} \; = \; Ax + Bu, \tag{6.14}$$

where the matrices A and B can be estimated off line. At each time t, the total dissipated power at the core, $P := P(t)$, and the spatially averaged core temperature, $T := T(t)$, are a linear combinations of u and x, respectively. Therefore the $P - T$ relationship can be described via the scalar differential equation

$$\dot{T} = aT + bP. \tag{6.15}$$

Consequently, the derivative term $\frac{dT}{dP}$ satisfies the equation

$$\frac{d}{dt}\Big(\frac{dT}{dP}\Big) = a\Big(\frac{dT}{dP}\Big) + b. \tag{6.16}$$

The constants a and b can be estimated off line via simulation and used to solve the latter equation. Moreover, if the settling time of this equation is shorter than the control cycles then the steady-state value of Equation 6.16 can be used, which is $-\frac{b}{a}$. This additional approximation simplifies the control algorithm without significantly degrading its tracking performance. Details of the computation of this term will be presented in the next section, where its effectiveness in temperature control will be demonstrated.

Using the above models for the systems S_1, S_2, and S_3, the derivative term $\frac{dT}{d\phi}$ can be approximated that is required by the regulation law via Equation 6.5. In fact, combining Equations (6.9), (6.10), (6.12), and (6.13), and taking derivatives, after some algebra, the following equation is obtained:

$$\frac{dT}{d\phi} = \frac{\left(\frac{dT}{dP}\right)\left(m\frac{P_s}{V} + (P - P_s)\left(\frac{1}{\phi} + \frac{2m}{V}\right)\right)}{1 - \left(\frac{dT}{dP}\right)P_s(\log 10)\left(\frac{-\gamma}{T^2}\right)}. \tag{6.17}$$

An important point to be noted here is that all of the terms in the RHS of this equation except for P_s and $\frac{dT}{dP}$ can be obtained from real-time measurements of a core, P_s can be calculated online using Equation (6.12), and $\frac{dT}{dP}$ can be estimated off-line by its steady-state value, $-\frac{b}{a}$, obtained from 6.16.

Consider now the case of multiple cores on a chip, where the problem is to regulate their temperatures to given (not-necessary identical) setpoints by adjusting their respective frequencies. Due to the thermal gradients between the cores, it appears that their temperatures have to be regulated jointly. However, extensive simulations, described in the next section, reveal that the Jacobian matrix of the function relating the cores' frequency vector to the temperature vector is diagonally dominant and this justifies the use of a distributed control where each core runs an adjustable-gain integrator as described in Section 6.2. The details of this control law will be presented in the next section.

6.4 Results

The proposed controller is tested on Manifold [161], a cycle-level, full-system processor simulation environment with a suitable interface for injecting the thermal controller. The Manifold framework simulates the architecture-level execution of applications based on state-of-the-art physical models [162]. A functional emulator front-end [163] boots a Linux kernel and executes compiled binaries from an established suite of benchmarks [118].

The processor that is simulated consists of four out-of-order execution cores, a two-level cache hierarchy, and a memory controller, and its architecture is shown in Figure 6.3. The centralized (joint) control consists of repeated applications of Equations $6.1 \rightarrow 6.8 \rightarrow$

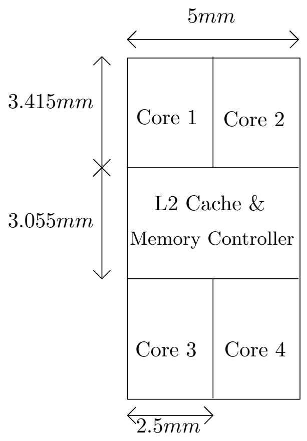


Figure 6.3: Floor Plan of the 4 Core Processor

6.2 $\rightarrow$ 6.3, where $u_{n-1} = \phi_{n-1} \in \mathbb{R}^4$ is the vector of core frequencies during the n^{th} cycle and $y_n = T_n \in \mathbb{R}^4$ is the vector of core temperatures at the end of the n^{th} cycle. Recall that Equation 6.8 denotes the controller's gain, and since it is diagonal, the control is implemented by the cores in a distributed fashion. In contrast Equation 6.1 represents the processor system and hence must be simulated jointly. This is done on Manifold in the following way.

Equation 6.1 can be written as $T_n = g_n(\phi_{n-1})$, where $\phi_{n-1} := (\phi_{n-1,1}, \ldots, \phi_{n-1,4})^\top \in \mathbb{R}^4$ and $T_n := (T_{n,1}, \ldots, T_{n,4})^\top \in \mathbb{R}^4$ according to their respective co-ordinates, with the second subscript $j = 1, \ldots, 4$ corresponding to the index of the core in Figure 6.3. In Equation 6.8 the 4×4 Jacobian matrix $\frac{dT_n}{d\phi_{n-1}}$ is approximated. Its diagonal terms, $\frac{\partial T_{n,j}}{\partial \phi_{n-1,j}}$, $j = 1, \ldots, 4$, are just the terms $\frac{dT}{d\phi}$ in the Left-Hand Side (LHS) of Equation 6.17 with the subscripts n, j indicating core j at the n^{th} control cycle. As mentioned earlier all the terms in the RHS of 6.17 can be obtained from real-time measurements and computation except for $\frac{dT}{dP}$, now referred to as $\frac{dT_{n,j}}{dP_{n-1,j}}$. For estimating this term Eqn. 6.16 is used in the steady state. To this end extensive cycle level simulations the processors are run in open loop with various input frequencies. Each simulation is run for successive cycles of 10ms, long enough for the temperature to reach its steady state, and it yields traces of power and its corresponding temperature at each cycle. The traces, providing over $4,000$ data pairs per core, indicate a nearly-affine power-to-temperature relation for each core regardless of the physical state (frequencies and temperatures) at the other three cores. The MATLAB Curve-Fitting Toolbox is used to approximate these power-temperature relations by respective curves, whose slopes serve to estimate the terms $\frac{\partial T_{n,j}}{\partial P_{n-1,j}}$. Since the P-T traces are generated across the entire spectrum of frequencies at all four cores, the slopes of the approximating curves do not depend on n, although they may depend on $j = 1, \ldots, 4$ according to the processor's floor plan. Thus, the steady-state solution of Equation 6.16 in the case of this work has the following approximation,

$$\frac{\partial T_{n,j}}{\partial P_{n-1,j}} \cong -\frac{b_j}{a_j}, \quad j = 1, \ldots, 4, \tag{6.18}$$

whose right-hand side is the slope of the curve associated with core j. The MATLAB Curve-Fitting Toolbox yields the following values, $3.97, 5.242, 3.877, 4.055$ for cores $1-4$, respectively, with an R-Square confidence metric > 0.97. As a further approximation, these four numbers are averaged and thus used $-\frac{b_j}{a_j} \cong 4.286$ for $j = 1, \ldots, 4$. This, in conjunction with 6.17 yields the terms $\frac{\partial T_{n,j}}{\partial \phi_{n-1,j}}$. It is to be noted that while this approximation of $\frac{\partial T_{n,j}}{\partial P_{n-1,j}}$ is independent of n or j, the partial derivative $\frac{\partial T_{n,j}}{\partial \phi_{n-1,j}}$ does depend on n and j through the other terms in the RHS of 6.17.

For the off-diagonal terms of $\frac{dT_n}{d\phi_{n-1}}$ it is observed (by the chain rule) that for $i, j = 1, \ldots, 4$,

$$\frac{\partial T_{n,i}}{\partial \phi_{n-1,j}} = \frac{\partial T_{n,i}}{\partial T_{n,j}} . \frac{\partial T_{n,j}}{\partial \phi_{n-1,j}}. \tag{6.19}$$

The second multiplicative term in the RHS of 6.19 was discussed in the previous paragraph. As for the first term, it is estimated by finite-difference approximations from the traces of simulation outputs. To this end, HotSpot, an established simulation platform designed to assess the thermal behavior of digital designs [164] is used. The thermal model generated by HotSpot consists of a linear, time-invariant circuit comprising resistors and capacitors, where potentials and currents represent temperature and power, respectively. The input to the circuit consists of current sources and the outputs are node voltages, and hence HotSpot is a suitable tool for modeling the thermal behavior of the core.

Varying the input power to the cores one-at-a-time, the temperature variations are obtained from which the finite-difference approximations for $\frac{\partial T_{n,i}}{\partial T_{n,j}}$ are derived. These approximating terms also are independent of n and hence denoted by $\frac{\partial T_i}{\partial T_j}$, but $\frac{\partial T_{n,j}}{\partial \phi_{n-1,j}}$ certainly depends on n through the second term in the RHS of 6.19.[3]

The matrix $\frac{\partial T_i}{\partial T_j}$, $i, j = 1, \dots, 4$, thus obtained from HotSpot, is

$$\frac{\partial T_i}{\partial T_j} = \begin{bmatrix} 1\times 10^6 & 0.0439 & 0.003378 & 0.003378 \\ 0.0439 & 1\times 10^6 & 0.003378 & 0.003378 \\ 0.003378 & 0.003378 & 1\times 10^6 & 0.0439 \\ 0.003378 & 0.003378 & 0.0439 & 1\times 10^6 \end{bmatrix} \times 10^{-6}.$$

This is clearly diagonally dominant, and hence the Jacobian matrix $\frac{dT_n}{d\phi_{n-1}}$ is expected to be diagonally dominant as well. This is indeed observed at each value of n, as the following randomly-chosen example from the Manifold runs shows,

$$\frac{dT_n}{d\phi_{n-1}} = \begin{bmatrix} 23800 & 1109 & 73.78 & 72.73 \\ 1045 & 25270 & 73.78 & 72.73 \\ 80.405 & 85.37 & 21870 & 945 \\ 80.405 & 85.37 & 958 & 21530 \end{bmatrix} \times 10^{-6}.$$

Therefore, the off-diagonal terms of the Jacobian matrix are neglected, thereby replacing the joint core-temperature control based on Equation 6.8 by four parallel one-dimensional controllers, one for each core, based on Equation 6.5.

The distributed controller is implemented in conjunction with processor timing simula-

[3]Manifold has the core frequencies as input but it does not permit varying the core powers one-at-a-time, while HotSpot allows us to do just that. This is the reason why both simulation environments are used in the manner described above.

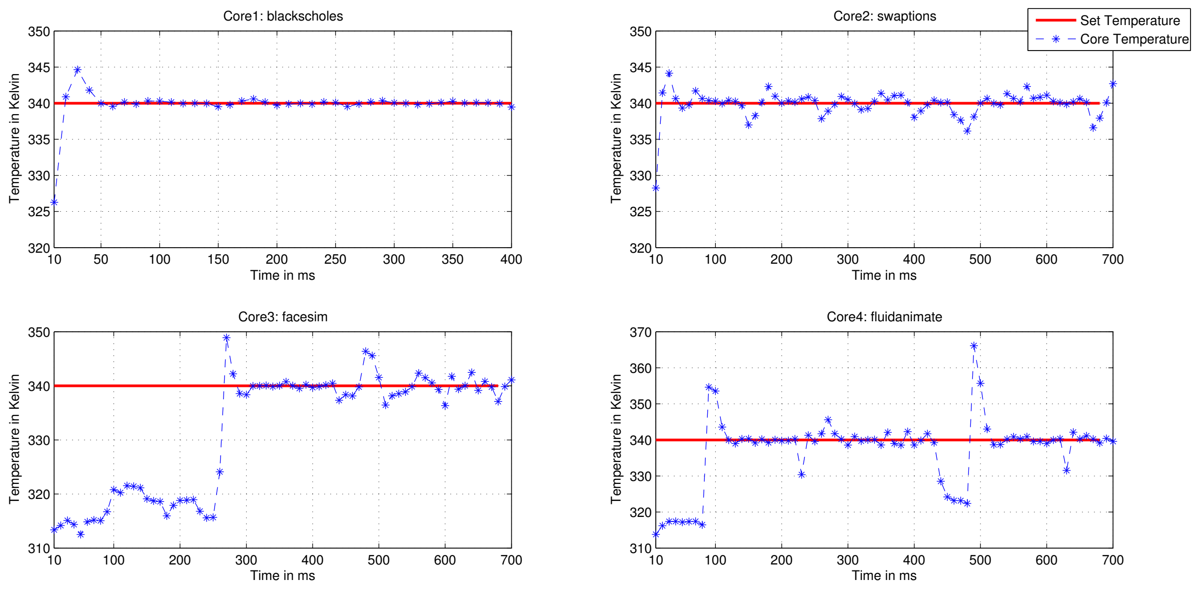


Figure 6.4: Tracking results with Continuous Frequencies

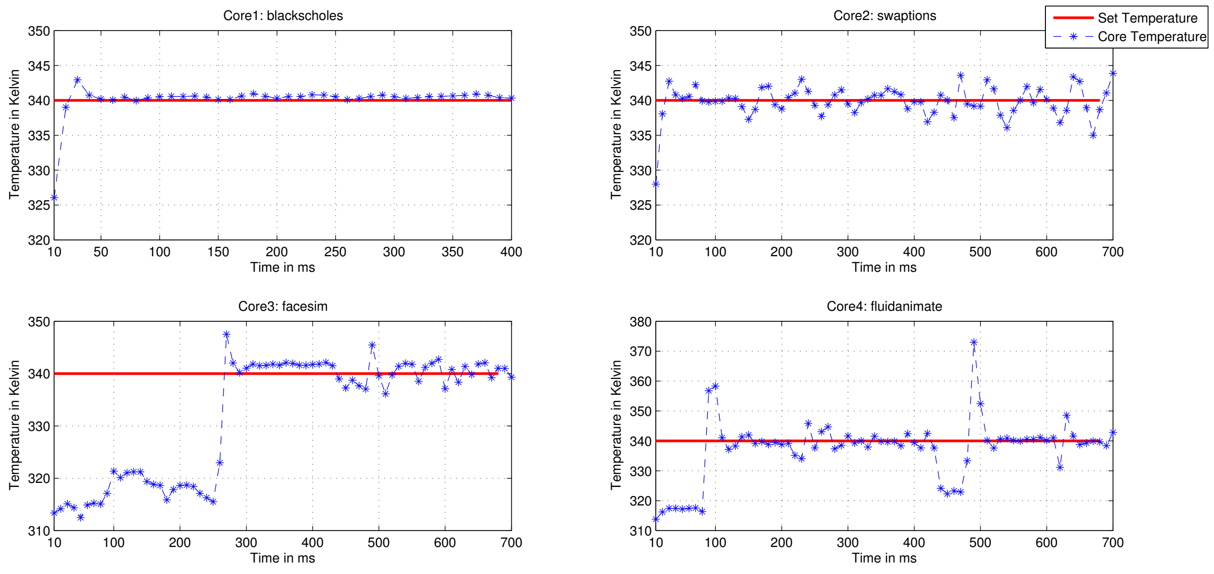


Figure 6.5: Tracking results with Discrete Frequencies

tion by Manifold. Each one of the cores executed a different benchmark program from the parsec suite of benchmarks [118]: `blackscholes`, `swaptions`, `facesim`, and `fluidanimate` are executed by Core 1, Core 2, Core 3, and Core 4 (see Figure 6.3), respectively. The target temperature of all cores is set to 340K, a typical value, and the range of frequencies is 1GHz to 4.7GHz. The control cycles at each one of the controllers are 10ms. `blackscholes` running on Core 1 lasts 400ms and hence the control is run for 40 cycles, while the rest of the benchmarks take longer than 700ms but the results are graphed only for the first 70 control cycles. The results are shown in the four graphs in Figure 6.4, and for each core the average temperature is computed from the end of the first overshoot to the cycle ending at the final time shown in the graph (400 ms for Core 1, 700 ms for the other cores).

In Core 1, convergence at 5 iterations (control cycles) is observed following a fast rise and a 5-degree overshoot. The average temperature (from the end of the first overshoot to iteration 40) is 339.995K. In Core 2 a similar rise and overshoot is seen as in Core 1, but then an oscillatory behavior and a not-so-smooth tracking is noted. The reason is that the benchmark `swaptions` has large and rapid variations in its activity factor $(\alpha(t))$ and hence in the dissipated dynamic power, causing ripples in the temperature profile. However, the computed average temperature is 339.96K - remarkably close to the target setpoint of 340K.

Core 3 shows no tracking until 250ms, then an overshoot followed by a 130-ms smooth tracking, and a period of minor ripples. The reason for the delayed tracking is that during the first 250ms the benchmark `facesim` is in a data-fetch (memory bound) phase when most of the computation units within the core are idle. Therefore there is no significant dynamic power dissipation and the core temperature does not rise. During that phase the core frequency first climbs to its maximum value (4.7Ghz) and then stays there until time 250ms[4]. Once the program enters the computation phase (time $>$ 250ms), the dynamic power rises which causes the core temperature to increase and the controller is now able to track the set temperature of 340K. The average temperature, computed as before, is 340.204K.

In Core 4 the benchmark program has two data-fetch periods and also periods of wide-

[4]This particular behavior is inefficient in terms of leakage energy and is highlighted in the next chapter.

range power dissipation during its execution. A similar delayed tracking is discerned as is observed with Core 3 but for a shorter duration, ending at $t = 80$ms. Later the program enters another data-fetch phase in the time range of 400 - 500ms, causing the core temperature to drop while the frequency rises to its maximum value. In both cases the data-fetch phase is followed by a computation phase which results in a temperature overshoot followed by a period of tracking except for ripples that are due to large variability in the dynamic power. The average temperature from the end of the first overshoot to the last control cycle shown in the graph is 339.565K.

In the previous simulation experiments, the frequency is allowed to take any value in the range 1GHz to 4.7GHz. However, in a typical processor only a finite set of frequencies can be applied to a core. Therefore, the simulation of the control technique is repeated for the following set of allowed frequencies, $\{1, 1.5, 1.8, 3.4, 3.7, 3.9, 4.0, 4.1, 4.2, 4.4, 4.7\}$ GHz. The only difference from the previous simulation is that in Equation 6.2 the control u_n is selected to be the nearest element in this set to the computed term $u_{n-1} + A_n e_{n-1}$. The results are shown in Figure 6.5, and they are similar to those in Figure 6.4 except that slightly larger ripples and minor steady-state errors are discerned. These are expected, and are due to the quantization errors in the selection of frequencies. However, the average temperatures at the cores, from the end of the first overshoot to the final time, are quite close to the setpoint reference: 340.482K, 339.986K, 340.623K, and 339.392K at Cores $1 - 4$, respectively.

This section is completed by comparing the tracking performance of the adaptive-gain controller with those using fixed gains. The need for an adaptive-gain control arises from unpredictable program activity factors $(\alpha(t))$, which may vary widely during the program. The same four-core system is simulated but the controllers are applied only to core 4 running the `fluidanimate` benchmark. The frequency range is continuous. A low gain of 10 and a high gain of 120 are chosen. The graphs of the temperature traces obtained from these two gains as well as the variable-gain control are shown in Figure 6.6. It is readily seen that the low gain results in the longest settling times, while the high gain yields larger oscillations. Not surprisingly, the tracking performance of the variable-gain controller is better than those of the two fixed-gain controls.

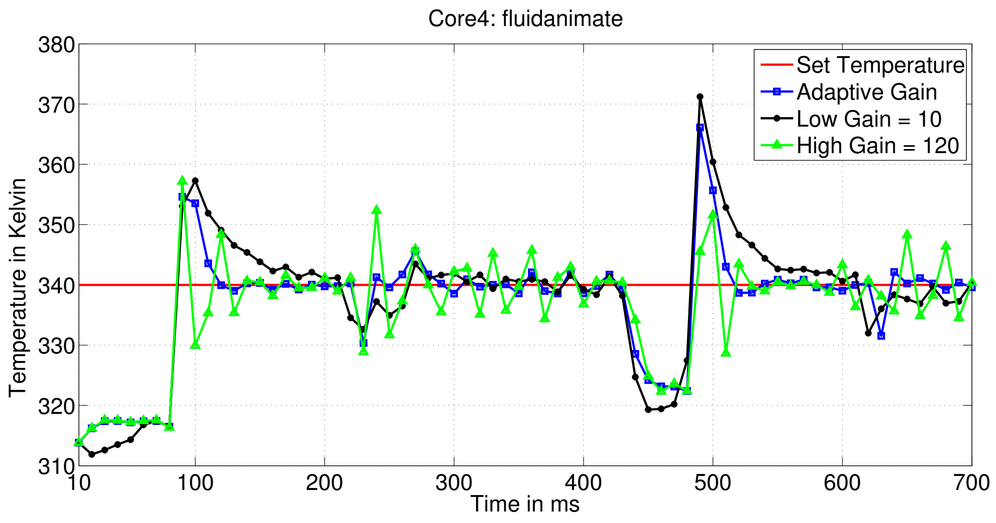


Figure 6.6: Tracking results with fixed gains and variable gains

6.5 Summary

Temperature regulation has emerged as a fundamental requirement of modern and future computing systems. The state of the practice to date has been dominated by ad-hoc adaptive heuristics. More recent attempts have begun to apply the rich landscape of control theory to this problem. However, these techniques have primarily dealt with temperature as a constraint while controlling power dissipation.

This work makes a subtle yet important observation - temperature ought to be directly regulated to track a target value while power should be managed to maximize performance. Regulating chip-wide temperature to a balanced thermal field is necessary while preventing transitions across a maximum temperature, since the latter can produce thermal fields that adversely affect reliability and performance. Furthermore, unlike prior works a nonlinear, time-varying plant model for a core is considered that explicitly captures the exponential dependence of temperature and static power, and a distributed control technique is devised that trades off precision with simplicity of real-time computations. Simulation results using a full system, cycle level simulator executing industry standard benchmark programs indicate convergence of the regulation technique despite the modeling approximations.

CHAPTER 7
CHARACTERIZATION OF A 3D PROCESSOR-MEMORY ARCHITECTURE

The performance of data intensive computing systems that process terabytes of data is increasingly limited by data movement and corresponding energy overheads. 3D packaging technologies enabled by advances such as, Through-Silicon-Via (TSV) technology [4], has led to stacking of silicon dice, thereby enabling the integration of memory and logic in a small footprint with significant reductions in data movement latency and energy. Stacking memory dice on top of compute dice exacerbates thermal issues, which if left unchecked, will preclude any performance gains from co-locating compute and memory. In particular, the exponential relationship between temperature and leakage current diminishes the performance that can be achieved for the rated heat capacity of the package. This limits the opportunity to exploit the order of magnitude increase in available memory bandwidth [165, 166].

The goal of this chapter is to understand the multi-physics interactions between temperature, application characteristics and the microarchitecture. Targeted microbenchmarks are run on a cycle-level simulator coupled with power and thermal calculations. An important concept called 'effective heat capacity' is introduced in this chapter. This is the heat generated beyond which further gains in performance are in-feasible with further increases in voltage-frequency of the compute logic. Among other results, this chapter makes the claim that temperature ought to be used as a resource like compute or memory cycles and not as a constraint to be met. Detailed microbenchmark characterization results are discussed and the chapter concludes with a summary of the insights which lead to the development of TRINITY.

7.1 Overview

3D packaging of silicon dice has been enabled by advances such as Through-Silicon-Via (TSV) technology [4]. Consequently, memory and logic can now be integrated into a single package. Reducing the physical distance between the components (i) reduces access latencies (ii) reduces energy per bit of data accessed and (iii) increases bandwidth by an order of magnitude. Compared to commercial standards like DDR3-1333 [5] and DDR4-2667 [6] whose bandwidths are 10.66 GB/s and 21.34 GB/s, respectively, 3D stacked memory technologies like HBM2 [7] and HMC2 [8] provide 256 GB/s and 320 GB/s, respectively. In order to effectively exploit the high bandwidth provided by 3D die stacked DRAM, multiple research efforts such as [9, 10, 11, 12, 13, 14, 15], are starting to explore moving compute logic inside the package as part of the die stack, revisiting the early efforts at architecting Processing-In-Memory (PIM) designs.

The compute logic layer in the 3D stack can range from simple atomic operations to multiple Out-of-Order (OoO) cores to general purpose low power GPUs. Stacking multiple dice on top of each other increases the thermal resistance between the bottom-most layer and the heat sink. To make matters even worse, silicon is not a good conductor of heat. Each layer now starts to behave as a 'thermal shield' for the layers below thereby causing higher leakage power and consequently higher temperature in the entire stack. At first look, the viability of co-locating compute and memory for better performance seems to be in jeopardy. The exponential dependence of leakage current and temperature can severely limit the potential of 3D stacked systems.

The early works in [167, 88] focus on architectural modifications such as placing the hottest layers closest to the heat sink. Placing the compute layer right below the heat sink allows efficient cooling. However, as noted in [22], manufacturing such a device can be prohibitively expensive. A typical processor has close to a thousand pins half of which are dedicated to voltage and ground signals. Therefore, placing the compute layer at the bottom of multiple stacked DRAM dice is practically viable but leads to worse thermal issues. Fortunately, the work in [165] claims that efficient cooling can alleviate thermal problems to a great extent.

Although these preliminary studies provide a general direction, they lack a detailed understanding of multi-physics interactions between microarchitectural parameters such as ops/byte and memory addressing patterns on temperature. Moving forward, designing and developing systems that can (i) exploit the large available memory bandwidth and (ii) optimally utilize the compute power without causing thermal violations, necessitates a fundamental understanding of the tight coupling between performance, energy and temperature and how this coupling is modulated by application workloads.

There is a rich body of work on managing thermal effects in processors, part of which has been discussed in the CHAPTERS 2 and 6, albeit in a 2D architecture. In a traditional 2D architecture where the core and the cache reside on the same die, the effectiveness of the heat sink in removing heat from the core blocks is much larger. This led some researchers to ignore thermal leakage between the core blocks entirely. Furthermore, leakage power was either considered a constant or negligible in comparison to the dynamic power leading to simplified power and thermal models.

Software-based efforts such as [20, 21, 81, 83, 84], typically seek to redistribute heat to avoid peak temperature violations. Hardware based efforts employ dynamic voltage frequency scaling (DVFS) to manage the thermal fields [82, 87, 80]. Detailed thermal modeling using software packages such as HotSpot [168] and 3D-ICE [169] enable the study of the role of microarchitectural designs on temperature variation. Although bulk of the work has been pursued for 2D packages, the understanding is still relevant to 3D packages. For example, researchers have explored the thermal coupling between cores on the same layer and between cores on different vertically stacked layers [170]. In general these approaches have dealt with temperature as a constraint. This thesis argues that temperature may be treated as a resource to be managed just like memory or compute cycles. This approach is rooted in a different view of the relationship between performance and heat capacity.

The heat capacity of the package is established based on the thermal design power (TDP) which is set independent of the application characteristics. However, some applications such as sparse matrix computations have components that are memory bound rather than compute bound. Temperature-based approaches to improve the performance of such applications by boosting voltage-frequency in an attempt to utilize the thermal headroom [51], will invariably increase power consumption with little or no performance gain and significant reductions in energy efficiency. On the other hand, compute intensive applications such as dense matrix algebra may extract performance benefits from DVFS schemes but can exceed the temperature bounds. Furthermore, thermal coupling between adjacent cores can increase leakage current (and therefore static power) and accelerate temperature rise leading to premature throttling [106] and, therefore, loss of performance and energy efficiency.

This thesis develops a strategy to ensure that for the amount of heat generated by the compute logic for an application, the maximum performance (throughput) is delivered. In the process, energy efficiency too is improved. A key insight is that, applications, and some application phases, simply cannot utilize the package thermal headroom even when operating at the highest voltage-frequency state. This thesis attempts to take advantage of the said observation by noting that for a specific application or phase there is an *effective heat capacity* (EHC) - this is the heat generated beyond which further gains in performance do not occur with further increases in voltage-frequency of the compute logic. For example, an application may be operating in a memory bound phase and increases in compute logic frequency has little effect on performance but may consume the thermal headroom. Accordingly, it is noted that the EHC is application-specific and time-varying. Consequently, the goal is to maximize the performance that can be extracted from the time-varying EHC. The solution must be online, adaptive, and robust to modeling errors. The EHC corresponds to a value of temperature which is referred to as the *effective maximum temperature*

(EMT). Practical implementations will seek to operate at the EMT and minimize thermal coupling induced leakage power.

This chapter first seeks to investigate the interactions between (a) thermal behaviors, (b) compute & memory microarchitecture, and (c) application workloads. In particular, the goal is to understand (i) the influence of workloads on thermal coupling effects between compute elements as well as between compute and memory elements (ii) thermal field variation due to memory addressing patterns, and (iii) how the critical performance relationship between DRAM and compute elements is affected by thermal effects.
The contributions of this chapter are as follows:

1. The introduction of the concept of effective heat capacity as a thermal resource to be managed.

2. A comprehensive simulation-based characterization of intra- and inter-die thermal coupling effects demonstrating the need to maximally utilize the effective heat capacity.

The insights obtained in this chapter form the basis for the controller, TRINITY, described in the next chapter.

7.2 Characterization

This section seeks to find answers to the following questions:
(1) What is the thermal impact of a hot core on neighboring cool cores? What are the performance implications for both the hot core and the cool cores?
(2) What is the thermal and performance behavior of a program thread executing at different physical locations on the core layer?
(3) How does memory addressing patterns in the L2 Cache layer affect the temperature of the core layer and vice-versa?

First, the details of the 3D stacked processor-memory framework is described. The characterization of temperature and performance under a variety of microbenchmark workloads is subsequently described in detail. Temperature is measured in Kelvin and performance in Million-Instructions-Per-Second (MIPS). The temperature numbers reported are steady state values. The microbenchmarks are designed such that they (i) exhibit variable ops/byte ratio, (ii) access specific memory locations, and (iii) execute on specific physical cores.

7.2.1 Experimental Framework

The physical layout is shown in Fig. 7.1 with the dimensions listed in Table 7.1 and Figure 7.2 represents the functional diagram of the 3D stacked architecture. The 3D stacked architecture consists of 16 Out-of-Order (OoO) cores with two levels of cache hierarchy [171], interfacing an HMC style [8] 4GB DRAM via an interconnection network. The simulator is also equipped with power estimation models based on McPat [172] and the thermal calculations are done using 3D-ICE [169], both scaled to 16nm. The front-end for the cycle level simulator is a multicore emulator called Qsim [163] that boots a Linux kernel and executes applications of interest. The x86 instruction streams thus generated are fed into the OoO core timing model. DRAMSim2 [144] is used as the DRAM timing simulator whose voltage and timing numbers are modified based on the work in [173].

7.2.2 Nomenclature

To better represent the characterization results, a naming convention is described in Figure 7.3 which is used throughout the characterization section. All the microbenchmarks are single threaded programs. Most of the results that follow have a single thread running on a single fixed core (source core) accessing data from a single fixed L2 Cache bank (source/remote bank). A distinction is made when two cores are running independent microbenchmark applications as and when required. While a microbenchmark is running on

Table 7.1: Simulation framework parameters. Technology node is 16nm.

Component	**Parameters and Values**
Processor	Out-of-Order, 6-stage pipeline, 4-wide issue/commit, $0.5-1.5$GHz
L1 Cache per core (16KB)	Private, 8-way, LRU replacement, 32 MSHRs, 64B lines, 1-cyc hit & lookup time
L2 Cache per bank (2MB)	16 banks in total, shared, 8-way, LRU replacement, 128 MSHRs, 64B lines, 24-cyc hit & lookup time
Network (1GHz)	4×4 torus ring, 6 port router, baseline x-y routing
Memory Controller	16 MCs in total, rank then bank round robin, close page, Addr-map- chan:row:bank:rank:col
DRAM config per vault	256MB, 1-channel, 4 ranks, 2 banks per rank, 64 bit bus @ 1600MHz
Heat Sink	Conventional heat sink, Heat transfer co-eff = $2.8 \times 10^{-8} W/\mu m^2 K$
Per-Layer	TOP LAYER = BEOL: 25μm SOURCE LAYER = SILICON: 10μm BOTTOM LAYER = SILICON: 25μm

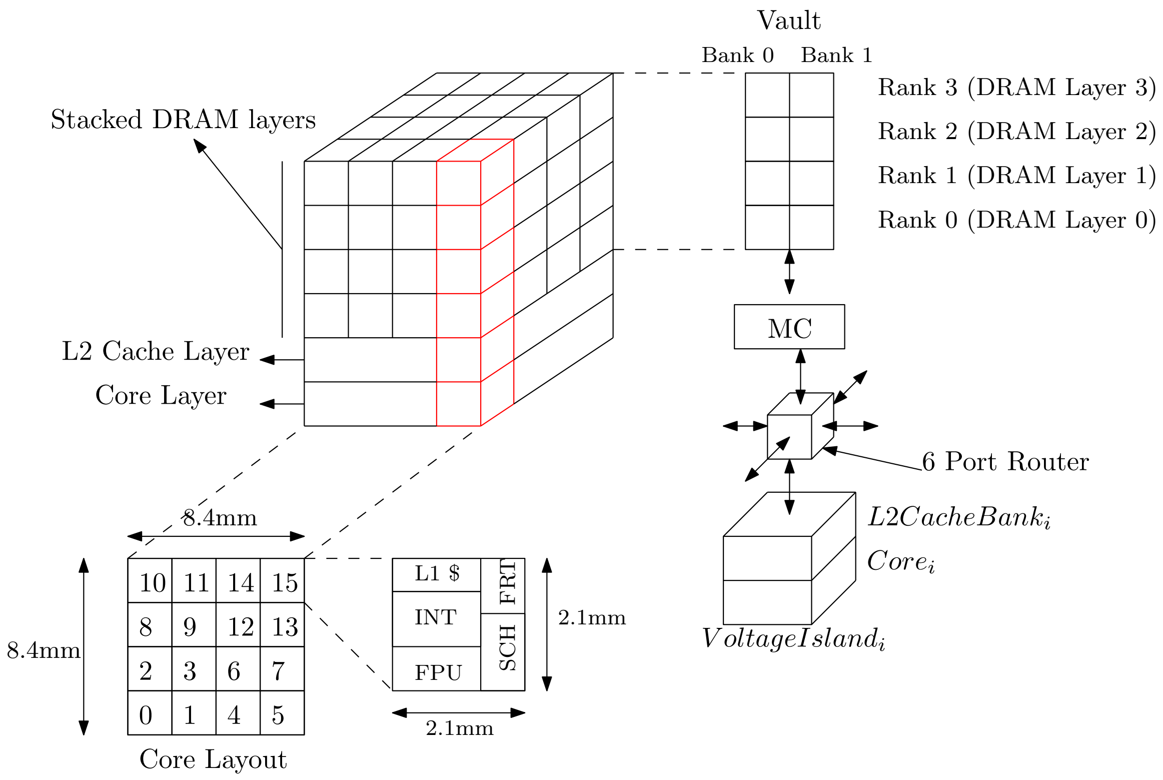


Figure 7.1: Physical layout of the 3D stacked architecture.

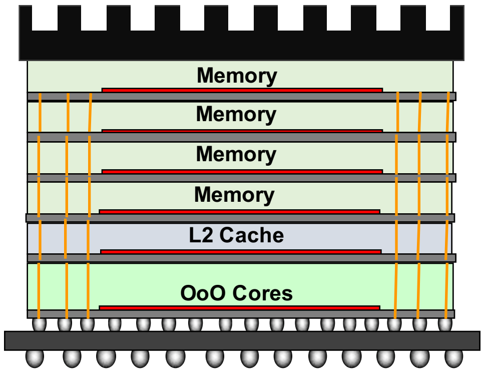


Figure 7.2: Functional description of the 3D stack.

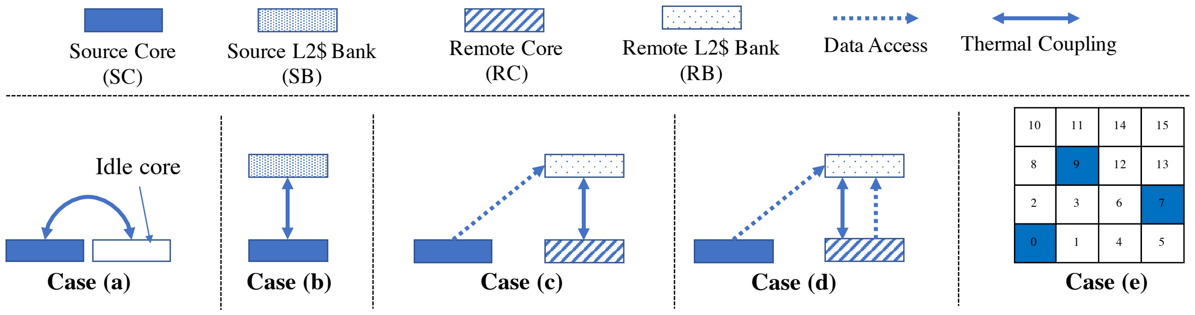


Figure 7.3: Microbenchmark characterization nomenclature.

a single core, the rest of the cores are powered up (V_{dd} and CLK are supplied) but idle. The 1-hop, 2-hop and diagonal neighbors of the source core are termed SC+1, SC+2 and SC+d, respectively. Similarly, for the L2 Cache banks there are SB+1, SB+2 and SB+d. In what follows, a “memory intensive benchmark” continuously performs `load` operations on sequential memory locations whereas a “compute intensive benchmark” repeats the following two steps: (i) `load` a block of data from memory (ii) perform integer and floating point operations for a fixed number of iterations.

Among the many cases of thermal coupling, 5 types are discussed in detail as shown in Figure 7.3:

(a) Thermal coupling between adjacent cores.

(b) Thermal coupling between a core and an L2 Cache bank directly on top.

(c) Thermal coupling between an L2 Cache bank and an idle core below it.

(d) Same as **(c)** but with a non-idle core.

(e) Thermal coupling variation when the computation is moved from the package boundary to the center of the die.

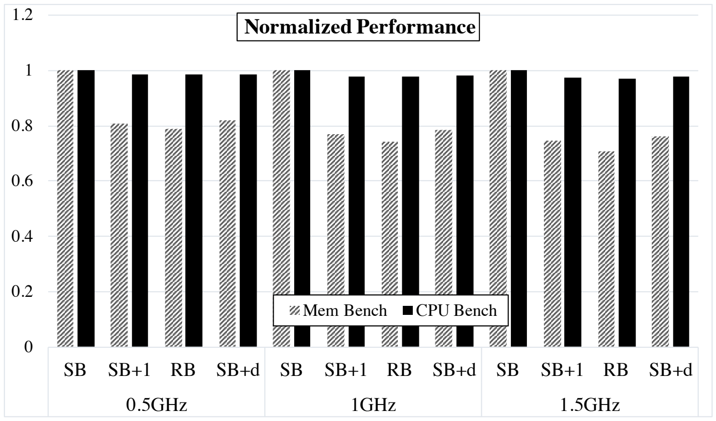
7.2.3 Thermal Coupling Analysis

Case (a): In Figure 7.4b, the temperature of the source core SC, average temperature of its 1-hop neighbors and 2-hop neighbors: SC+1, SC+2 and SC+d, respectively is plotted for a memory bound microbenchmark at three different clock frequencies with the SC accessing data from SB, SB+1, SB+d and RB. Figure 7.4c is similar to Figure 7.4b except that the microbenchmark is compute bound. It is noted that regardless of whether the benchmark running on SC is memory intensive or compute intensive, the steady state temperature of SC+1 which is idle, can go as high as 325 Kelvin due to thermal coupling. Thermal coupling effects are seen to be negligible beyond a 2-hop neighborhood. This is in concurrence with earlier works [75] (albeit [75] is for a 2D architecture). The extent of thermal coupling in a 3D architecture however, is most pronounced within the 1-hop neighborhood due to heat shielding from upper layers.

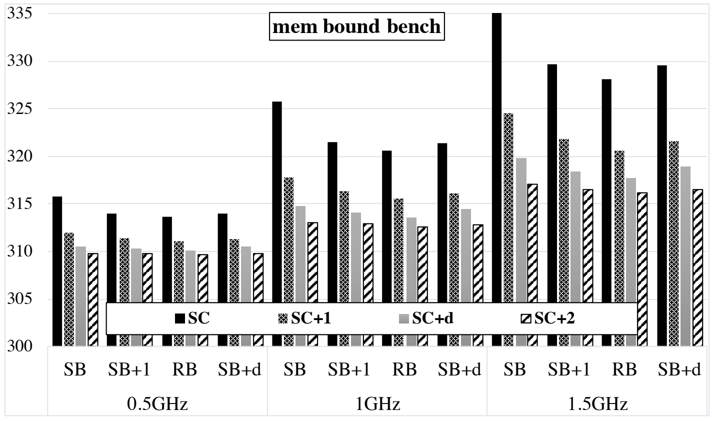
Figures 7.5 and 7.6 represent the same set of experiments as described in the previous paragraph, except that the SC is Core_2 and Core_3, respectively. While the trends in both the figures for performance and temperature are similar to the ones observed for Core_0, the magnitude of steady state temperature of the SC and its neighbors are considerably different. This aspect is analyzed carefully shortly. Nevertheless, the key observation to be made here is:

Observation 1: A 'hot' core reduces the EHC of neighboring 'cool' cores by up to 7 Kelvin.

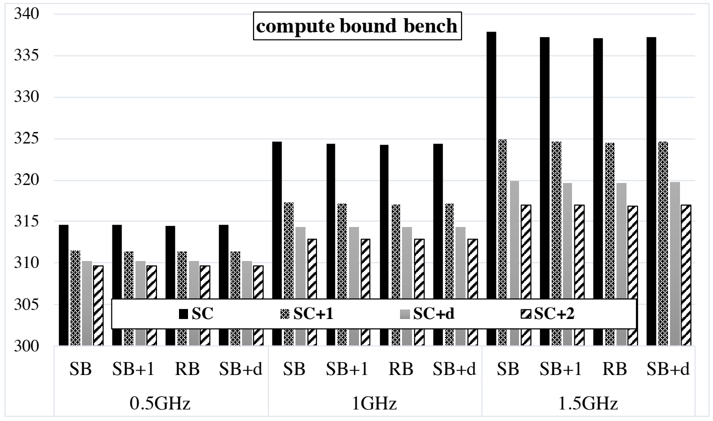
A second more subtle observation is obtained by analyzing the steady state temperature and performance of the SC when accessing SB and RB (For example, see Figs. 7.4a and 7.4b). By addressing a RB, the SC temperature can be reduced by up to 8 Kelvin. This however, comes at the price of 30% reduction in performance. Therefore,



(a) Performance on y-axis is normalized w.r.t SB.

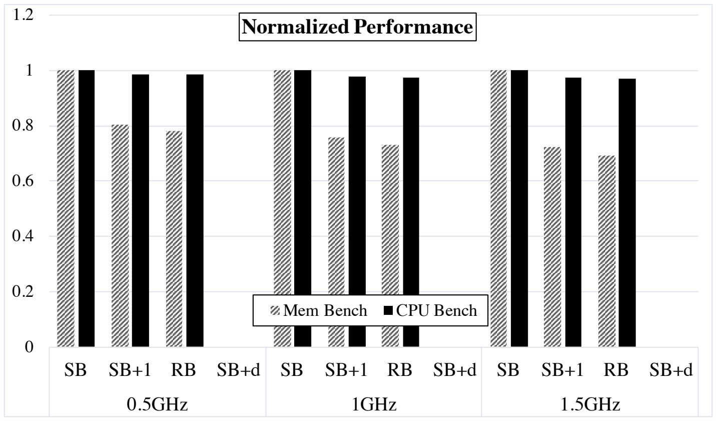


(b) Temperature of source core and its neighbors in Kelvin on y-axis.

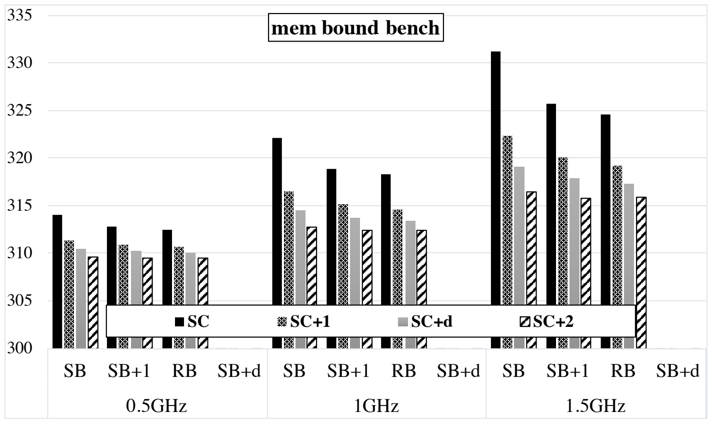


(c) Temperature of source core and its neighbors in Kelvin on y-axis.

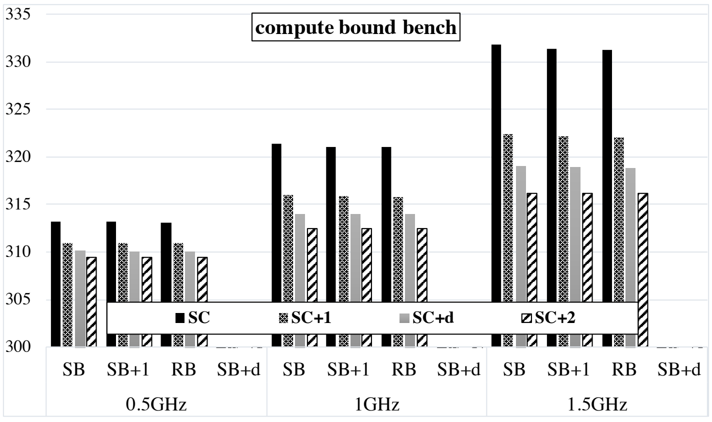
Figure 7.4: Performance and temperature variation when running mem bound and compute bound benchmarks on a source core accessing source and remote cache banks at different core frequencies.



(a) Performance on y-axis is normalized w.r.t SB.

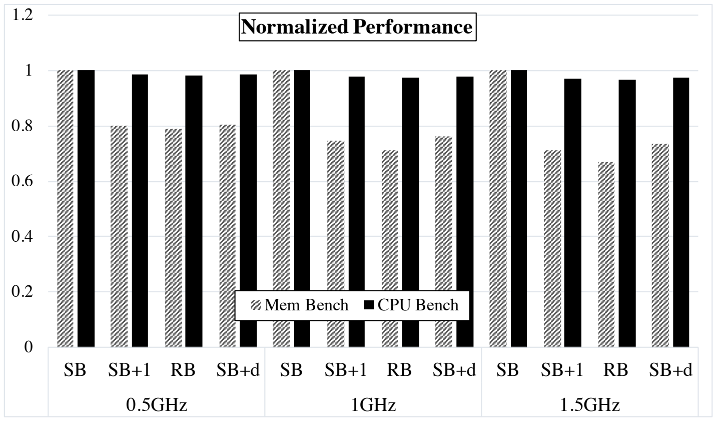


(b) Temperature of source core and its neighbors in Kelvin on y-axis.

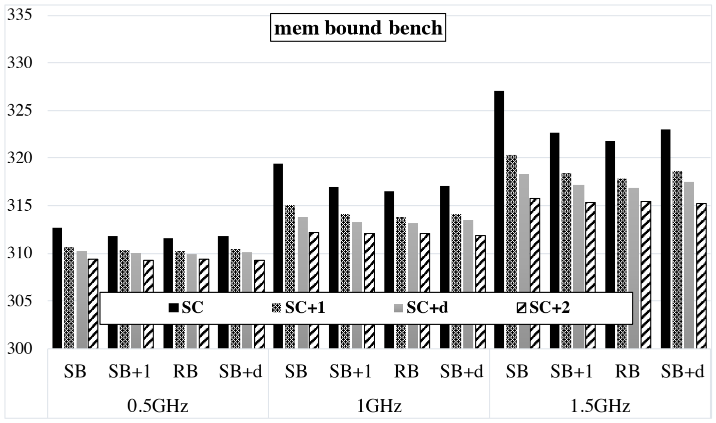


(c) Temperature of source core and its neighbors in Kelvin on y-axis.

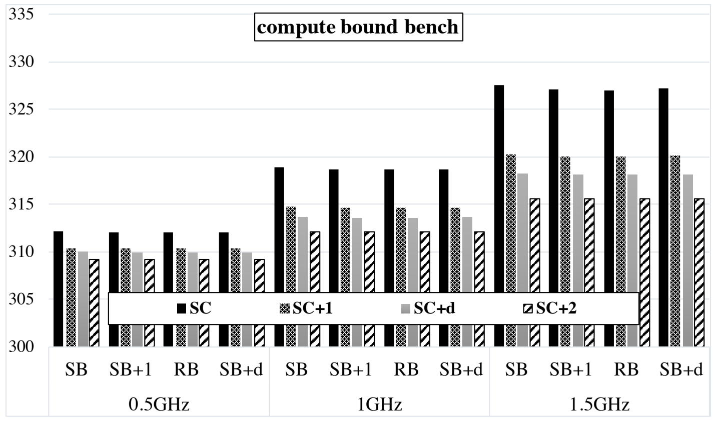
Figure 7.5: Performance and temperature variation when running mem bound and compute bound benchmarks on a source core accessing source and remote cache banks at different core frequencies. SC is Core$_2$.



(a) Performance on y-axis is normalized w.r.t SB.



(b) Temperature of source core and its neighbors in Kelvin on y-axis.



(c) Temperature of source core and its neighbors in Kelvin on y-axis.

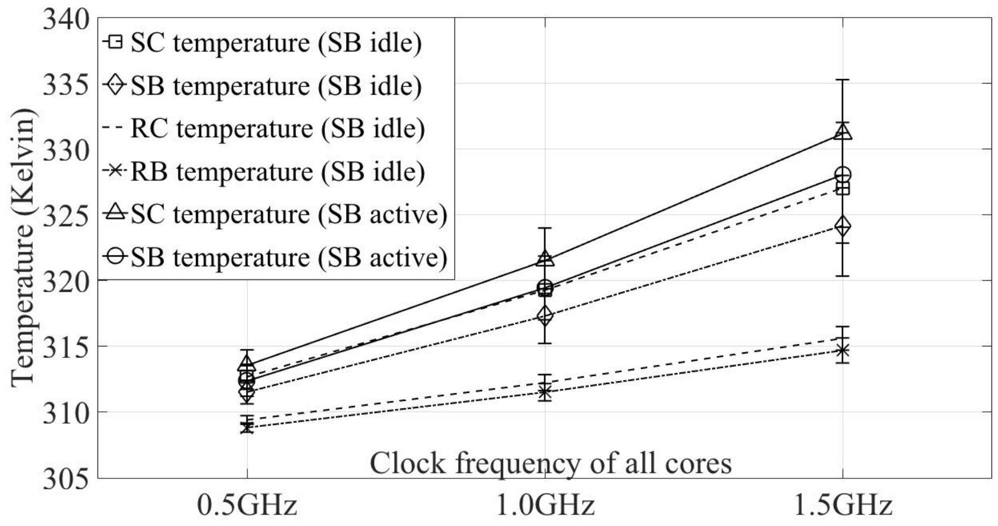
Figure 7.6: Performance and temperature variation when running mem bound and compute bound benchmarks on a source core accessing source and remote cache banks at different core frequencies. SC in Core$_3$.

Observation 2: *Memory address re-mapping has the potential to trade-off performance for reduction in temperature.*

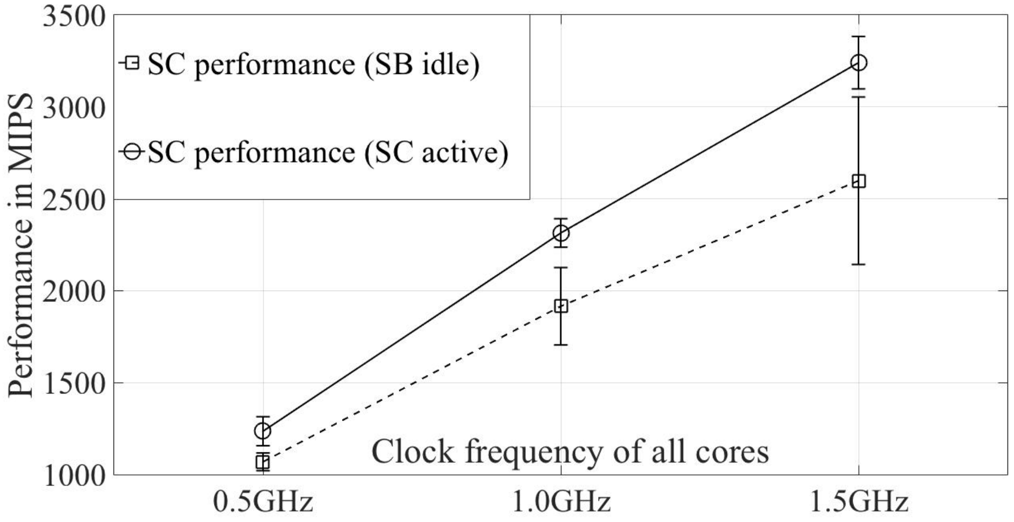
To completely understand the thermal coupling between the compute and memory layers, the inter-layer thermal coupling is divided into **Cases (b), (c)** and **(d)**. Figure 7.7 should be referred for **Cases (b)** and **(c)** and Figure 7.8 for **Case (d)**. Before proceeding to the analysis, it is essential to note that for the 3D architecture under consideration, in steady state, the core layer *always* has the highest temperature when compared to the upper layers.

Case (b): The heat flow between the SC and the SB is influenced by whether the SB is 'active' or 'idle'. The temperature trends for the SC and SB are presented in Figure 7.7a. When the SB is idle, the average SC temperatures are 312.7, 319.3 and 327.1 Kelvin at 0.5, 1.0 and 1.5GHz, respectively. But when the SB is active, the same SC temperatures increase by about 1, 2 and 4 Kelvin for 0.5, 1.0 and 1.5GHz, respectively. This clearly demonstrates the influence of memory addressing on the core layer temperatures. Not only does the average temperature rise with increase in frequency, but also the variance. At higher clock frequencies, thermal ramifications due to memory addressing patterns are more pronounced. The performance trend as seen if Fig. 7.7b is in accordance with expectations, in that, instruction throughput increases directly due to clock frequency increase.

Case (c): Moving along the same analysis path as before, for this case of thermal coupling, the aim is to understand the variations in temperature of an 'idle' core directly underneath an 'active' L2 Cache bank. The temperature plots of the remote core (RC) and remote cache bank (RB) in Figure 7.7a illustrate this situation. Analogous to the previous case, bulk of the power dissipated by the idle RC underneath the active RB is on account of static power. Furthermore, as clock frequency increases, idle RC temperature can increase up to 5 Kelvin higher than the lowest temperature on the core layer.



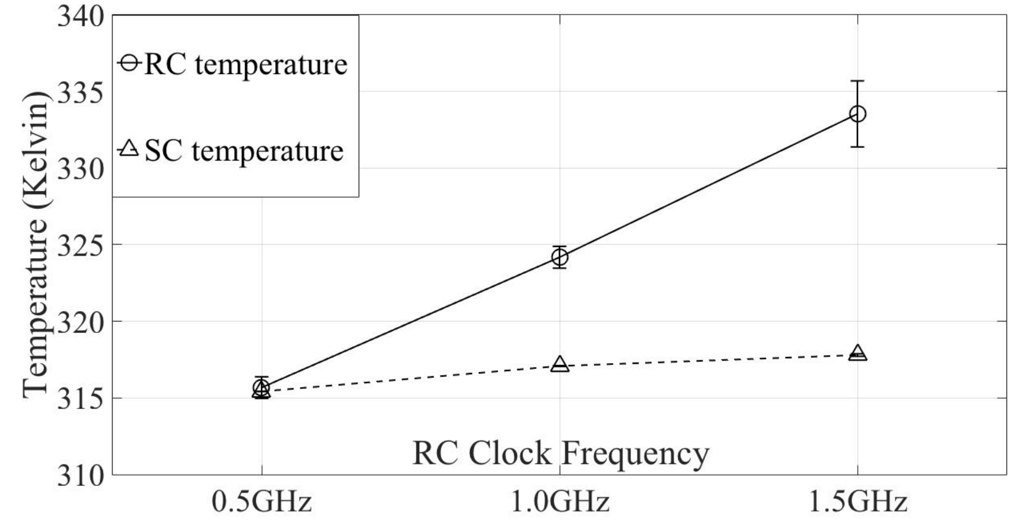
(a) Temperature variation of (i) source core and cache (ii) remote core and cache.



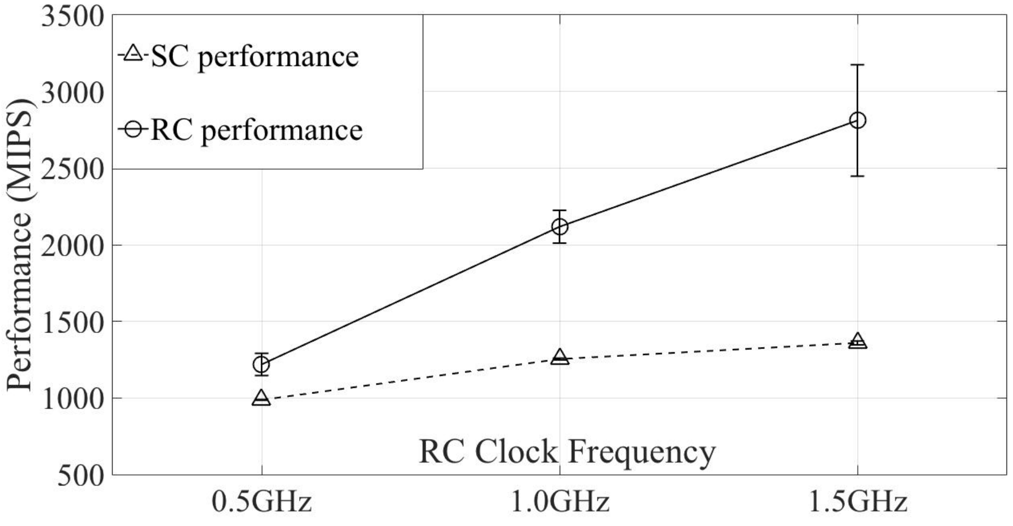
(b) Performance variation of the source core when source bank is ‘idle’ and ‘active’.

Figure 7.7: Thermal coupling Cases (b) and (c). The error bars are variances in temperature due to different ops/byte and physical locations of the source core.

Case (d): This case is essentially a superposition of **Cases (b)** and **(c)**. The experiments here attempt to replicate a scenario where multiple cores can access a single L2 Cache bank. As described in Fig. 7.3, both the SC and the RC access the RB. Since RC is not idle anymore, an increasing trend is observed in its temperature with clock frequency. The slope of this increase however, is slightly steeper when compared to SC temperature (SB active) in Figure 7.7a. Furthermore, the increase in the clock frequency of the {RC - RB} voltage island causes the performance of RC and SC to improve (See Fig. 7.8b). Due to difference in network delays however, slope of the performance curve for the SC is much smaller than that for the RC.



(a) Temperature variation of source core and remote core.



(b) Performance variation of source core and remote core.

Figure 7.8: Thermal coupling Case (d). The error bars are variances in temperature due to different ops/byte and physical locations of the source core.

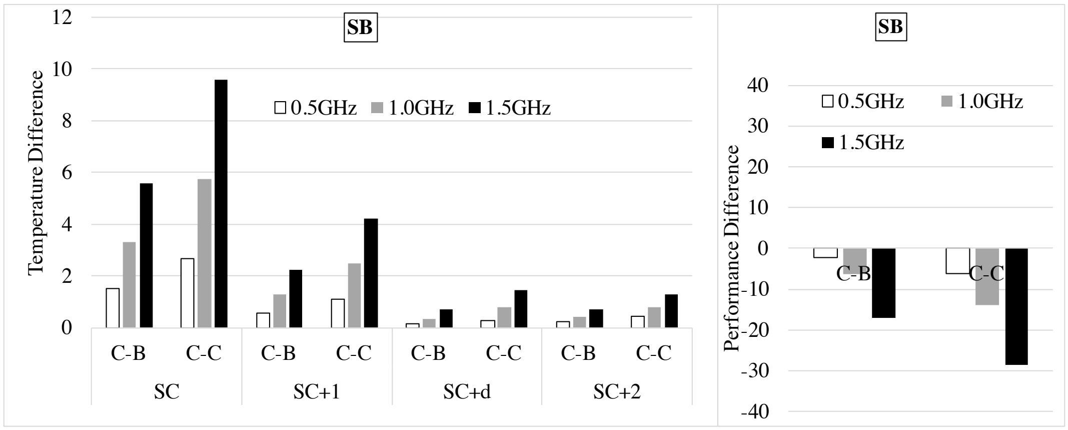
Case (e): Carrying forward from **Case (a)**, the same set of experiments are repeated as before but the microbenchmark is run on a SC that is physically located at three specific locations: (1) Corner (2) Boundary and (3) Center. Using 'Corner' as the reference, the differences in temperature and performance for the other two locations are calculated. Specifically, the differences are annotated as follows: Corner - Boundary (C-B) and Corner - Center (C-C). The trend of the data obtained is plotted in Figure 7.9. The difference between Fig. 7.9a and Fig. 7.9b is only with the memory location addressed, SB and RB, respectively.

Temperature difference in Kelvin and performance difference in MIPS are plotted on the y-axis. In general, moving the application thread from the corner to the boundary or center reduces the temperature of the SC between $1 - 10$ Kelvin with negligible loss in performance. The greatest difference is seen for the C-C case. Not only does the SC experience reduction in temperature, its neighbors SC+1 too benefit by up to 4 Kelvin due to the relocation. Note however, that this phenomenon does not nullify **Case (a)**. Only the magnitude of thermal coupling is mitigated to a small extent.

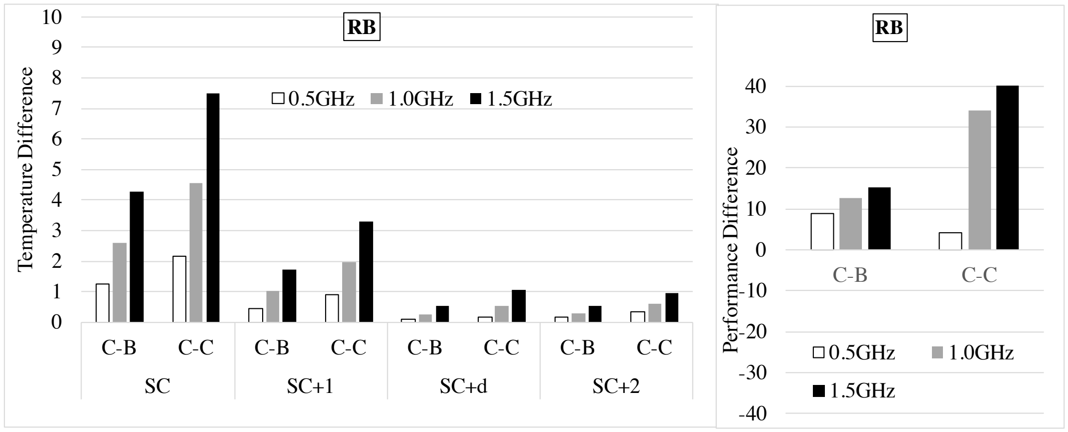
<u>*Observation 3*</u>*: Package boundaries become increasingly important in 3D stacked environments. OS level thread scheduling in cooperation with DVFS schemes can lead to better utilization of the EHC.*

7.3 Summary

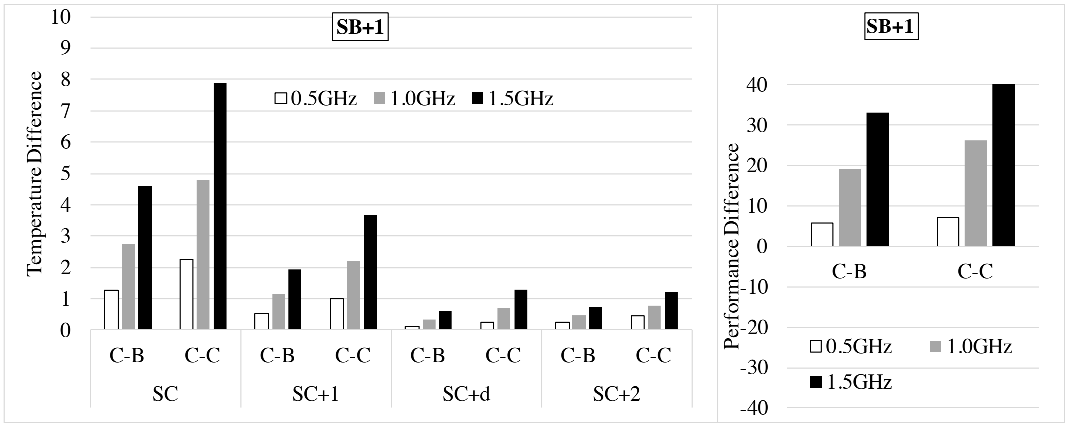
Microbenchmark characterization of the 3D stack sheds light on subtle yet key insights. The EHC of an application thread is affected not only by its own phases but also by memory addressing patterns of neighboring cores. A greedy approach to maximizing performance can indeed utilize the thermal headroom of the package but may not deliver the best energy efficiency (ops/Joule). Consequently, the higher temperatures, especially in thermally constrained environments such as the one under consideration, can increase thermal stresses and localized hotspots in turn reducing lifetime reliability of the device. Nevertheless, *max-*



(a) Source core accessing source bank. Performance and spatial temperature comparison of source core at Corner vs. Center vs. Boundary.



(b) Source core accessing remote bank. Performance and spatial temperature comparison of source core at Corner vs. Center vs. Boundary.



(c) Source core accessing 1-hop neighbor of source bank. Performance and spatial temperature comparison of source core at Corner vs. Center vs. Boundary.

Figure 7.9: Influence of package boundaries on thermal coupling and performance.

imizing performance in the face of unavoidable thermal coupling, necessitates a strategy that cooperatively balances performance, energy and temperature.

CHAPTER 8
COORDINATED MANAGEMENT IN 3D ARCHITECTURES: PERFORMANCE, ENERGY AND TEMPERATURE

This chapter presents an approach to the coordinated control of performance, energy and temperature on 3D processor-memory stacks. Using the concept of effective heat capacity introduced in CHAPTER 7, this chapter presents a technique called TRINITY to manage temperature as a resource. TRINITY is a DVFS controller that implements an on-line optimization technique that continuously balances performance, energy and thermal behaviors to fully utilize the effective heat capacity. Unlike prior research efforts which: (i) consider power, performance and temperature in isolation or in pairs, (ii) do not explicitly model static power, and (iii) are heuristics based, this work reveals the complex interplay between performance, energy, temperature, microarchitectural parameters and package physical constraints. Each voltage island implements an independently operated TRINITY controller which is: (i) based on numerical optimization, (ii) computationally inexpensive to implement, (iii) self-tuning, (iv) distributed (per-core), and (iv) application agnostic. The spatially adjacent controllers are implicitly coupled due to thermal effects. Thus, a network of interacting controllers seek to locally maximize throughput from the locally available effective heat capacity. Their coordinated actions indirectly makes the most efficient use the package heat capacity. The vehicle for exploration and demonstration is a cache-coherent multi-core processor integrated as the bottom die in a 3D DRAM stack. An analysis of energy efficiency, temperature and also lifetime reliability is presented. In cycle-level simulations of a 16 core architecture, for up to 11% increase in EDP, TRINITY keeps the temperature lower by up to 6 K as compared to a heuristics method similar to the `ondemand` Linux CPU governor. An added benefit of the reduced temperature is the increase in lifetime reliability of the 3D stack by up to 26%.

8.1 Overview

CHAPTERS 6 and 7 a few state-of-the-art DTM techniques were presented. CHAPTER 6 essentially extends the scope of adaptive gain integral feedback controllers from power and throughput regulation to a more complex problem of temperature regulation. Dynamically managing temperature in a 2D architecture is relatively easier when compared to a 3D architecture. The dimensionality of the system model is considerably larger and the coupling between temperature and performance is expected to be more complex. The characterization in CHAPTER 7 seeks to fundamentally understand the nature of this coupling. In conjunction with the detailed characterization of CHAPTER 7, few more examples are described here to substantiate the concept of effective heat capacity and the need to consider temperature as a resource.

Consider the regulator tracking results for `facesim` shown in Figure 6.4 of CHAPTER 6. For the first 250ms, although the core frequency saturates at the maximum value, the core temperature remains well below 340K. When the application is in a memory bound phase, due to the lack of activity on the core, it cannot use all of the package thermal capacity. Temperature of the core running `blackscholes` on the other hand, a compute intensive application, can easily reach the target temperature of 340K. The core running `facesim` dissipates energy in the form of heat for those 250ms of operation, without appreciable increase in performance.

Another related example is shown in [106]. On an AMD Accelerated Processing Unit (APU) that has a CPU and a GPU on the same die, the authors show that thermal coupling between the CPU and GPU can lead to premature throttling. While running a 100% CPU workload, the idle GPU temperatures reach levels close to the sustainable maximum. The same is observed when 100% GPU workloads are executed. These thermal *signatures* are application-specific *and* time varying. The authors subsequently demonstrate a heuristics based algorithm to minimize EDP.

These two examples clearly demonstrate that even in 2D architectures, temperature and performance are tightly coupled. Greedy performance maximization policies can and will lead to thermal throttling significantly reducing performance. Both examples indicate the existence of an Effective Maximum Temperature (defined in Section 7.1). The next set of motivational examples show that dynamically managing temperature is even harder in 3D die stacked architectures.

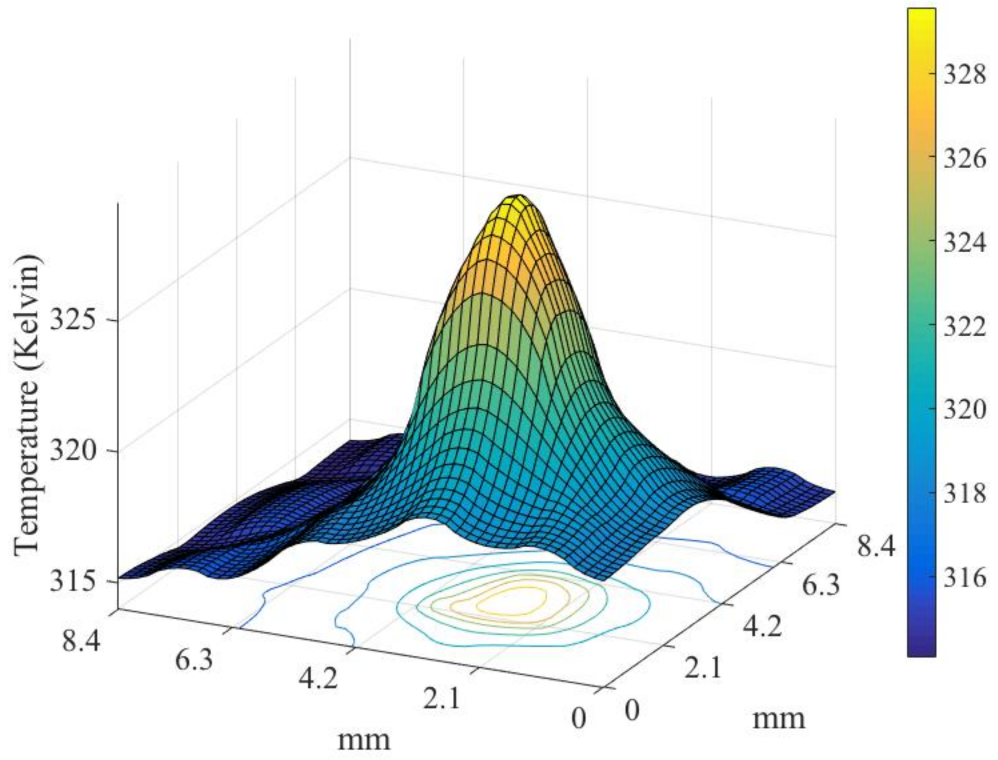


Figure 8.1: Heat map of the core layer showing reduction in thermal headroom for neighboring cores.

Consider a 3D architecture as illustrated in Figures 7.1 and 7.2, where 16 cores are integrated at the bottom of a 3D DRAM stack. When only one of the cores is executing an application thread while the rest are idle, the resulting thermal gradient from the 'hot' core to the neighboring 'cool' cores is shown in Figure 8.1. It is noted that the program thread executing on a core can increase the temperature of neighboring cores by as much as 7 Kelvin. Not shown here, is another observation that on migrating this thread from a location next on the package boundary to a location in the center of the core die decreases the temperature of the active core by up to 10 Kelvin (these are computed as steady state temperatures). *Ideally, one would like the thermal gradients to be zero, performance to be maximum, and the temperature to be the local EMT at every core.*

Achieving this goal via temperature regulation techniques alone are of limited utility. For example, consider the use of a temperature regulator [39] at each core (CHAPTER 6). The objective of the regulator is to maintain a fixed temperature. A graph benchmark is run and the target temperature is set to 340 Kelvin for each core. In Figure 8.2 it is observed that none of the cores can reach the target temperature. For cores which are idle i.e. threads are waiting to be woken up, the controller tries to raise the temperature of the core by increasing the corresponding voltage-frequency but ends up wasting energy due to increase in leakage power at higher temperatures. There is no improvement in the core performance. Temperature regulation in this form is therefore inefficient in 3D stacks because, (i) target tracking temperature (which is the EMT) has to be known *apriori*, (ii) target temperature will be different for different cores and will vary at run-time, and (iii) temperature dynamics is a rather slow process (100s of milli-seconds) in comparison to application characteristics that can vary rapidly (micro-seconds). *Therefore, control techniques must be on-line, adaptive, and application agnostic.*

The preceding example with temperature regulation re-emphasizes the important point made earlier - for certain applications and during certain application phases, package heat capacity is not utilized completely. This points to the existence of an EHC which corresponds to the temperature of the cores beyond which there is little increase in performance. This temperature is the EMT. It also represents an energy efficient (ops/joule) operating point. Note that the heat capacity of the entire package is established independent of the specific workload and that effective heat capacity of an application can be time varying. A thread currently in a memory intensive phase with EMT of X Kelvin, may transition into a compute intensive phase where its EMT is Y Kelvin ($Y > X$). Without profiling an application extensively, tracking the EMT is a challenge. To further illustrate the effect of EHC, data from two benchmark applications `blackscholes` (PARSEC [118]) and `tc` (GraphBig [145]) is presented in Table 8.1. The average temperature of the cores in Kelvin and average performance (Million-Instructions-Per-Second (MIPS)) for the two benchmarks is listed for three different fixed frequencies.

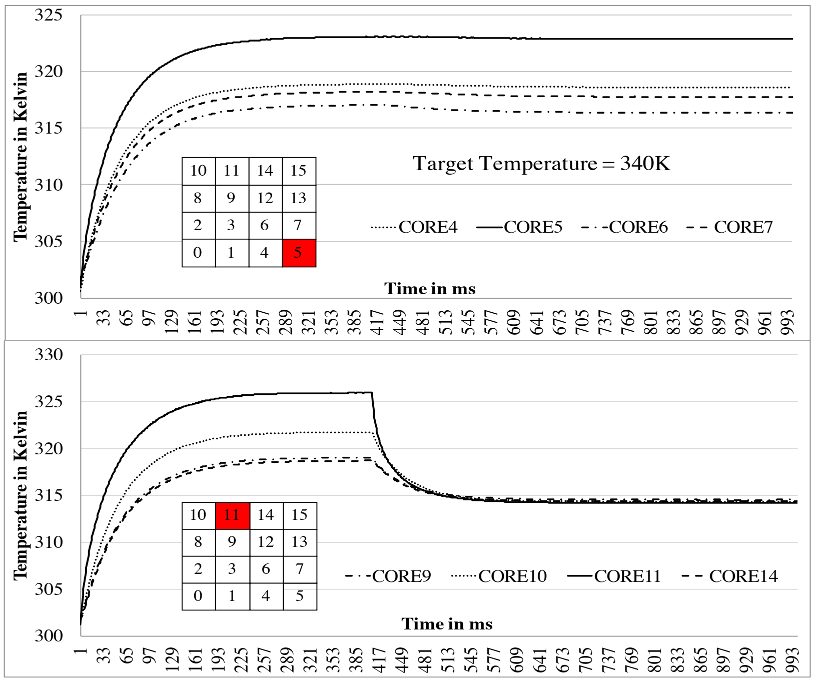


Figure 8.2: Temperature Regulation Inefficiency: Except for Core$_5$ and Core$_{11}$, rest of the cores are idle. At 400ms mark Core$_{11}$ becomes idle as well.

Table 8.1: Table demonstrating variable application heat capacities and room for improving balance between performance, temperature and energy.

	Bench.	**0.5GHz**	**1.0GHz**	**1.5GHz**
Temp. (K)	`blacks.`	318.18	329.68	340.93
	`tc`	313.91	318.93	323.85
Perf. (MIPS)	`blacks.`	12378.3	23710.7	33065.6
	`tc`	2741.3	3633.9	4026.7
ED^{2}P	`blacks.`	0.67	0.33	0.26
	`tc`	0.76	0.58	0.58

Performance and temperature characteristics of both applications vary widely. In order to demonstrate that there is room to improve performance and reduce energy and temperature in these systems, Energy Delay^2 Product (ED^{2}P) is also computed at the three fixed frequencies. For compute intensive applications like `blackscholes`, best ED^{2}P is achieved at the highest frequency. But, for memory intensive benchmarks like `tc`, there is no appreciable improvement in ED^{2}P beyond 1.0GHz, *The goal of a power/energy management algorithm should be to dynamically track these behaviors with distributed on-line control.*

It is important to make a distinction between peak temperature and effective heat capacity. The former is a constraint that all thermal management schemes seek to observe. Heat capacity reflects the net amount of heat that can be generated. Observing only the former will not maximize performance for the corresponding amount of heat. The target should be to extract as much performance as possible from the heat generated by the application. Thread scheduling techniques that seek to redistribute heat can be re-purposed towards this end. *In this sense, effective heat capacity is a resource which an on-line distributed controller network could be designed to exploit efficiently.*

8.2 TRINITY

This section details the proposed approach, TRINITY, an online DVFS controller that dynamically balances the <u>three</u> parameters: performance, energy and temperature to completely utilize the EHC in a <u>3D stack</u>. TRINITY is, (i) application agnostic, (ii) self-tuning, (iii) distributed (per-core), (iv) based on numerical optimization, and (v) computationally inexpensive to implement. The TRINITY controllers on each core are implicitly coupled via temperature. Therefore, the individual actions taken by the network of controllers works towards making the best use of the EHC. The controller is designed considering practical implementation challenges such as (i) measurement and actuation delays (ii) computation delays and (iii) hardware limitations such as having a discrete set of voltage-frequency states. The following sections present a detailed description of the system models, optimization problem and the solution approach.

8.2.1 System Models

The models used by optimization problems can be broadly classified into two types: (a) detailed, accurate and computationally expensive and (b) approximate, less accurate and computationally inexpensive. The latter is chosen because the goal is to design a controller that can be implemented on a physical machine. The loss in model accuracy affects the controller efficacy as will be highlighted in Section 8.3.

Power Model: The power model in [39] is linearized and a third order polynomial is arrived at which captures both leakage and dynamic power of the core. The equation is as follows:

$$P_k = \alpha f_k^3 + \beta f_k + \gamma T_k + \delta f_k T_k + \epsilon \tag{8.1}$$

where k represents the sample time instant, f_k and T_k are clock frequency and core temperature, respectively. The first term models the dynamic power and the last four terms of the equation represent leakage power. Since leakage power is strongly correlated with the technology node and packaging parameters, via non-linear regression, β, γ, δ and ϵ are calculated offline (See Table 8.2). To enable TRINITY to be application agnostic, the constant α, which represents the *activity factor* has to be determined online. Figure 8.3a shows that the approximation for the leakage power is within $\pm$5mW of the value measured on the simulator.

Table 8.2: Parameters estimated offline.

β	-426.7$\times 10^{-3}$	a_1	0.9998
γ	0.674$\times 10^{-3}$	b_1	8.46
δ	1.618$\times 10^{-3}$	c_1	37
ϵ	-90.38$\times 10^{-3}$	Δt	1ms

Temperature Model: Temperature at any given point in the 3D stack at any given time t is given by the dynamical equation

$$\dot{\mathbf{T}}(t) = \mathbf{A}\mathbf{T}(t) + \mathbf{B}\mathbf{P}(t) \tag{8.2}$$

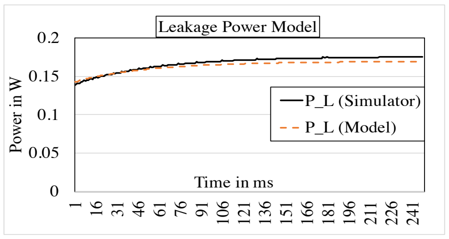
where $\mathbf{T}(t), \mathbf{P}(t) \in \mathbb{R}^M$ are the temperature and power vectors, respectively and the matrices **A** and **B** consist of the thermal resistance and capacitance of the 3D stack [160]. In each of the 6 layers in the 3D stack, broadly, there are 16 power dissipating elements, therefore, $M = 16 \times 6 = 96$. The large **A** and **B** matrices capture the inter-layer and intra-layer thermal coupling allowing for an accurate estimation of the temperature trajectory. At this juncture it is relevant to note that for the 3D stack under consideration, it is observed that the time constant for the rise in temperature is approximately 40ms, therefore, the settling time is around 200ms. These numbers are in agreement with practical observations [106]. Solving an optimization problem becomes increasingly computationally intensive as the dimensionality of the model increases (typically $O(M^3)$). Instead of using Eqn. 8.2 it is observed that discretizing and linearizing Eqn. 8.2 for a short duration of time Δt, reduces the model complexity drastically from $O(16^3) \rightarrow O(1)$. The price paid for this reduction in complexity is the loss of accuracy in predicting future temperature. Nonetheless, the temperature of a core can now be estimated Δt seconds into the future using the following scalar equation:

$$T_{k+1} = a_1 T_k + \Delta t(b_1 P_k + c_1) \tag{8.3}$$

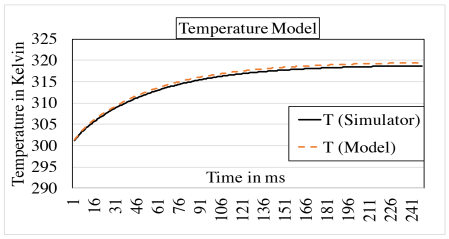
where it is seen that up to $\Delta t = 1$ms, the simplified temperature model is accurate to within 1 Kelvin as compared with values obtained from the simulator (See Fig. 8.3b). Analogous to the power model, the constants a_1, b_1 and c_1 are dependent on the technology node and packaging design choices. Therefore they are estimated offline via non-linear regression (See Table 8.2). The temperature estimate T_{k+1} depends on the *measured* values at time sample k and thus does not accumulate modeling errors at each time step.

Performance Model: Instruction throughput i.e. MIPS, is chosen as the metric. Performance is related to the clock frequency of the core via the following equation:

$$\chi_k(f_k) = IPC_k \cdot f_k \tag{8.4}$$



(a) Leakage Power Simulator vs. Model.



(b) Temperature Model Simulator vs. Model.

Figure 8.3: Leakage power and temperature model.

where IPC_k is Instructions-Per-Cycle and f_k is the core clock frequency at sample time k. As shall be described shortly, this linear approximation counterbalances temperature rise and is therefore sufficient for the purposes of the optimization problem under consideration.

8.2.2 Solution Strategy

The objective of the problem we wish to solve is encoded mathematically as follows:

$$\max_{f_{k+1}} \quad Qz_{k+1}^2 + R\chi_k(f_{k+1}) \tag{8.5}$$

subj. to

$$\underline{f} \leq f_{k+1} \leq \bar{f} \tag{8.6}$$

$$0 < z_{k+1} \tag{8.7}$$

where f_{k+1} is the core frequency which is within the bounds $\underline{f} = 0.5$GHz and $\bar{f} = 1.5$GHz. The term $z_{k+1} = T_{MAX} - T_{k+1}$, where T_{k+1} is the temperature of the core in Kelvin (Eqn. 8.3) and T_{MAX} is an upper bound for a core's temperature. The cost function described by Eqn. 8.5 consists of two parts, the former that penalizes increase in temperature and the latter that rewards performance. The weights Q and R ($Q, R > 0$ for problem feasibility) are tuning parameters that can be modified at run-time to give variable importance to performance and temperature. For the problem under consideration, $Q = 1$ and R is allowed to *tune itself* at run-time. Equations 8.5 - 8.7 are solved periodically after every $\mathbb{T}$ seconds by each core *independently* to determine f_{k+1}^*, the clock frequency that maximizes the cost function in Eqn. 8.5.

The intuition behind the problem definition is as follows: Consider an application whose performance saturates above a particular clock frequency and does not vary with time. The periodic calculation of f^*_{k+1} drives the system eventually towards a point where the temperature of the core reaches steady state. This steady state temperature is nothing but the EMT and any further increase in the clock frequency will reduce the cost thereby satisfying the original goal of maximally utilizing the EHC. For a particular choice of R, the behavior of the objective function is illustrated in Figure 8.4. The value $\mathbb{T}$, referred to as the control cycle, is a design parameter which has to be at least greater than (i) measurement, (ii) actuation, and (iii) computation delays. On processors available in the market currently, measurement and actuation delays are approximately 10s of micro seconds [174]. The control cycle also depends on the model accuracy since, as observed in the previous section, the simplified temperature model has sufficient accuracy up to a duration of 1ms. Therefore, $\mathbb{T}$ is set to 1ms in the experiments. Clock frequencies are spaced 50MHz apart between $0.5 - 1.5$GHz giving 21 discrete values. To solve the problem described in Eqn. 8.5, the three steps of the algorithm are listed in Fig. 8.5. Since each core solves the optimization problem independently, computing f^*_{k+1} requires finding the maximum in an array of 21 elements.

Coordination between individual controllers is via the temperature model in Eqn. 8.3. Temperature measured at time k is a consequence of control inputs at time $k - 1$. The decision f_{k+1} taken by a controller for the upcoming control cycle $k + 1$ depends on the control input of the neighboring controller at time k. This 'augmentation' of control inputs is reflected via the temperature measurements.

The tuning parameter R influences all three variables: temperature, performance and leakage energy. In fact, $R \in [R_{min}, R_{max}]$ such that for $R < R_{min}$, $f^*_{k+1} = \underline{f}$ and for $R > R_{max}$, $f^*_{k+1} = \bar{f}$. Emphasizing temperature over performance leads to lower leakage energy, whereas, greater importance to performance could potentially lead to wasted leak-

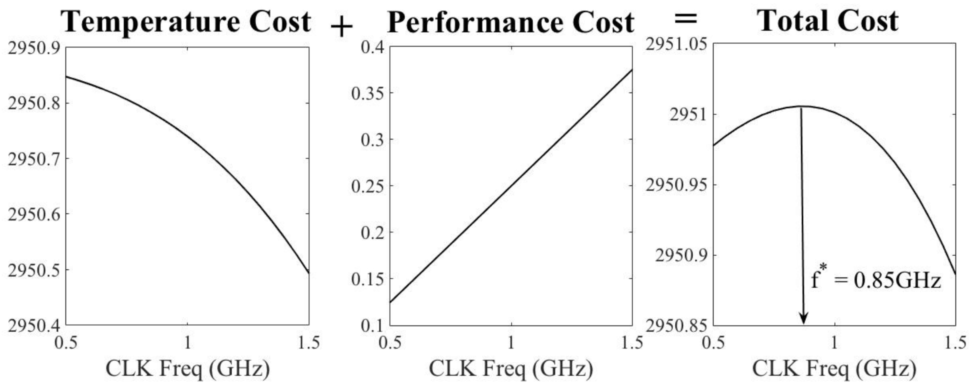


Figure 8.4: Behavior of the optimization cost.

age energy. Therefore, it is essential to choose an appropriate value in order to extract the desired behavior. Fixing the value of R is one approach. However, it is observed that such a strategy, (i) makes the solution application specific (ii) requires extensive time consuming offline analysis, and (iii) could easily push the controller into saturation where f^*_{k+1} will be remain at either $\underline{f}$ or $\bar{f}$ for prolonged periods of time. In order to adapt to dynamically varying application phases, R is allowed to re-calibrate itself periodically. This period is termed $\mathbb{T}_R \geq \mathbb{T}$. The pseudo code for the re-calibration is described in Figure 8.6. The re-calibration step basically determines the bounds for R i.e. $[R_{min}, R_{max}]$ and calculates the next value as $R = R_{min} + \eta(R_{max} - R_{min})$ where $\eta \in [0, 1]$. For the OoO core under consideration, $\eta = \sqrt{IPC_k/IPC_{max}}$ with IPC_{max} = Issue Width $= 4$. The IPC ratio heuristic is a means to obtain information about the compute or memory boundedness of the application. The square root of the ratio is chosen to push R towards R_{max}, and, hence, towards better performance.

8.3 Results

This section discusses the performance of TRINITY. The simulation environment is as described in 7.2.1 and the physical layout is as shown in Figures 7.2 and 7.1. The list

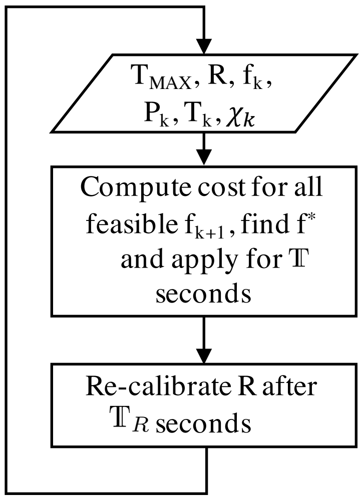

T_{MAX}, R, f_k, P_k, T_k, χ_k
Compute cost for all feasible f_{k+1}, find f^* and apply for $\mathbb{T}$ seconds
Re-calibrate R after $\mathbb{T}_R$ seconds

Figure 8.5: TRINITY Algorithm.

of benchmark applications used are detailed here. Next, an evaluation of the proposed control scheme is presented in detail. A DVFS strategy similar to the `ondemand` Linux CPU governor is implemented on the simulator and is compared against TRINITY. We also present results by fixing the core frequencies to 0.5, 1.0 and 1.5GHz. Since TRINITY attempts to balance performance, energy and temperature, Energy-Delay-Product (EDP), along with temperature, is used as the primary comparison metric. In what follows, Energy-Delay^2-Product (ED2P), Energy Efficiency (ops/Joule), performance (MIPS) and lifetime reliability measured as Mean Time To Failure (MTTF), are used to analyze TRINITY from different perspectives.

8.3.1 Benchmarks

The optimization technique proposed is evaluated over 6 benchmark applications from the PARSEC, Splash2x [118] and GraphBig [145] suite. Specifically, `blackscholes` and `barnes` (PARSEC and SPLASH2x) and `kcore`, `pagerank`, `connectedcomponent` and `tc` (GraphBig) are chosen. Each of the benchmark applications are executed with 16 threads. The GraphBig applications stress the memory whereas PARSEC and SPLASH2x stress the compute units thus giving a range of application behavior.

```
$R_{init} = 1\times10^{-6}$
$J(i) = z_{k+1}(i)^2 + R_{init} * \chi_{k+1}(i)$ ; $i = \{0,1,2,\dots,20\}$
$\Delta J_0 = J(1) - J(0)$ , $\Delta J_{20} = J(20) - J(19)$
$R = R_{init}$
$if$ $(\Delta J_0 < 0, \Delta J_{20} < 0)$
        Increase R until $\Delta J_0 > 0$ & $\Delta J_{20} < 0$
        $R_{MIN} = R$
        foo1($R_{MIN}$,1)

$if$ $(\Delta J_0 > 0, \Delta J_{20} > 0)$
        Decrease R until $\Delta J_0 > 0$ & $\Delta J_{20} < 0$
        $R_{MAX} = R$
        foo1($R_{MAX}$,0)
------------------------------------------------------------------
foo1($R_{now}$,flag) {
        $if$(flag == 1)
                Increase $R_{now}$ until $\Delta J_0 > 0$ & $\Delta J_{20} > 0$
                $R_{MAX} = R_{now}$
        $if$(flag == 0)
                Decrease $R_{now}$ until $\Delta J_0 < 0$ & $\Delta J_{20} < 0$
                $R_{MIN} = R_{now}$
}
```

Figure 8.6: Pseudo code for re-calibrating R.

8.3.2 Analyzing TRINITY Performance

EDP results are plotted in Figure 8.7 comparing TRINITY against the `ondemand` heuristic. The left y-axis represents EDP and the right y-axis represents the spatially averaged temperature of the core layer. The control cycle $\mathbb{T}$ and $\mathbb{T}_{\mathbb{R}}$ are set to 1ms. Calibrating R in the first control cycle requires some computational effort but subsequent re-calibrations can be optimized so as to allow R to be computed for every control cycle. Nevertheless, an analysis of controller efficacy with $\mathbb{T}_R = 5$ms is also discussed.

The trend of EDP is not the same for every benchmark. Consequently, the strategy to balance performance, energy and temperature should be different. For compute intensive workloads like `blackscholes` and `barnes`, the highest clock frequency (1.5GHz) delivers the best performance but also results in the highest temperature. This causes thermal throttling which significantly reduces performance. TRINITY on the other hand, tries to trade performance for benefits in temperature, 4 K on average with respect to `ondemand`. Energy efficiency results however reveal that TRINITY and `ondemand` perform equally well; TRINITY is 6.4% better, arguably within simulation error bounds. The implication is indeed in line with the definition of EMT. TRINITY chooses the clock frequencies such that same or better performance can be achieved at a much lower temperature.

Analyzing memory intensive benchmarks (`kcore`, `pagerank`, `connectedcomponent` and `tc`), there is no appreciable improvement in performance (MIPS) as core frequencies are increased. For example, average MIPS for `kcore` at 0.5, 1.0 and 1.5GHz is 4360.3, 5074.2 and 5265.2, respectively. Possessing *apriori* knowledge that the application to be executed is memory intensive, could lead to choosing a lower clock frequency as a possible strategy. While it certainly keeps the entire 3D stack at a lower temperature, EDP can suffer considerably. Although the average power is small, the application takes much longer to complete. In these situations, TRINITY tunes R in such a way that the lower half of the clock frequencies (0.5 - 1.0GHz) are chosen in the memory intensive phases. Except for `connectedcomponent`, temperature of the core layer for the remaining three workloads is lower by about 6 K. For `connectedcomponent`, both `ondemand` and TRINITY perform equally well and no appreciable temperature or EDP difference is observed.

To understand the source of temperature reduction, the average power dissipated at individual layers is plotted in Figure 8.8. The x-axis represents different DVFS options for each benchmark and the y-axis shows average power in Watts. Total power for each DVFS setting is broken down into dynamic and leakage power for the core, L2 Cache and DRAM layers. This distribution of power helps understand the primary source of

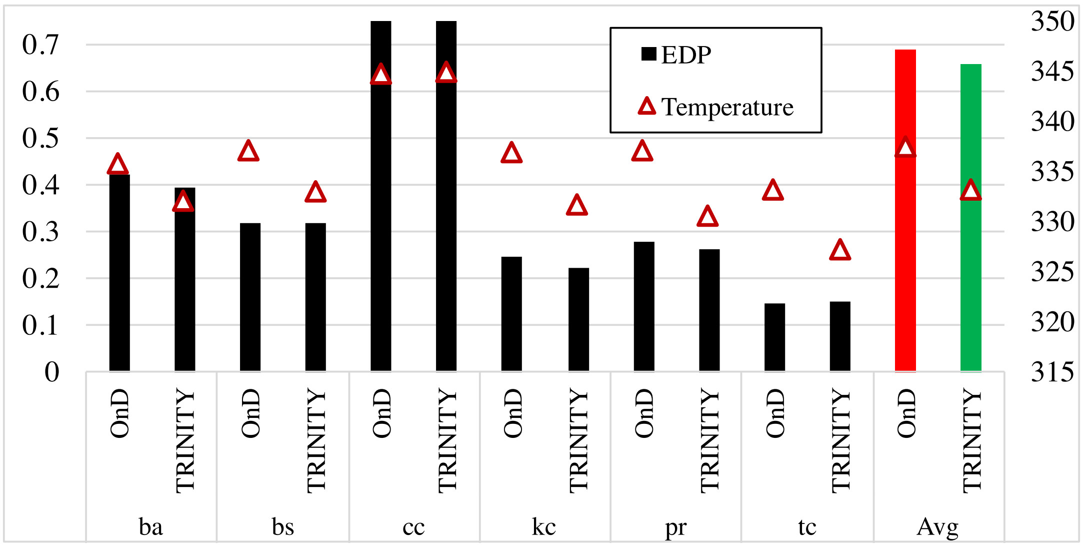


Figure 8.7: Controller performance compared against the `ondemand` heuristic. Controller Parameters: $T = 1\text{ms}$ and $T_R = 1\text{ms}$. Left y-axis and right y-axis units are EDP and Kelvin, respectively.

power consumption for each benchmark application. `blackscholes` and `barnes`, both compute intensive, consume majority of the power in the core layer. `kcore`, `pagerank` and `tc` being memory intensive consume greater power in the L2 Cache layer, specifically dynamic power. Additionally, DRAM dynamic power is higher as well due to increased L2 Cache misses. `connectedcomponent`, unlike other memory bound workloads shows much higher power consumed in the core die. However, power consumed in the L2 Cache and DRAM is larger as compared to compute intensive benchmarks.

As seen in Fig. 8.8, the bulk of the power reduction (consequently reduction in temperature) comes from reducing dynamic power consumption of the core and cache layers. This is intuitive since DVFS implemented by TRINITY directly affects only the core and the corresponding L2 Cache bank. As compared to `ondemand`, dynamic power of the core and cache layers reduce by 11.7% and 18%, respectively. Furthermore, with respect to `ondemand`, TRINITY is also able to reduce leakage power of the core and cache layers by 15.5% and 16.5%, respectively. The power reduction can be attributed to the on-line adaptation of R. In memory intensive parts of the application, η is low (< 1) thus guiding the controller to choose the lower end of the clock frequencies. In compute intensive regions,

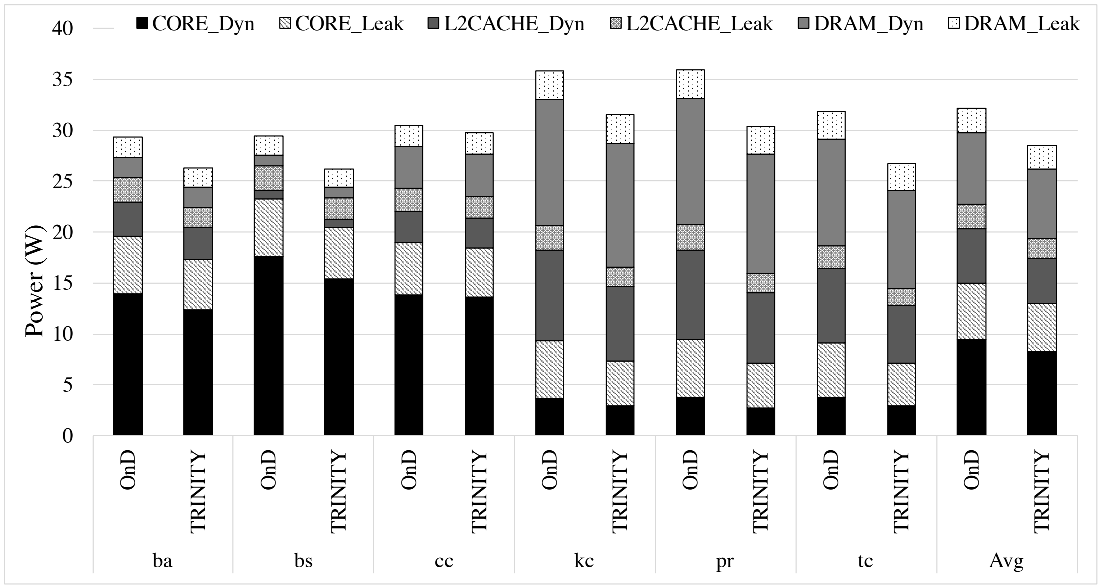


Figure 8.8: Average power consumption by TRINITY compared against the `ondemand` heuristic. Controller Parameters: $T = 1\text{ms}$ and $T_R = 1\text{ms}$.

η is high (> 2) allowing for higher clock frequencies to be chosen.

8.3.3 Impact on Lifetime Reliability

Changes in operating temperatures and voltages lead to significant impacts on reliability. Two dominant reliability mechanisms, electromigration (EM) and time-dependent dielectric breakdown (TDDB), are used to evaluate the reliability implications of TRINITY, compared to that of other execution modes. The reliability models and parameters are used from the work in [153] and references therein. Equations 8.8 and 8.9 show the reliability models of EM and TDDB, expressed as mean-time-to-failure (MTTF).

$$\text{MTTF}_{\text{EM}} = A_{\text{EM}} \times \frac{1}{t_{\text{act}}} \times V^{-n} \times e^{\frac{E_a}{kT}} \tag{8.8}$$

$$\text{MTTF}_{\text{TDDB}} = A_{\text{TDDB}} \times \frac{1}{t_{\text{act}}} \times V^{-c(a+bT)} \times e^{\frac{x+y/T+zT}{kT}} \tag{8.9}$$

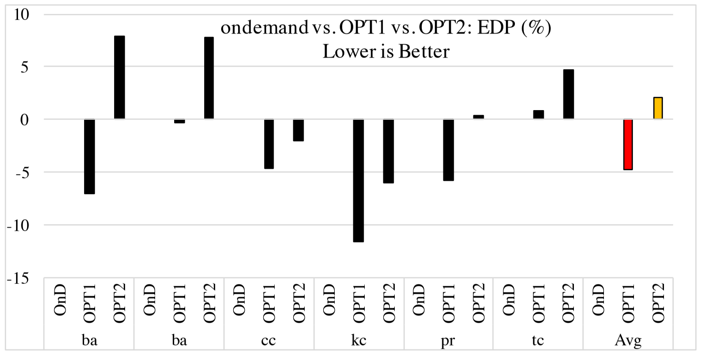
In the reliability equations, V and T are operating voltage and temperature. t_{act} ($0 \leq t_{\text{act}} \leq 1$) is active-state residency obtained from the execution time of each workload. For instance, $t_{\text{act}} = 0.5$ means that a workload utilizes the computing system for 50% of time. It is assumed that the system can be ideally power-gated for the remaining period and thus has no reliability impact; the system may be used to process other workloads, but resulting reliability impacts contribute to those workloads. k is Boltzmann's constant, and other parameters are model-dependent scaling parameters [153]. As shown in Eqn. 8.8, EM is primarily accelerated by temperature, and voltage has a secondary effect. In fact, a few degrees of average temperature change throughout the lifetime can easily produce several months to years of EM variations. On the other hand, TDDB is more sensitive to voltage changes, but temperature also has a non-negligible effect. The results show that TRINITY achieves 26% and 13%, better reliability than the `ondemand` heuristic, respectively.

8.3.4 Effect of TRINITY Parameter Variations

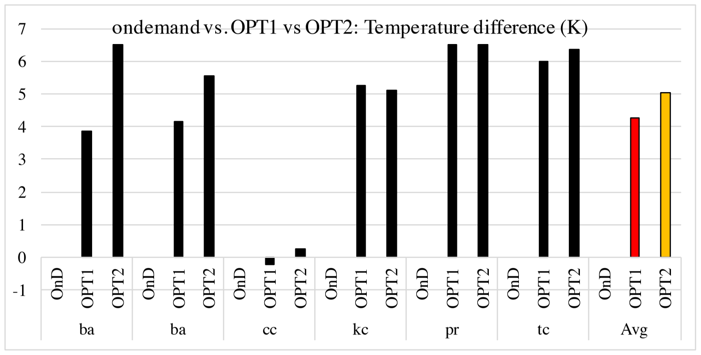
TRINITY is designed so that it can be implemented on a real physical system. Simplifying the model and reducing computational complexity reduces the number of parameters that can be manually tuned. In this section, the sensitivity of TRINITY to variations in $\mathbb{T}$ and $\mathbb{T}_R$, which are the only manually tuned parameters, is discussed. Reducing the control cycle duration and $\mathbb{T}_R$ has the benefit of capturing rapidly varying application phases. But it could also increase the amount of controller computations per unit time. Two cases are compared here: (1) OPT1 ($\mathbb{T} = 1$ms, $\mathbb{T}_R = 1$ms) and (2) OPT2 ($\mathbb{T} = 1$ms, $\mathbb{T}_R = 5$ms).

The y-axis in Figure 8.9b represents the absolute difference between TRINITY and `ondemand` whereas the y-axis in Figures 8.9a, 8.9d and 8.9c represent % difference. Overall, OPT2 fares slightly worse than OPT1 when using the metrics EDP, ED2P and ops/Joule. OPT2 keeps the temperature of the cores about a degree cooler than OPT1 by trading off performance. This aspect is clearly observed in Figures 8.9a and 8.9b.

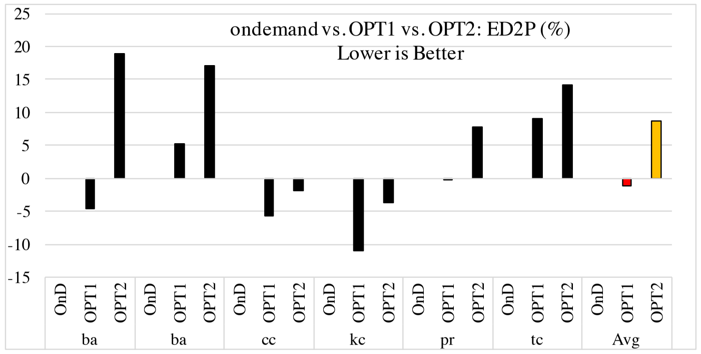
The notable feature concurs with intuition: Increasing $\mathbb{T}_R$ implies tuning R less frequently thereby making the controller less responsive to changes in application phases. This increases the effect of performance mispredictions and thus reduces EDP, ED2P and



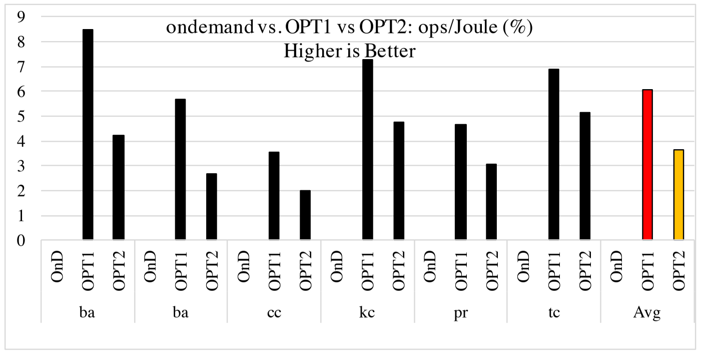
(a) Comparison of EDP against `ondemand` for different TRINITY parameters.



(b) Comparison of Temperature against `ondemand` for different TRINITY parameters.



(c) Comparison of ED2P against `ondemand` for different TRINITY parameters.



(d) Comparison of ops/Joule against `ondemand` for different TRINITY parameters.

Figure 8.9: TRINITY Parameter Variation

ops/Joule. Consequently, the average temperature for OPT2 is higher than OPT1.

Akin to any practical thermal/power/energy management approach, TRINITY too faces the challenge of modeling precision vs. controller performance. Applications that would benefit from TRINITY are those that have a mixture of compute and memory bound phases because of the ability to adapt itself at run-time to maximally utilize the EHC. However, if those phases are shorter than the control interval $\mathbb{T}$, they might end up being overlooked. TRINITY works particularly well for memory intensive applications like GraphBig because at the same EDP, the average temperature and voltage is lower than `ondemand` which improves MTTF by 10%.

8.4 Summary

This chapter presents an approach to the coordinated control of performance, energy and temperature on 3D processor-memory stacks. It introduces the concept of effective heat capacity as a thermal resource to be managed. Drawing inspiration from comprehensive simulation-based characterization of intra- and inter-die thermal coupling effects in CHAPTER 7 and examples described in Section 8.1, the ability to maximally utilize the effective heat capacity is illustrated. An on-line DVFS controller called TRINITY is developed to this goal. Unlike previous research efforts which (i) consider power, performance and temperature in isolation or in pairs, (ii) do not explicitly model static power, (iii) are heuristics based, this work acknowledges the complex interplay between performance, energy, temperature, microarchitectural parameters and package physical constraints. An analysis of EDP, ED2P, energy efficiency, temperature and also lifetime reliability is presented demonstrating the benefits of intelligently managing temperature as a resource and not just as a constraint.

CHAPTER 9
CONCLUSIONS AND FUTURE WORK

”Die stacking is happening in the mainstream. It is happening now because we need it. It is going to change who and how we build sockets in the future” - Bryan Black, Keynote MICRO - 2013 [44]. In the post Dennard scaling era, 3D stacking of silicon dice has emerged as a major contender for sustaining system performance in accordance with Moore’s Law. 3D stacked memory technologies such as HBM, HMC and Wide I/O provide an order of magnitude better memory bandwidth with better energy efficiency than current DDR standards. While die stacked memories are commercially available already, researchers have been exploring ways to tackle the memory wall problem by placing compute and memory in the same 3D package. The processor performance improves significantly but, better performance comes at the price of stringent thermal constraints. A fundamental understanding of the multi-physics interactions between performance, energy and temperature is critical to ensure the viability of 3D die stacked processor-memory architecture.

9.1 Thesis Conclusions

This thesis attempts to develop this understanding by first investigating various techniques on 2D architectures and later extrapolating the insights obtained to the 3D processor-memory architecture. Traditionally, power/energy/thermal management in 2D designs have been dominated by heuristics. Implementing a simple algorithm has been the driving force for architects to adopt such an approach. Of late, researchers have been slowly embracing a more formal control theoretic approach that can guarantee stability, robustness etc. This thesis points out the limits of dynamic management techniques based on (potentially multiple) heuristics and their inefficiency and inability to manage SoCs with multiple diverse components. Accordingly, the works presented in this thesis draw inspiration from

the rich area of control theory, and develops regulation and optimization techniques and demonstrates their efficacy on a cycle level simulator as well as some real physical systems. Furthermore, this thesis proposes simple, effective and robust feedback controllers that can be implemented on a real physical system with minimal overheads.
The main ideas developed in this thesis are listed below:

- Performance characterization of current power management strategies in SoCs demonstrating the scope for coordinated power management.
- A control-theoretic solution to the coordinated management of the core and the memory to minimize energy consumption for a target performance level.
- Extending the coordinated control framework to multi-core multi-memory-controller systems to improve energy efficiency.
- A distributed feedback controller to regulate core temperatures in a 2D multi-core processor.
- A comprehensive characterization of a 3D stacked processor-memory architecture.
- A distributed coordinated control framework for performance, energy and thermal management in a 3D stacked processor-memory architecture.

The first contribution of this thesis is the characterization of various power/energy management methods implemented in different hardware and software levels of the SoC stack, using a smartphone as an example of a mobile device. An important observation on the interaction between DVFS governors for the processor and memory is made during this investigation: Governors implementing their policies in an isolated manner are energy inefficient.

Equipped with this insight, a coordinated feedback controller managing the energy consumption of the processor and memory is implemented on a commercially available smartphone. The feedback controller is application-specific, in that, it utilizes performance and power data obtained offline. By choosing the CPU frequency and memory bandwidth simultaneously, it achieves $4 - 31\%$ better energy efficiency for a worst case performance loss of $< 1\%$ for 6 real world applications.

The next contribution expands the scope of the application-specific coordinated feedback controller to multi-core multi-memory controller systems. Simple analytic models for performance and power for the processor and memory are arrived at by regression. An online optimization, coupled with application phase detection, minimizes the EDP of the entire system. This technique is implemented on a cycle level simulator configured to simulate (i) single-core single-memory-controller and (ii) four-cores two-memory-controllers. The EDP results for both system configurations show that the controller indeed chooses a combination of CPU and memory frequency that gives the lowest system-wide EDP.

The next set of contributions of this thesis deal with temperature and its coupling with performance. As opposed to considering temperature as a constraint to be observed in an optimization problem, an adaptive gain integral controller is designed to regulate temperature of a core to a fixed value. Experiments on a cycle-level simulator demonstrates rapid convergence and robustness to modeling inaccuracies. It also highlights an important fact relating workload characteristics and temperature: The ability to utilize the package thermal capacity is dependent on the application phase and varies dynamically.

To investigate this further, a set of characterization experiments are conducted on a 3D stacked processor-memory architecture. A detailed characterization of the multi-physics interaction between (a) thermal behaviors (b) compute and memory microarchitecture and (c) application workloads is conducted. A concept called "effective heat capacity" is developed. It is the heat generated beyond which no further gains in performance is observed with increases in voltage-frequency of the compute logic.

The last part of this thesis attempts to maximize the utilization of the effective heat capacity in a 3D stacked processor-memory architecture. Accordingly, this thesis advocates the temperature as a *resource* just like compute or memory cycles. Supported by the observations from the detailed characterization a real-time, numerical optimization based, application agnostic controller, TRINITY, is developed for managing performance, energy *and* temperature. Compared to heuristics based schemes, TRINITY achieves up to 11% improvement in energy-delay-product while keeping temperature lower by 4 Kelvin. A secondary benefit is the increase in device reliability by up to 26%.

9.2 Future Work

The underlying essence of this thesis is the coordinated control of the *processor* and the *memory*. By carefully investigating the interactions between thermal behaviors, compute and memory microarchitecture and the applications, control theory based solutions for improving energy efficiency have been proposed. Today's computers, be it hand held mobile devices or high performance servers, are heterogeneous systems; A few examples are: ARM's big.LITTLE [93], AMD's APU (CPU + GPU) [175], heterogeneous memory systems (die stacked + off-chip + non-volatile memories etc.). Optimizing the energy efficiency of such systems with diverse components is a challenge and an open problem. Areas for potential future research are briefly described below.

Energy Management in Mobile Devices: Smartphones and hand held mobile devices are ubiquitous. CHAPTERS 3 and 4 are only a starting point. Managing the energy consumption of the SoC which contains a heterogeneous CPU, a GPU, off-chip memory, WiFi/LTE modules etc. in a distributed manner is a fertile area for future research. Instead of increasing the battery capacity to ensure longer battery life, intelligently managing energy consumed by individual components is a better option. Static one-off solutions based on heuristics is unsustainable. Consequently, formal control theoretic approaches, that are simple to implement on the firmware, for coordinating DVFS of different diverse modules on the SoC should be investigated.

Application-specific controllers are particularly well suited on mobile platforms. Additionally, mobile device manufacturers and application developers already collect usage statistics from the end user. However, the usage trends, device type, environmental conditions etc. can vary from person to person. Machine learning techniques can be applied to classify the collected data based on (i) set of top applications used by the customer, (ii) device type, (iii) usage conditions (hot vs. cold temperature areas) etc. Performance and power consumption data unique to a particular end-user can be periodically sent to his/her device to improve energy efficiency.

Thermal and Performance-Aware Data Mapping in stacked memories: 3D processor-memory stacks can have temporally and spatially varying thermal fields. Higher DRAM temperatures can result in serious performance reduction on account of increased refresh rates. Memory addressing patterns is another source of performance variation as described in CHAPTER 7. The trade-off between performance and temperature on account of various addressing patterns needs to be investigated thoroughly. The insights thus obtained can help in dynamically optimizing metrics for the memory such as energy-per-bit. Providing temperature feedback from the different memory layers to the OS can help maximize the utilization of the available bandwidth. Temperature-aware remapping of pages within the stacked memory layers along with memory DVFS has the potential to improve overall system performance.

REFERENCES

[1] C. Lefurgy, K. Rajamani, F. Rawson, W. Felter, M. Kistler, and T. W. Keller, "Energy management for commercial servers," *Computer*, vol. 36, no. 12, pp. 39–48, 2003.

[2] A. N. Udipi, N. Muralimanohar, N. Chatterjee, R. Balasubramonian, A. Davis, and N. P. Jouppi, "Rethinking dram design and organization for energy-constrained multi-cores," in *ACM SIGARCH Computer Architecture News*, ACM, vol. 38, 2010, pp. 175–186.

[3] D. Zhang, N. Jayasena, A. Lyashevsky, J. L. Greathouse, L. Xu, and M. Ignatowski, "Top-pim: Throughput-oriented programmable processing in memory," in *Proceedings of the 23rd international symposium on High-performance parallel and distributed computing*, ACM, 2014, pp. 85–98.

[4] M. Motoyoshi, "Through-silicon via (tsv)," *Proceedings of the IEEE*, vol. 97, no. 1, pp. 43–48, 2009.

[5] *Ddr3 sdram standard*, `https://www.jedec.org/standards-documents/docs/jesd-79-3d`, 2012.

[6] *Ddr4 sdram standard*, `https://www.jedec.org/standards-documents/docs/jesd79-4a`, 2017.

[7] *High bandwidth memory (hbm) dram*, `https://www.jedec.org/standards-documents/docs/jesd235a`, 2015.

[8] *Hybrid memory cube consortium*, `http://www.hybridmemorycube.org/`.

[9] K. Banerjee, S. J. Souri, P. Kapur, and K. C. Saraswat, "3-d ics: A novel chip design for improving deep-submicrometer interconnect performance and systems-on-chip integration," *Proceedings of the IEEE*, vol. 89, no. 5, pp. 602–633, 2001.

[10] C. C. Liu, I. Ganusov, M. Burtscher, and S. Tiwari, "Bridging the processor-memory performance gap with 3d ic technology," *IEEE Design & Test of Computers*, vol. 22, no. 6, pp. 556–564, 2005.

[11] B. Black, M. Annavaram, N. Brekelbaum, J. DeVale, L. Jiang, G. H. Loh, D. McCaule, P. Morrow, D. W. Nelson, D. Pantuso, *et al.*, "Die stacking (3d) microarchitecture," in *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, IEEE Computer Society, 2006, pp. 469–479.

[12] G. H. Loh, “3d-stacked memory architectures for multi-core processors,” in *ACM SIGARCH computer architecture news*, IEEE Computer Society, vol. 36, 2008, pp. 453–464.

[13] D. H. Woo, N. H. Seong, D. L. Lewis, and H.-H. S. Lee, “An optimized 3d-stacked memory architecture by exploiting excessive, high-density tsv bandwidth,” in *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, IEEE, 2010, pp. 1–12.

[14] S. Borkar, “3d integration for energy efficient system design,” in *Proceedings of the 48th Design Automation Conference*, ACM, 2011, pp. 214–219.

[15] P. Emma, A. Buyuktosunoglu, M. Healy, K. Kailas, V. Puente, R. Yu, A. Hartstein, P. Bose, and J. Moreno, “3d stacking of high-performance processors,” in *High Performance Computer Architecture (HPCA), 2014 IEEE 20th International Symposium on*, IEEE, 2014, pp. 500–511.

[16] *Iot devices will outnumber the world’s population this year for the first time*, `https://www.zdnet.com/article/iot-devices-will-outnumber-the-worlds-population-this-year-for-the-first-time/`.

[17] M. Halpern, Y. Zhu, and V. J. Reddi, “Mobile cpu’s rise to power: Quantifying the impact of generational mobile cpu design trends on performance, energy, and user satisfaction,” in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, IEEE, 2016, pp. 64–76.

[18] *Jedec releases lpddr4 standard for low power memory devices*, `https://www.jedec.org/news/pressreleases/jedec-releases-lpddr4-standard-low-power-memory-devices`.

[19] *Jedec publishes wide i/o 2 mobile dram standard*, `https://www.jedec.org/news/pressreleases/jedec-publishes-wide-io-2-mobile-dram-standard`.

[20] J. Ahn, S. Yoo, and K. Choi, “Dynamic power management of off-chip links for hybrid memory cubes,” in *Proceedings of the 51st Annual Design Automation Conference*, ACM, 2014, pp. 1–6.

[21] M. J. Khurshid and M. Lipasti, “Data compression for thermal mitigation in the hybrid memory cube,” in *2013 IEEE 31st International Conference on Computer Design (ICCD)*, IEEE, 2013, pp. 185–192.

[22] A. Agrawal, J. Torrellas, and S. Idgunji, “Xylem: Enhancing vertical thermal conduction in 3d processor-memory stacks,” in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, ACM, 2017, pp. 546–559.

[23] Q. Deng, D. Meisner, A. Bhattacharjee, T. F. Wenisch, and R. Bianchini, “Coscale: Coordinating cpu and memory system dvfs in server systems,” in *45th Annual IEEE/ACM International Symposium on Microarchitecture*, 2012, pp. 143–154.

[24] I. Paul, W. Huang, M. Arora, and S. Yalamanchili, “Harmonia: Balancing compute and memory power in high-performance gpus,” in *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, IEEE, 2015, pp. 54–65.

[25] 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 *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, IEEE Computer Society, 2014, pp. 445–457.

[26] R. Begum, D. Werner, M. Hempstead, G. Prasad, and G. Challen, “Energy-performance trade-offs on energy-constrained devices with multi-component dvfs,” in *Workload Characterization (IISWC), 2015 IEEE International Symposium on*, IEEE, 2015, pp. 34–43.

[27] A. Pathania, Q. Jiao, A. Prakash, and T. Mitra, “Integrated cpu-gpu power management for 3d mobile games,” in *Design Automation Conference (DAC), 2014 51st ACM/EDAC/IEEE*, IEEE, 2014, pp. 1–6.

[28] Y. Zhu, M. Halpern, and V. J. Reddi, “The role of the cpu in energy-efficient mobile web browsing,” *IEEE Micro*, vol. 35, no. 1, pp. 26–33, 2015.

[29] C. Ykman-Couvreur, P. Avasare, G. Mariani, G. Palermo, C. Silvano, and V. Zaccaria, “Linking run-time resource management of embedded multi-core platforms with automated design-time exploration,” *IET Computers & Digital Techniques*, vol. 5, no. 2, pp. 123–135, 2011.

[30] K. Skadron, T. Abdelzaher, and M. R. Stan, “Control-theoretic techniques and thermal-rc modeling for accurate and localized dynamic thermal management,” in *High-Performance Computer Architecture, 2002. Proceedings. Eighth International Symposium on*, IEEE, 2002, pp. 17–28.

[31] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, “Temperature-aware microarchitecture,” in *ACM SIGARCH Computer Architecture News*, ACM, vol. 31, 2003, pp. 2–13.

[32] N. Almoosa, W. Song, Y. Wardi, and S. Yalamanchili, “A power capping controller for multicore processors,” in *2012 American Control Conference (ACC)*, IEEE, 2012, pp. 4709–4714.

[33] X Chen, H Xiao, Y Wardi, and S Yalamanchili, "Throughput regulation in shared memory multicore processors," in *2015 IEEE 22nd International Conference on High Performance Computing (HiPC)*, IEEE, 2015, pp. 12–20.

[34] X. Wang, K. Ma, and Y. Wang, "Adaptive power control with online model estimation for chip multiprocessors," *IEEE Transactions on parallel and Distributed Systems*, vol. 22, no. 10, pp. 1681–1696, 2011.

[35] F. Zanini, D. Atienza, L. Benini, and G. De Micheli, "Multicore thermal management with model predictive control," in *Circuit Theory and Design, 2009. ECCTD 2009. European Conference on*, IEEE, 2009, pp. 711–714.

[36] S. Murali, A. Mutapcic, D. Atienza, R. Gupta, S. Boyd, L. Benini, and G. De Micheli, "Temperature control of high-performance multi-core platforms using convex optimization," in *2008 Design, Automation and Test in Europe*, IEEE, 2008, pp. 110–115.

[37] K. Rao, J. Wang, S. Yalamanchili, Y. Wardi, and H. Ye, "Application-specific performance-aware energy optimization on android mobile devices," *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2017.

[38] C. Imes, D. H. Kim, M. Maggio, and H. Hoffmann, "Poet: A portable approach to minimizing energy under soft real-time constraints," in *21st IEEE Real-Time and Embedded Technology and Applications Symposium*, 2015, pp. 75–86.

[39] K. Rao, W. Song, S. Yalamanchili, and Y. Wardi, "Temperature regulation in multicore processors using adjustable-gain integral controllers," in *2015 IEEE Conference on Control Applications (CCA)*, IEEE, 2015, pp. 810–815.

[40] W. A. Wulf and S. A. McKee, "Hitting the memory wall: Implications of the obvious," *ACM SIGARCH computer architecture news*, vol. 23, no. 1, pp. 20–24, 1995.

[41] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*. Elsevier, 2011.

[42] P. Machanick, "Approaches to addressing the memory wall," *School of IT and Electrical Engineering, University of Queensland*, 2002.

[43] A. White, "Exascale challenges: Applications, technologies, and co-design,"

[44] B. Black, "Die stacking is happening!" *In 46th IEEE/ACM International Symposium on Microarchitecture Keynote*, 2013.

[45] *High bandwidth memory, reinventing memory technology*, `http://www.amd.com/en-us/innovations/software-technologies/hbm`.

[46] J. T. Pawlowski, "Hybrid memory cube (hmc)," in *Hot Chips 23 Symposium (HCS), 2011 IEEE*, IEEE, 2011, pp. 1–24.

[47] V. Pallipadi and A. Starikovskiy, "The ondemand governor," in *Proceedings of the Linux Symposium*, sn, vol. 2, 2006, pp. 215–230.

[48] M. Kambadur and M. Kim, "An experimental survey of energy management across the stack," in *ACM International Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA)*, 2014, pp. 329–344.

[49] A. Miyoshi, C. Lefurgy, E. Hensbergen, R. Rajamony, and R. Rajkumar, "Critical power slope: Understanding the runtime effects of frequency scaling," in *16th International Conference on Supercomputing*, 2002, pp. 35–44.

[50] E. Rotem, "Intel architecture, code name skylake deep dive: A new architecture to manage power performance and energy efficiency," in *Intel Developer Forum*, 2015.

[51] E. Rotem, A. Naveh, A. Ananthakrishnan, E. Weissmann, and D. Rajwan, "Power-management architecture of the intel microarchitecture code-named sandy bridge," *Ieee micro*, vol. 32, no. 2, pp. 20–27, 2012.

[52] S. Jahagirdar, V. George, I. Sodhi, and R. Wells, "Power management of the third generation intel core micro architecture formerly codenamed ivy bridge," in *Hot Chips 24 Symposium (HCS), 2012 IEEE*, IEEE, 2012, pp. 1–49.

[53] B. Sinharoy, R Swanberg, N. Nayar, B Mealey, J. Stuecheli, B. Schiefer, J. Leenstra, J Jann, P Oehler, D Levitan, *et al.*, "Advanced features in ibm power8 systems," *IBM Journal of Research and Development*, vol. 59, no. 1, pp. 1–1, 2015.

[54] *Amd phenom ii key architectural features*, `http://www.amd.com/us/products/desktop/processors/phenom-ii/Pages/phenom-ii-key-architectural-features.aspx`.

[55] E. Le Sueur and G. Heiser, "Dynamic voltage and frequency scaling: The laws of diminishing returns," in *Proceedings of the 2010 international conference on Power aware computing and systems*, 2010, pp. 1–8.

[56] 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.

[57] J. Heo, D. Henriksson, X. Liu, and T. Abdelzaher, "Integrating adaptive components: An emerging challenge in performance-adaptive systems and a server farm case-study," in *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*, IEEE, 2007, pp. 227–238.

[58] R. P. Pothukuchi, A. Ansari, P. Voulgaris, and J. Torrellas, "Using multiple input, multiple output formal control to maximize resource efficiency in architectures," in *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*, IEEE, 2016, pp. 658–670.

[59] V. Hanumaiah, D. Desai, B. Gaudette, C.-J. Wu, and S. Vrudhula, "Steam: A smart temperature and energy aware multicore controller," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 13, no. 5s, p. 151, 2014.

[60] Y. Wang, K. Ma, and X. Wang, "Temperature-constrained power control for chip multiprocessors with online model estimation," in *ACM SIGARCH computer architecture news*, ACM, vol. 37, 2009, pp. 314–324.

[61] S. Fan, S. M. Zahedi, and B. C. Lee, "The computational sprinting game," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 2, pp. 561–575, 2016.

[62] R. Bitirgen, E. Ipek, and J. F. Martinez, "Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach," in *Microarchitecture, 2008. MICRO-41. 2008 41st IEEE/ACM International Symposium on*, IEEE, 2008, pp. 318–329.

[63] X. Wang and J. F. Martínez, "Rebudget: Trading off efficiency vs. fairness in market-based multicore resource allocation via runtime budget reassignment," *ACM SIGPLAN Notices*, vol. 51, no. 4, pp. 19–32, 2016.

[64] ——, "Xchange: A market-based approach to scalable dynamic multi-resource allocation in multicore architectures," in *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, IEEE, 2015, pp. 113–125.

[65] S. S. Jha, W. Heirman, A. Falcón, T. E. Carlson, K. Van Craeynest, J. Tubella, A. González, and L. Eeckhout, "Chrysso: An integrated power manager for constrained many-core processors," in *Proceedings of the 12th ACM International Conference on Computing Frontiers*, ACM, 2015, p. 19.

[66] W.-Y. Liang and P.-T. Lai, "Design and implementation of a critical speed-based dvfs mechanism for the android operating system," in *The 5th International Conference on Embedded and Multimedia Computing (EMC)*, 2010, pp. 1–6.

[67] A. K. Mishra, S. Srikantaiah, M. Kandemir, and C. R. Das, "Cpm in cmps: Coordinated power management in chip-multiprocessors," in *High Performance Com-*

puting, Networking, Storage and Analysis (SC), 2010 International Conference for, IEEE, 2010, pp. 1–12.

[68] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu, “No power struggles: Coordinated multi-level power management for the data center,” in *ACM SIGARCH Computer Architecture News*, ACM, vol. 36, 2008, pp. 48–59.

[69] P. Juang, Q. Wu, L.-S. Peh, M. Martonosi, and D. W. Clark, “Coordinated, distributed, formal energy management of chip multiprocessors,” in *Proceedings of the 2005 international symposium on Low power electronics and design*, ACM, 2005, pp. 127–130.

[70] X Chen, Y Wardi, and S Yalamanchili, “Power regulation in high performance multicore processors,” *arXiv preprint arXiv:1709.04859*, 2017.

[71] N. Almoosa, W Song, S. Yalamanchili, and Y. Wardi, “Throughput regulation in multicore processors via ipa,” in *Decision and Control (CDC), 2012 IEEE 51st Annual Conference on*, IEEE, 2012, pp. 7267–7272.

[72] W.-Y. Liang, M.-F. Chang, Y.-L. Chen, and C.-F. Lai, “Energy efficient video decoding for the android operating system,” in *IEEE International Conference on Consumer Electronics (ICCE)*, 2013, pp. 344–345.

[73] O. Sahin, P. T. Varghese, and A. K. Coskun, “Just enough is more: Achieving sustainable performance in mobile devices under thermal limitations,” in *Computer-Aided Design (ICCAD), 2015 IEEE/ACM International Conference on*, IEEE, 2015, pp. 839–846.

[74] J. Donald and M. Martonosi, “Techniques for multicore thermal management: Classification and new exploration,” in *ACM SIGARCH Computer Architecture News*, IEEE Computer Society, vol. 34, 2006, pp. 78–88.

[75] A. Bartolini, M. Cacciari, A. Tilli, and L. Benini, “Thermal and energy management of high-performance multicores: Distributed and self-calibrating model-predictive controller,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 1, pp. 170–183, 2013.

[76] B. Shi, Y. Zhang, and A. Srivastava, “Dynamic thermal management for single and multicore processors under soft thermal constraints,” in *Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design*, ACM, 2010, pp. 165–170.

[77] Y. Fu, N. Kottenstette, C. Lu, and X. D. Koutsoukos, “Feedback thermal control of real-time systems on multicore processors,” in *Proceedings of the tenth ACM international conference on Embedded software*, ACM, 2012, pp. 113–122.

[78] S. Mittal, “A survey of architectural techniques for dram power management,” *International Journal of High Performance Systems Architecture*, vol. 4, no. 2, pp. 110–119, 2012.

[79] C. Weis, I. Loi, L. Benini, and N. Wehn, “Exploration and optimization of 3-d integrated dram subsystems,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 4, pp. 597–610, 2013.

[80] K. Kang, J. Jung, S. Yoo, and C.-M. Kyung, “Maximizing throughput of temperature-constrained multi-core systems with 3d-stacked cache memory,” in *Quality Electronic Design (ISQED), 2011 12th International Symposium on*, IEEE, 2011, pp. 1–6.

[81] W.-H. Lo, K.-z. Liang, and T. Hwang, “Thermal-aware dynamic page allocation policy by future access patterns for hybrid memory cube (hmc),” in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2016, pp. 1084–1089.

[82] J. Meng, K. Kawakami, and A. K. Coskun, “Optimizing energy efficiency of 3-d multicore systems with stacked dram under power and thermal constraints,” in *Proceedings of the 49th Annual Design Automation Conference*, ACM, 2012, pp. 648–655.

[83] D. Zhao, H. Homayoun, and A. V. Veidenbaum, “Temperature aware thread migration in 3d architecture with stacked dram,” in *Quality Electronic Design (ISQED), 2013 14th International Symposium on*, IEEE, 2013, pp. 80–87.

[84] L.-N. Tran, F. J. Kurdahi, A. M. Eltawil, and H. Homayoun, “Heterogeneous memory management for 3d-dram and external dram with qos,” in *Design Automation Conference (ASP-DAC), 2013 18th Asia and South Pacific*, IEEE, 2013, pp. 663–668.

[85] P. Balaprakash, D. Buntinas, A. Chan, A. Guha, R. Gupta, S. H. K. Narayanan, A. A. Chien, P. Hovland, and B. Norris, “Exascale workload characterization and architecture implications,” in *Proceedings of the High Performance Computing Symposium*, Society for Computer Simulation International, 2013, p. 5.

[86] G. L. Loi, B. Agrawal, N. Srivastava, S.-C. Lin, T. Sherwood, and K. Banerjee, “A thermally-aware performance analysis of vertically integrated (3-d) processor-memory hierarchy,” in *Proceedings of the 43rd annual Design Automation Conference*, ACM, 2006, pp. 991–996.

[87] Y.-J. Chen, C.-L. Yang, P.-S. Lin, and Y.-C. Lu, “Thermal/performance characterization of cmps with 3d-stacked drams under synergistic voltage-frequency control

of cores and drams," in *Proceedings of the 2015 Conference on research in adaptive and convergent systems*, ACM, 2015, pp. 430–436.

[88] K. Puttaswamy and G. H. Loh, "Thermal herding: Microarchitecture techniques for controlling hotspots in high-performance 3d-integrated processors," in *High Performance Computer Architecture, 2007. HPCA 2007. IEEE 13th International Symposium on*, IEEE, 2007, pp. 193–204.

[89] L. Yang, R. Dick, H. Lekatsas, and S. Chakradhar, "Online memory compression for embedded systems," *ACM Transactions on Embedded Computing Systems*, vol. 9, no. 3, 2010.

[90] W. Zang and A. Gordon-Ross, "A survey on cache tuning from a power/energy perspective," *ACM Computing Survey*, vol. 45, no. 3, 2013.

[91] T. Kolpe, A. Zhai, and S. Sapatnekar, "Enabling improved power management in multicore processors through clustered dvfs," in *Design, Automation, and Test in Europe*, 2011, pp. 1–6.

[92] K. Ma, X. Li, W. Chen, C. Zhang, and X. Wang, "Greengpu: A holistic approach to energy efficiency in gpu-cpu heterogeneous architectures," in *41st International Conference on Parallel Processing*, 2012, pp. 48–57.

[93] *Big.little technology*, `https://www.arm.com/products/processors/technologies/biglittleprocessing.php`.

[94] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced cpu energy," in *1st USENIX Conference on Operating System Design and Implementation*, 1994.

[95] J. Chen and L. John, "Energy-aware application scheduling on a heterogeneous multi-core system," in *IEEE International Symposium on Workload Characterization*, 2008, pp. 5–13.

[96] G. Liu, M. Fan, and G. Quan, "Neighbor-aware dynamic thermal management for multi-core platform," in *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2012, pp. 187–192.

[97] I. Yeo, C. C. Liu, and E. J. Kim, "Predictive dynamic thermal management for multicore systems," in *Proceedings of the 45th annual Design Automation Conference*, ACM, 2008, pp. 734–739.

[98] *Energy aware scheduling (eas) progress update*, `http://www.linaro.org/blog/core-dump/energy-aware-scheduling-eas-progress-update/`.

[99] K. Cooper and L. Torczon, *Engineering a Compiler*, 2nd. Morgan Kaufmann, 2012.

[100] O. Asare and M. Goudarzi, “Opportunities fo embedded software power reductions,” in *24th Canadian Conference on Electrical and Computer Engineering*, 2011, pp. 763–766.

[101] J. Ayala and M. López-Vallejo, “Improving register file banking with a power-aware unroller,” in *Proceedings of PARC*, 2004.

[102] J. Pallister, S. Hollis, and J. Bennett, *Identifying compiler options to minimise energy consumption for embedded platforms*, `http://arxiv.org/abs/1303.6485`, 2013.

[103] J. Cebrián, L. Natvig, and J. Meyer, “Improving energy efficiency through parallelization and vectorization on intel core i5 and i7 processors,” in *SC Companion: High Performance Computing, Networking, Storage and Analysis*, 2012, pp. 675–684.

[104] T. Hönig, C. Eibel, R. Kapitza, and W. Schröder-Preikschat, “Seep: Exploiting symbolic execution for energy-aware programming,” *ACM SIGOPS Operating Systems Review*, vol. 45, no. 3, pp. 58–62, 2011.

[105] M. Kambadur and M. Kim, “Nrg-loops: Adjusting power from within applications,” in *International Symposium on Code Generation and Optimization*, 2016, pp. 206–215.

[106] I. Paul, S. Manne, M. Arora, W. L. Bircher, and S. Yalamanchili, “Cooperative boosting: Needy versus greedy power management,” in *ACM SIGARCH Computer Architecture News*, ACM, vol. 41, 2013, pp. 285–296.

[107] I. Paul, V. Ravi, S. Manne, M. Arora, and S. Yalamanchili, “Coordinated energy management in heterogeneous processors,” *Scientific Programming*, vol. 22, no. 2, pp. 93–108, 2014.

[108] Q. Deng, D. Meisner, L. Ramos, T. F. Wenisch, and R. Bianchini, “Memscale: Active low-power modes for main memory,” in *ACM SIGPLAN Notices*, ACM, vol. 46, 2011, pp. 225–238.

[109] M. H. Santriaji and H. Hoffman, “Grape: Minimizing energy for gpu applications with performance requirements,” *IEEE MICRO*, 2016.

[110] *Monsoon power monitor*, `https://www.msoon.com/LabEquipment/PowerMonitor/`.

[111] *Android debug bridge*, `https://developer.android.com/studio/command-line/adb.html`.

[112] *Antutu*, `http://www.antutu.com/en/index.shtml`.

[113] *Geekbench3*, `http://support.primatelabs.com/kb/geekbench/interpreting-geekbench-3-scores`.

[114] *Vellamo*, `https://play.google.com/store/apps/details?id=com.quicinc.vellamo&hl=en`.

[115] *3dmark*, `http://www.futuremark.com/benchmarks/3dmark/android`.

[116] L. Gomez, I. Neamtiu, T. Azim, and T. Millstein, "Reran: Timing-and touch-sensitive record and replay for android," in *2013 35th International Conference on Software Engineering (ICSE)*, IEEE, 2013, pp. 72–81.

[117] D. Pandiyan, S.-Y. Lee, and C.-J. Wu, "Performance, energy characterizations and architectural implications of an emerging mobile platform benchmark suite-mobilebench," in *Workload Characterization (IISWC), 2013 IEEE International Symposium on*, IEEE, 2013, pp. 133–142.

[118] 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*, ACM, 2008, pp. 72–81.

[119] J. L. Henning, "Spec cpu2006 benchmark descriptions," *ACM SIGARCH Computer Architecture News*, vol. 34, no. 4, pp. 1–17, 2006.

[120] D. Brodowski, "Linux kernel cpufreq subsystem," *`http://www.kernel.org/pub/linux/utils/kernel/cpufreq/cpufreq-info.html`*,

[121] *Advanced configuration and power interface*, `http://www.acpi.info/`.

[122] C. Gao, A. Gutierrez, M. Rajan, R. G. Dreslinski, T. Mudge, and C.-J. Wu, "A study of mobile device utilization," in *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2015, pp. 225–234.

[123] H. David, C. Fallin, E. Gorbatov, U. R. Hanebutte, and O. Mutlu, "Memory power management via dynamic voltage/frequency scaling," in *Proceedings of the 8th ACM international conference on Autonomic computing*, ACM, 2011, pp. 31–40.

[124] P. T. Bezerra, L. A. Araujo, G. B. Ribeiro, A. C.d.S. B. Neto, A. G. Silva-Filho, C. A. Siebra, F. Q.B. da Silva, A. L. Santos, A. Mascaro, and P. H. Costa, "Dy-

namic frequency scaling on android platforms for energy consumption reduction," in *Proceedings of the 8th ACM Workshop on Performance Monitoring and Measurement of Heterogeneous Wireless and Wired Networks*, ser. PM^{2}HW^{2}N '13, 2013, pp. 189–196.

[125] C. Isci, A. Buyuktosunoglu, C.-Y. Cher, P. Bose, and M. Martonosi, "An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget," in *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, 2006, pp. 347–358.

[126] T. Patikirikorala and A. Colman, "Feedback controllers in the cloud," in *Proceedings of APSEC*, 2010.

[127] L. Cao and H. M. Schwartz, "Analysis of the kalman filter based estimation algorithm: An orthogonal decomposition approach," *Automatica*, vol. 40, no. 1, pp. 5–19, 2004.

[128] *Ffmpeg library*, `https://ffmpeg.org/about.html`.

[129] A. Gutierrez, R. G. Dreslinski, T. F. Wenisch, T. Mudge, A. Saidi, C. Emmons, and N. Paver, "Full-system analysis and characterization of interactive smartphone applications," in *Workload Characterization (IISWC), 2011 IEEE International Symposium on*, IEEE, 2011, pp. 81–90.

[130] C. Isci, G. Contreras, and M. Martonosi, "Live, runtime phase monitoring and prediction on real systems with application to dynamic power management," in *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, 2006, pp. 359–370.

[131] J. Lau, S. Schoemackers, and B. Calder, "Structures for phase classification," in *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2004, pp. 57–67.

[132] M. Ware, K. Rajamani, M. Floyd, B. Brock, J. C. Rubio, F. Rawson, and J. B. Carter, "Architecting for power management: The ibm® power7 approach," in *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, IEEE, 2010, pp. 1–11.

[133] V. Pandey, W. Jiang, Y. Zhou, and R. Bianchini, "Dma-aware memory energy management.," in *HPCA*, vol. 6, 2006, pp. 133–144.

[134] A. R. Lebeck, X. Fan, H. Zeng, and C. Ellis, "Power aware page allocation," *ACM Sigplan Notices*, vol. 35, no. 11, pp. 105–116, 2000.

[135] H. Huang, K. G. Shin, C. Lefurgy, and T. Keller, "Improving energy efficiency by making dram less randomly accessed," in *Proceedings of the 2005 international symposium on Low power electronics and design*, ACM, 2005, pp. 393–398.

[136] X. Fan, C. Ellis, and A. Lebeck, "Memory controller policies for dram power management," in *Proceedings of the 2001 international symposium on Low power electronics and design*, ACM, 2001, pp. 129–134.

[137] B. Diniz, D. Guedes, W. Meira Jr, and R. Bianchini, "Limiting the power consumption of main memory," *ACM SIGARCH Computer Architecture News*, vol. 35, no. 2, pp. 290–301, 2007.

[138] V. Delaluz, M. Kandemir, N. Vijaykrishnan, A. Sivasubramaniam, and M. J. Irwin, "Hardware and software techniques for controlling dram power modes," *IEEE Transactions on Computers*, vol. 50, no. 11, pp. 1154–1173, 2001.

[139] J. H. Ahn, J. Leverich, R. Schreiber, and N. P. Jouppi, "Multicore dimm: An energy efficient memory module with independently controlled drams," *IEEE Computer Architecture Letters*, vol. 8, no. 1, pp. 5–8, 2009.

[140] H. Zheng, J. Lin, Z. Zhang, E. Gorbatov, H. David, and Z. Zhu, "Mini-rank: Adaptive dram architecture for improving memory power efficiency," in *Microarchitecture, 2008. MICRO-41. 2008 41st IEEE/ACM International Symposium on*, IEEE, 2008, pp. 210–221.

[141] E. Cooper-Balis and B. Jacob, "Fine-grained activation for power reduction in dram," *IEEE Micro*, vol. 30, no. 3, pp. 34–47, 2010.

[142] Q. Deng, D. Meisner, A. Bhattacharjee, T. F. Wenisch, and R. Bianchini, "Multiscale: Memory system dvfs with multiple memory controllers," in *Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design*, ACM, 2012, pp. 297–302.

[143] *Tn-41-01:calculating memory system power for ddr3*, `https://www.micron.com/resource-details/3465e69a-3616-4a69-b24d-ae459b295aae`.

[144] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "Dramsim2: A cycle accurate memory system simulator," *IEEE Computer Architecture Letters*, vol. 10, no. 1, pp. 16–19, 2011.

[145] L. Nai, Y. Xia, I. G. Tanase, H. Kim, and C.-Y. Lin, "Graphbig: Understanding graph computing in the context of industrial solutions," in *High Performance Computing, Networking, Storage and Analysis, 2015 SC-International Conference for*, IEEE, 2015, pp. 1–12.

[146] D. Abts, N. D. Enright Jerger, J. Kim, D. Gibson, and M. H. Lipasti, “Achieving predictable performance through better memory controller placement in many-core cmps,” *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3, pp. 451–461, 2009.

[147] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, J. F. Brown III, and A. Agarwal, “On-chip interconnection architecture of the tile processor,” *IEEE micro*, vol. 27, no. 5, pp. 15–31, 2007.

[148] M. Awasthi, D. Nellans, K. Sudan, R. Balasubramonian, and A. Davis, “Handling the problems and opportunities posed by multiple on-chip memory controllers,” in *Parallel Architectures and Compilation Techniques (PACT), 2010 19th International Conference on*, IEEE, 2010, pp. 319–330.

[149] 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, no. 5, pp. 256–268, 1974.

[150] *International technology roadmap for semiconductors (itrs), 2011*, `http://www.itrs2.net/2011-itrs.html`.

[151] J. Kong, S. W. Chung, and K. Skadron, “Recent thermal management techniques for microprocessors,” *ACM Computing Surveys (CSUR)*, vol. 44, no. 3, p. 13, 2012.

[152] H. Qian, X. Huang, H. Yu, and C. H. Chang, “Cyber-physical thermal management of 3d multi-core cache-processor system with microfluidic cooling,” *Journal of Low Power Electronics*, vol. 7, no. 1, pp. 110–121, 2011.

[153] W. J. Song, S. Mukhopadhyay, and S. Yalamanchili, “Managing performance-reliability tradeoffs in multicore processors,” in *Reliability Physics Symposium (IRPS), 2015 IEEE International*, IEEE, 2015, pp. 3C–1–12.

[154] P. Lancaster, “Error analysis for the newton-raphson method,” *Numerische Mathematik*, vol. 9, no. 1, pp. 55–68, 1966.

[155] J. M. Ortega and W. C. Rheinboldt, *Iterative solution of nonlinear equations in several variables*. Siam, 1970, vol. 30.

[156] T. D. Burd, T. A. Pering, A. J. Stratakos, and R. W. Brodersen, “A dynamic voltage scaled microprocessor system,” *IEEE Journal of solid-state circuits*, vol. 35, no. 11, pp. 1571–1580, 2000.

[157] R. McGowen, C. A. Poirier, C. Bostak, J. Ignowski, M. Millican, W. H. Parks, and S. Naffziger, “Power and temperature control on a 90-nm itanium family processor,” *IEEE Journal of Solid-State Circuits*, vol. 41, no. 1, pp. 229–237, 2006.

[158] J. A. Butts and G. S. Sohi, "A static power model for architects," in *Proceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture*, ACM, 2000, pp. 191–201.

[159] J. Rabaey, *Low power design essentials*. Springer Science & Business Media, 2009.

[160] Y. Han, I. Koren, and C. M. Krishna, "Tilts: A fast architectural-level transient thermal simulation method," *Journal of Low Power Electronics*, vol. 3, no. 1, pp. 13–21, 2007.

[161] J. Wang, J. Beu, R. Bheda, T. Conte, Z. Dong, C. Kersey, M. Rasquinha, G. Riley, W. Song, H. Xiao, *et al.*, "Manifold: A parallel simulation framework for multicore systems," in *Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on*, IEEE, 2014, pp. 106–115.

[162] W. J. Song, S. Mukhopadhyay, and S. Yalamanchili, "Kitfox: Multiphysics libraries for integrated power, thermal, and reliability simulations of multicore microarchitecture," *IEEE Transactions on Components, Packaging and Manufacturing Technology*, vol. 5, no. 11, pp. 1590–1601, 2015.

[163] C. D. Kersey, A. Rodrigues, and S. Yalamanchili, "A universal parallel front-end for execution driven microarchitecture simulation," in *Proceedings of the 2012 Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools*, ACM, 2012, pp. 25–32.

[164] *Hotspot version 5.0*, `http://lava.cs.virginia.edu/HotSpot/index.htm`.

[165] Y. Eckert, N. Jayasena, and G. H. Loh, "Thermal feasibility of die-stacked processing in memory," 2014.

[166] D. Milojevic, S. Idgunji, D. Jevdjic, E. Ozer, P. Lotfi-Kamran, A. Panteli, A. Prodromou, C. Nicopoulos, D. Hardy, B. Falsari, *et al.*, "Thermal characterization of cloud workloads on a power-efficient server-on-chip," in *Computer Design (ICCD), 2012 IEEE 30th International Conference on*, IEEE, 2012, pp. 175–182.

[167] K. Puttaswamy and G. H. Loh, "Thermal analysis of a 3d die-stacked high-performance microprocessor," in *Proceedings of the 16th ACM Great Lakes symposium on VLSI*, ACM, 2006, pp. 19–24.

[168] R. Zhang, M. R. Stan, and K. Skadron, "Hotspot 6.0: Validation, acceleration and extension," *University of Virginia, Tech. Rep*, 2015.

[169] A. Sridhar, A. Vincenzi, M. Ruggiero, T. Brunschwiler, and D. Atienza, "3d-ice: Fast compact transient thermal modeling for 3d ics with inter-tier liquid cooling,"

in *Proceedings of the International Conference on Computer-Aided Design*, IEEE Press, 2010, pp. 463–470.

[170] C. Zhu, Z. Gu, L. Shang, R. P. Dick, and R. Joseph, "Three-dimensional chip-multiprocessor run-time thermal management," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 8, pp. 1479–1492, 2008.

[171] J. G. Beu, M. C. Rosier, and T. M. Conte, "Manager-client pairing: A framework for implementing coherence hierarchies," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, ACM, 2011, pp. 226–236.

[172] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, IEEE, 2009, pp. 469–480.

[173] S. M. Hassan and S. Yalamanchili, "Understanding the impact of air and microfluidics cooling on performance of 3d stacked memory systems," in *Proceedings of the Second International Symposium on Memory Systems*, ACM, 2016, pp. 387–394.

[174] A. Mazouz, A. Laurent, B. Pradelle, and W. Jalby, "Evaluation of cpu frequency transition latency," *Computer Science-Research and Development*, vol. 29, no. 3-4, pp. 187–195, 2014.

[175] *Amd reveals fusion cpu+gpu, to challenge intel in laptops*, `https://arstechnica.com/information-technology/2010/02/amd-reveals-fusion-cpugpu-to-challege-intel-in-laptops/`.