COORDINATED POWER MANAGEMENT IN HETEROGENEOUS PROCESSORS

A Thesis
Presented to
The Academic Faculty

by

Indrani Paul

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

Georgia Institute of Technology
May 2015

COORDINATED POWER MANAGEMENT IN HETEROGENEOUS PROCESSORS

Approved by:

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

Dr. Karsten Schawn
School of Computer Science
Georgia Institute of Technology

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

Dr. Lizy K. John
Electrical and Computer Engineering
University of Texas, Austin

Dr. Hyesoon Kim
School of Computer Science
Georgia Institute of Technology

Date Approved: [Mar 23, 2015]

To my parents, husband, and children

ACKNOWLEDGEMENTS

This thesis would not have been possible without the assistance, guidance and constant inspiration of many individuals. First of all, I would like to express my deepest appreciation and thanks to my advisor Dr. Sudhakar Yalamanchili for his valuable guidance, insights, immense knowledge, patience and flexibility which made this a thoughtful and rewarding journey. His technical, editorial and general advice and mentoring was essential to the completion of this dissertation and has taught me innumerable lessons and insights on the workings of academic research in general. Although my doctoral studies have come to an end, I look forward to continue our relationship through many collaborations in future.

I would also like to thank my dissertation committee members Dr. Saibal Mukhopadhyay, Dr. Hyesoon Kim, Dr. Karsten Schawn and Dr. Lizy K. John for their support and guidance as this dissertation moved from a proposal to a complete study. Their suggestions and feedback helped me in shaping up my final thesis and develop the big picture behind all the individual pieces of work in this thesis.

I would like to thank my lab members at the Computer Architecture and Systems Laboratory at Georgia Tech, Minhaz Hassan, Nawaf Almoosa, and Jeffrey Young for our discussions and talks over the last few years. Although being a remote part-time student did not allow me to spend a lot of time in the lab, I am grateful for the support they have provided both technically and logistically.

Words cannot express my gratefulness to Advanced Micro Devices (AMD) for the immense support and flexibility it has provided to pursue and complete my PhD. Thanks

to Srilatha (Bobbie) Manne and Mike Schulte for convincing me to join AMD Research to work on the Exascale supercomputing project while pursuing my PhD. Special thanks to John Keaty for his strong support and encouragement and for being very flexible so that I could balance my work, PhD and family. Working at AMD Research has allowed me to amortize the considerable efforts required for doctoral research with fulfilling employment. Thanks to all my colleagues, Mike Schulte, Bobbie Manne, Manish Arora, Wei Huang, Wayne Burleson and Joseph Greathouse to name a few, for the endless interesting discussions and insights on a variety of topics ranging from power, architecture, microarchitecture, thermals, datacenters, software to general philosophies of life.

Most importantly, none of this would have been possible without the love, support, encouragement and patience from my family as I undertook this long journey. I am indebted to my parents who provided me the strong education foundation to begin with, motivated me to pursue PhD and taught me to believe in myself even through failures and setbacks. Thanks to my two beautiful daughters, one of which came along the way in the final leg of my PhD journey, who allowed me to put things in perspective through their bright little smiles, and made this journey meaningful. Finally, thanks to my best friend and my husband Sarat, who actually underwent this journey all along with me by my side, and sometimes was part of this even more than me. He not only encouraged me to get into the PhD program to fulfill my dreams as I was nervous about undertaking this big commitment while having a full-time family and a full-time job, he stayed committed to this process with me till the end by putting up with my countless work-hours and keeping me focused on my priorities when I had too many things to deal with.

TABLE OF CONTENTS

ACKNOWLEDGMENTS……...iv

LIST OF TABLES ……...xi

LIST OF FIGURES ……xii

SUMMARY ……xvi

INTRODUCTION..... 1

1.1 THESIS STATEMENT 3

1.2 THESIS CONTRIBUTIONS 4

1.3 THESIS ORGANIZATION..... 7

ORIGIN OF THE PROBLEM AND RELATED WORK 10

2.1 SHIFT IN COMPUTE PARADIGM..... 10

2.1.1 End of Dennard Scaling..... 10

2.1.2 Multi-core and Many-core Architectures..... 11

2.1.3 Path to Heterogeneous Computing 12

2.2 RESEARCH EFFORTS IN HETEROGENEOUS ARCHITECTURE 14

2.2.1 Workload Driven Optimizations..... 14

2.2.2 Integration Driven Optimizations 15

2.2.3 Physical Constraint Driven Optimizations..... 16

2.3 SUMMARY..... 20

BACKGROUND 21

3.1 GPU/APU PROGRAMMING MODEL 21

3.2 GPU HARDWARE DESIGN..... 22

3.3 HETEROGENEOUS ARCHITECTURE OVERVIEW..... 25

3.3.1 Trinity Accelerated Processing Unit..... 26

3.3.2 Tahiti Discrete GPU 27
3.4 POWER MANAGEMENT IN HETEROGENEOUS PROCESSORS 28
3.4.1 Trinity Power Management 30
3.4.2 Tahiti Power Management 32
3.4.3 Memory Power Management 33
3.5 SUMMARY 33

NEW MANAGEMENT CHALLENGES IN HETEROGENEOUS PROCESSORS 34

4.1 THERMAL COUPLING AND THERMAL SIGNATURES 35
4.1.1 Thermal Coupling 35
4.1.2 Thermal Signatures 38
4.1.3 Summary 41
4.2 PERFORMANCE COUPLING 42
4.2.1 Performance Coupling between Heterogeneous Compute Elements 43
4.2.2 Performance Coupling between Compute and Memory Elements 47
4.3 SUMMARY OF KEY MANAGEMENT CHALLENGES 62

THERMAL COUPLING MANAGEMENT 65

5.1 OVERVIEW 65
5.2 GREEDY VS. NEEDY POWER MANAGEMENT: INTERACTIONS BETWEEN THERMAL COUPLING AND PERFORMANCE COUPLING 67
5.3 RUN-TIME METRICS 73
5.4 COOPERATIVE BOOSTING (CB) 73
5.4.1 Structure 73
5.4.2 Algorithm 76
5.5 EXPERIMENTAL SET-UP 78
5.6 RESULTS 80

5.6.1 Performance 81
5.6.2 Thermal and Performance Coupling Analysis 83
5.6.3 Power and Energy 87
5.6.4 Summary 89
5.7 CONCLUSIONS 91

CPU-GPU PERFORMANCE COUPLING MANAGEMENT 92

6.1 OVERVIEW 92
6.2 SENSITIVITY ANALYSIS AND OPPORTUNITIES 94
6.2.1 Shared Resource Interference 96
6.2.2 Computation and Control Divergence 97
6.2.3 Performance Coupling and Kernel Sensitivity 98
6.3 RUN-TIME SYSTEM FOR ENERGY MANAGEMENT: METRICS, MODELS AND MANAGEMENT ALGORITHM 102
6.3.1 Frequency Sensitivity Correlation and Run-time Metrics 103
6.3.2 DynaCo: Coordinated Dynamic Energy Management Scheme 106
6.3.3 Summary 109
6.4 EXPERIMENTAL SET-UP 110
6.5 RESULTS 113
6.5.1 Performance, Power, and Energy 113
6.5.2 Performance Analysis and Power Shifting 117
6.6 CONCLUSIONS 121

GPU-MEMORY PERFORMANCE COUPLING MANAGEMENT 123

7.1 OVERVIEW 123
7.2 COMPUTE AND MEMORY BANDWIDTH SENSITIVITY ANALYSIS 126
7.2.1 Kernel Occupancy and Latency Hiding 126
7.2.2 Load Imbalance Due to Branch Divergence and Kernel Complexity 128

7.2.3 Architectural Clock Domains 129
7.3 COMPUTE AND MEMORY BANDWIDTH SENSITIVITY PREDICTORS 130
7.3.1 Performance Sensitivity Measurements 131
7.3.2 Performance Counter Correlation 132
7.3.3 Sensitivity Predictor Creation 132
7.4 HARMONIA: COORDINATED TWO-LEVEL POWER MANAGEMENT 133
7.4.1 Harmonia: Structure 135
7.4.2 Harmonia: Algorithm 135
7.5 EXPERIMENTAL SETUP 137
7.5.1 Metrics 139
7.6 RESULTS 140
7.6.1 Performance, Power, and Energy Efficiency 141
7.6.2 Adaptation Behavior 144
7.6.3 Summary of Key Insights 150
7.7 CONCLUSIONS 150
CONCLUSIONS AND FUTURE WORK 152
8.1 SUMMARY OF KEY CONTRIBUTIONS 152
8.1.1 Analysis and Abstractions of New Management Challenges 153
8.1.2 Characterization of Emerging Classes of Applications 153
8.1.3 Models and Run-time Metrics for Coordinated Management 154
8.1.4 Coordinated Management under Global Constraints 155
8.2 FUTURE WORK 155
8.2.1 Thermal and Performance Coupling Management in CPU-GPU-Memory Systems 156
8.2.2 Thermal and Performance Coupling Management in Datacenters 157

8.2.3 Workload Scheduling and Computation Balancing in Heterogeneous Systems 157

8.3 CONCLUSIONS 158

REFERENCES 159

VITA 168

LIST OF TABLES

Table 1: HW- and SW-managed DVFS states for the CPU compute units on the Trinity A8-455M APU. 78

Table 2: Summary of benchmarks used for CB evaluation. 79

Table 3: APU frequency sensitivity analysis of various performance metrics. 102

Table 4: CPU DVFS states for AMD A10-5800 APU. 110

Table 5: GPU DVFS states for AMD A10-5800 APU. 110

Table 6: Application datasets used for DynaCo evaluation. 111

Table 7: Performance counters and metrics for Harmonia 131

Table 8: Harmonia sensitivity model and parameters. 133

Table 9: GPU DVFS states for AMD HD7970 dGPU. 138

LIST OF FIGURES

Figure 1: Research efforts in heterogeneous CPU-GPU architecture. 13

Figure 2: Canonical GPU architecture. 23

Figure 3: Die shot of AMD Trinity APU [79] 25

Figure 4: Die Shot of Intel SandyBridge Processor. 26

Figure 5: Tahiti GPU architecture [80]. 28

Figure 6: Thermal entities in an AMD Trinity APU 30

Figure 7: RC network modeling thermal coupling in an APU. 31

Figure 8: Example of the impact of thermal coupling. 36

Figure 9: Thermal densities under CPU-centric (left) and GPU-centric (right) workloads. 38

Figure 10: Thermal time constant with a CPU centric workload. 40

Figure 11: Thermal time constant with a GPU centric workload. 40

Figure 12: Bulk-synchronous parallel programming model. 44

Figure 13: CPU-GPU execution dependency in a heterogeneous processor. 45

Figure 14: Example phase behavior in an exascale proxy application (Lulesh) 46

Figure 15: Power breakdown in a typical modern discrete GPU 47

Figure 16: Parallelism over time for two input graphs. 48

Figure 17: 3D plot for the performance scaling of a compute bound kernel with compute frequency and memory bandwidth. 51

Figure 18: Performance scaling of MaxFlops with available ops/byte in hardware. 52

Figure 19: 3D plot for the performance scaling of a memory bound kernel (Devicememory) with active compute units (CUs) and memory bandwidth. 54

Figure 20: Performance scaling of DeviceMemory with available ops/byte in hardware. 54

Figure 21: 3D plot for the performance scaling of a balanced kernel (Matrixmultiplciation) with compute frequency and memory bandwidth. 56

Figure 22: Performance scaling of LUD with available ops/byte in hardware..... 57

Figure 23: 3D plot for performance of a kernel that plateaus eventually and does not scale..... 58

Figure 24: 3D plot for performance of a kernel that peaks and then falls off as more resources are added. 59

Figure 25: *DeviceMemory*'s GPU card power across compute configurations at constant 264GB/s memory bandwidth. 61

Figure 26: *MaxFlops*'s GPU card power across memory bandwidth configurations at 32CUs and 1GHz compute frequency. 61

Figure 27: Chip-scale coordinated power management..... 64

Figure 28: Impact of CPU P-state limit on performance, GPU residency, and GPU utilization. 68

Figure 29: Thermal throttling in Needle with greedy boost algorithm and CPU P-state limiting. GPU-high, GPU-med, and GPU-low refer to high medium and low GPU operating frequencies, respectively.. 71

Figure 30: P-state limit effects on GPU memory bandwidth. 72

Figure 31: Block diagram of Cooperative Boosting (CB) framework. 74

Figure 32: Cooperative boosting algorithm. 75

Figure 33: Performance results with static limits and CB. 81

Figure 34: Thermal behavior of Binary Search with CB. 84

Figure 35: Viewdle performance analysis with CB. 85

Figure 36: Thermal throttling in Needle with CB. 87

Figure 37: Reduction in power for CB relative to baseline. 88

Figure 38: Energy-delay^2 product normalized to baseline. 89

Figure 39: Thermal profile of miniMD running on GPU. 95

Figure 40: Break-down of memory interference between CPU and GPU and corresponding CPU DVFS residency. 97

Figure 41: GPU frequency sensitivity to control divergence..... 98

Figure 42: Percent increase in kernel run-time due GPU DVFS changes relative to the baseline (BAPM). 99

Figure 43: Percent increase in kernel run-time due CPU DVFS changes relative to the baseline (BAPM). 99

Figure 44: DynaCo-1levelTh pseudo-code. 107

Figure 45: DynaCo-multilevelTh pseudo-code. 108

Figure 46: Performance impact of DynaCo 114

Figure 47: Phase variation within MATVEC 114

Figure 48: Energy efficiency with DynaCo 116

Figure 49: Power savings with DynaCo 116

Figure 50: GPU DVFS residency for DynaCo and baseline. 118

Figure 51: CPU DVFS residency with DynaCo-1levelTh. 119

Figure 52: CPU DVFS residency with DynaCo-multilevelTh. 120

Figure 53: Effects of VGPR-caused kernel occupancy limitation. 127

Figure 54: Impact on compute frequency sensitivity from load imbalance (branch divergence) and no. of instructions. 128

Figure 55: Impact of clock domains on compute frequency sensitivity for memory-intensive workloads. 130

Figure 56: Harmonia algorithm overview. 134

Figure 56: Performance sensitivity of Vector ALU Busy (VALUBusy) to number of active CUs (left) and memory bandwidth (right) 137

Figure 57: Performance, energy, energy-delay2 and energy-delay comparisons for LUD and DeviceMemory. Energy optimality leads to significant performance. 139

Figure 58: Overall combined performance and energy gain from Harmonia, using the ED2 metric. 141

Figure 59: Overall energy gain from Harmonia. 142

Figure 60: Overall performance from Harmonia. 143

Figure 61: Overall power savings from Harmonia. 144

Figure 62: Behavior of Graph500.BottomStepUp over time. 145

Figure 63: Memory bus frequency residency change as time progresses in Graph500.BottomStepUp..... 146

Figure 64: Residency of the hardware tunables in Graph500. 146

Figure 65: Relative GPU and memory power consumption. 147

Figure 65: Relative contributions of CG vs. FG in Harmonia. 149

SUMMARY

With the end of Dennard scaling, the scaling of device feature size by itself no longer guarantees sustaining the performance improvement predicted by Moore's Law. As industry moves to increasingly small feature sizes, performance scaling will become dominated by the physics of the computing environment and in particular by the transient behavior of interactions between power delivery, power management and thermal fields. Consequently, performance scaling must be improved by managing interactions between physical properties, which we refer to as processor physics, and system level performance metrics, thereby improving the overall efficiency of the system.

The industry shift towards heterogeneous computing is in large part motivated by energy efficiency. While such tightly coupled systems benefit from reduced latency and improved performance, they also give rise to new management challenges due to phenomena such as physical asymmetry in thermal and power signatures between the diverse elements and functional asymmetry in performance. Power-performance tradeoffs in heterogeneous processors are determined by coupled behaviors between major components due to the i) on-die integration, ii) programming model and the iii) processor physics. Towards this end, this thesis demonstrates the needs for coordinated management of functional and physical resources of a heterogeneous system across all major compute and memory elements. It shows that the interactions among performance, power delivery and different types of coupling phenomena are not an artifact of an architecture instance, but is fundamental to the operation of many core and heterogeneous architectures. Managing such coupling effects is a central focus of this dissertation. This awareness has the potential to exert significant influence over the design of future power and performance management algorithms.

The high-level contributions of this thesis are i) in-depth examination of characteristics and performance demands of emerging applications using hardware measurements and analysis from state-of-the-art heterogeneous processors and high-performance GPUs, ii) analysis of the effects of processor physics such as power and thermals on system level performance, iii) identification of a key set of run-time metrics that can be used to manage these effects, iv) development of online coordinated power management techniques for heterogeneous CPU-GPU-memory systems, and v) a detailed evaluation of the impact of these coordinated power management techniques on system level metrics.

CHAPTER 1

INTRODUCTION

Microprocessors have historically enjoyed exponential performance growth due to device-geometry scaling guided by Moore's Law, and supply voltage scaling governed by Dennard scaling rules to maintain affordable chip power envelopes. However, with the end of Dennard scaling, reducing device feature size by itself no longer guarantees sustaining the performance improvement predicted by Moore's Law. The diminishing voltage scaling margins coupled with the steady rise in leakage power at ultra-low voltages have drastically elevated power consumption, and risked operating in temperature regimes beyond the capability of existing cooling solutions. As industry moves to increasingly small feature sizes, performance scaling has become dominated by the physics of the computing environment and in particular by the transient behavior of interactions between power delivery, power management, and thermal fields. In particular, scaling trends point out that achievable performance capacity is limited by the thermal design power (TDP) rather than the computational capacity and frequency scaling. This so called Power Wall has presented serious challenges to the semiconductor industry, where the costs of developing new technology nodes are justified by their performance returns.

As W. Dally noted [29], performance scaling in the post-Dennard dark-silicon era will involve improving efficiency of power and energy usage instead of clock speed scaling. The modern industry shift towards heterogeneous computing is largely in part motivated by energy and power efficiencies. The trend towards heterogeneous processors continues with Accelerated Processing Units or APUs, which consist of multiple CPU and GPU processing elements integrated onto the same die. In the future we also expect to see heterogeneous systems with integrated CPU, GPU, and memory. While such tightly

coupled systems benefit from reduced latency and improved performance, they also give rise to new management challenges due to phenomena such as physical asymmetry in thermal and power signatures between the diverse elements and functional asymmetry in performance. These asymmetric relationships are sharing increasingly demanding power envelopes as we employ denser packaging technologies, e.g., 3D and silicon interposers. This leads to complex interactions between processor physics such as power delivery, thermal coupling (heat transfer from one element to others) and performance coupling (functional dependency among integrated multiple elements). *The fundamental problem is the manner in which these physical constraints are managed has a major influence on the eventual performance of the system. Power management solutions which do not consider and account for these interactions can have detrimental effects on system level metrics such as performance and energy efficiency.*

The preceding problem statement indicates that there are a variety of complex thermal and performance interactions between the diverse compute and memory elements in modern tightly coupled heterogeneous computing platforms that affect the power consumption and energy efficiency of the processor. For example, a common state-of-the-practice mechanism is to boost the frequencies of CPU or GPU cores to utilize all of the available thermal headroom for improving performance. Examples include Turbo Core from AMD and Turbo Boost from Intel [83][99]. However, due to the tight on-die physical integration of heterogeneous compute and memory elements, the power and thermal behaviors produce interactions resulting in t*hermal coupling*, where heat generated from one core raises the temperature (and leakage power) of adjacent cores and components. In addition, overall performance is a function of dependencies between the various elements and their fine-grain interactions. For example, the CPU may be feeding data streams to sustain computations on the GPU. If these multiple types of physical and functional dependencies are not carefully managed, the net effect is inefficient use of all of the available power and thermal headroom resulting in lower power efficiency and

performance. Past work has addressed dynamic thermal management in multi-core processors [14][26][95] and efficient power management in heterogeneous processors using emergent applications [68][75][115]. However, they do not directly address some of the new management challenges that arise in heterogeneous processors. Relatively few efforts to date have targeted understanding, characterizing, and managing the multi-physics and multi-scale (nanoseconds to milliseconds) interactions among power, thermals, and performance.

1.1 Thesis Statement

The new management challenges point towards the need for chip-scale coordinated power management to achieve performance scaling and energy efficiencies under a chip-wide TDP envelope and power budget. The efficiency with which we can manage these interactions will depend on our understanding of the relationships between the physical phenomena (power and temperature), functional behaviors (direct and indirect performance coupling) and their impact on system performance. Given the trends in heterogeneous processors and emerging applications, it is natural to ask: *How do processor physics and performance interact with respect to modern power management algorithms, and how should future power-management solutions account for these interactions?* This leads to the following **thesis statement:** *Power-performance tradeoffs in heterogeneous processors are determined by coupled behaviors between major components due to the i) on-die integration, ii) programming model, and the iii) processor physics. Thus, effective power management requires coordinated management across all major compute and memory components.*

This thesis demonstrates the needs for coordinated management of functional and physical resources of a heterogeneous system. It proposes abstractions with which to articulate and reason about how physical behaviors and various coupling effects impact system-level performance. It shows that this interaction is not an artifact of an architecture

instance, but is fundamental to the operation of many core and heterogeneous architectures. Managing such coupling effects is a central focus of this dissertation. This awareness has the potential to exert significant influence over the design of future power and performance management algorithms. To this end, this thesis i) examines characteristics and performance demands of emerging applications using hardware measurements and analysis from state-of-the-art heterogeneous processors and high-performance GPUs, ii) analyzes the effects of processor physics such as power and thermals on system level performance, iii) identifies a key set of run-time metrics that can be used to manage these effects, and iv) develops online coordinated power management techniques to optimize system level global metrics in heterogeneous CPU-GPU-memory processors.

1.2 Thesis Contributions

Specially, the high-level contributions of this dissertation research are as follows:

Characterization of New Management Challenges: Our studies in this research point out that various forms of coupling effects between major subsystems in a processor (CPU, GPU, memory) are beginning to dominate energy and performance efficiencies. This phenomenon will become more pronounced at future technology nodes. We develop an in-depth characterization and understanding of the impact of thermal coupling, power delivery, and performance coupling on heterogeneous processor performance. We demonstrate the practical constraints that physics places on effective architectures at future technology nodes and its effects on system-level performance. We also illustrate the distinct power and thermal signatures of the various integrated elements that exist in modern processors and which require tight coordination of their power states. This understanding forms the basis of the remaining work in this dissertation towards developing guiding principles for coordinated power management in tightly coupled heterogeneous processors, and the encapsulation of these principles in power management algorithms and their implementations.

Characterization of Emerging Classes of Applications: To acquire a thorough understanding of management challenges in modern processors, we perform an in-depth characterization of the emerging class of compute applications on integrated CPU-GPU processors as well as high performance discrete GPUs. The companion emergence of programming models such as CUDA, OpenACC, and OpenCL enable cooperative execution of both CPU and GPU towards an application's overall performance. We show that for applications that require cooperative execution of both CPU and GPU cores, state-of-the-art boost algorithms to utilize available thermal headroom can break down and degrade – rather than improve - performance. This is in contrast to other traditional applications such as graphics applications where the GPU is the obvious choice, or single-threaded, general-purpose applications which are suitable for the CPU. We also perform extensive scaling studies of these compute class of applications to assess their sensitivities to a variety of architectural attributes such as CPU, GPU, number of cores, frequency and memory bandwidth to understand their interactions with power, performance, and thermals.

Utilizing knowledge of workloads, the new management challenges and their interactions, the next part of the dissertation focuses on building guiding principles, online predictors and chip-scale coordinated power management algorithms to optimize system level metrics such as performance and energy efficiency under global constraints such as thermal limits and power budgets.

Maximizing Performance under Power and Thermal Constraints: This work examines the consequences of thermal coupling and its interaction with power management techniques in state-of-the-art heterogeneous processors consisting of a set of CPU and GPU cores. We experimentally demonstrate that for classes of applications that utilize both the CPU and the GPU, modern boost algorithms that greedily seek to convert thermal headroom into performance can interact with thermal coupling effects between the CPU and the GPU to degrade performance. We propose a dynamic power-management

approach called cooperative boosting (CB) to allocate power dynamically between CPU and GPU in a manner that balances thermal coupling against the needs of performance coupling to optimize performance under a given power and thermal constraint. Through real hardware-based measurements, we evaluate CB against a state-of-the-practice commercial boost algorithm and show that overall application performance and power savings increase by an average of 10% and 8% (up to 52% and 34%), respectively, resulting in average energy efficiency improvement of 25% (up to 76%) over a wide range of benchmarks [90].

Maximizing Energy Efficiency Under Performance Constraints: This part of the thesis examines energy management in a heterogeneous processor consisting of an integrated CPU-GPU for high-performance computing (HPC) applications. Energy management for HPC applications is challenged by their uncompromising performance requirements and complicated by the need for coordinating energy management across distinct core types – a new and less understood problem. We examine the intra-node CPU-GPU frequency sensitivity of HPC applications on tightly coupled CPU-GPU architectures as the first step in understanding power and performance optimization for a heterogeneous multi-node HPC system. The insights from this analysis form the basis of a coordinated energy management scheme, called DynaCo, for integrated CPU-GPU architectures. We implement DynaCo on a modern heterogeneous processor and compare its performance to a state-of-the-art power- and commercial performance-management algorithm. DynaCo improves measured average energy-delay squared (ED^2) product by up to 30% with less than 2% average performance loss across several Exascale and other HPC workloads [91][92] .

Interactions Between Compute and Memory System: In this work, we address the problem of efficiently managing the relative power demands of a high performance GPU and its memory system. We develop a management approach that tunes the hardware platform to maintain balance between the power dissipated in compute and memory access

across application phases under tight power and thermal budgets. We propose the use of online predictors that can capture performance sensitivities to hardware tunables. Using these sensitivity predictors we construct a two-level coordinated power management scheme, Harmonia, which configures the hardware to operate at its balance point by coordinating the hardware power states of the GPU (i.e. number of compute units and compute frequency) and memory system (i.e. memory bandwidth). Through hardware based measurements on a commodity GPU, we evaluate Harmonia against a state of the practice GPU commercial power management algorithm and show that Harmonia improves measured average energy-delay squared (ED^2) product by up to 36% with negligible performance loss across several representative HPC workloads [89].

1.3 Thesis Organization

The rest of the thesis is organized as follows:

CHAPTER 2 discusses the origin of the problem and the current landscape of research. It presents a literature survey describing technology trends and limitations that are guiding the transition to the heterogeneous computing era. It discusses research challenges and current research efforts in heterogeneous processors. It also presents current state of the art research on power and thermal management and modeling in both multi-core and heterogeneous processors. This section will conclude with highlighting important challenges and research directions for the future and how addressing such challenges can exert significant influence on future performance and power management designs.

CHAPTER 3 discusses the necessary background on heterogeneous and massively parallel high bandwidth GPU architectures along with their programming models and modern power management techniques to establish the generalization of the concepts discussed in this dissertation.

While there has been a large body of work addressing dynamic thermal and power management in homogeneous multi-core and heterogeneous processors (as reviewed in

CHAPTER 2), these efforts mostly focus on understanding individual physical phenomenon and steady state behaviors. They do not directly address some of the new management challenges that arise in heterogeneous processors. CHAPTER 4 presents an in-depth characterization of the new management challenges in modern processors along with extensive studies of emerging classes of applications. This chapter concludes with the highlights of the various forms of coupling effects between major subsystems in a processor (CPU, GPU, memory) that are beginning to dominate energy and performance efficiencies. Managing such coupling effects is the focus of the next few chapters.

CHAPTER 5 discusses a dynamic power management approach called Cooperative Boosting (CB) to coordinate power allocation between CPU and GPU in order to maximize performance under the thermal and power constraints of the processor package. This chapter presents a characterization of diverse compute elements in terms of their thermal signatures and the resulting thermal gradients between diverse compute elements. Particular attention is paid to how these physical phenomena affect thermal coupling and consequently impact system level performance by affecting power management algorithms. This understanding forms the basis for runtime power management using CB.

In CHAPTER 6 we examine the CPU-GPU frequency sensitivity of HPC applications on tightly coupled CPU-GPU architectures. We propose key run-time metrics that can be monitored to assess degree of direct and indirect dependencies between CPU and GPU for performance coupled operations and use those towards coordinating power states to maximize return on performance with additional power allocation. The insights from this analysis form the basis of a coordinated energy management scheme, called DynaCo, for integrated CPU-GPU architectures where we optimize for energy efficiency under performance constraints.

CHAPTER 7 extends the coordinated power management concepts to the memory system including global memory. In the future with the advent of High Bandwidth Memory (HBM), Hybrid Memory Cube (HMC) and other die stacking memory technologies,

heterogeneous architectures with integrated GPU and memory will emerge sharing the package or chip-level thermal design power envelope (TDP). Thus, we are concerned with effectively sharing the power envelope between the memory system and the GPU. In this work we discuss the notion of hardware balance and develop a two-level management approach that tunes the hardware platform to maintain balance between the power dissipated in compute and memory system across application phases. We demonstrate that such coordinated power sharing and shifting technique is imperative for future power constrained processors and can lead to significant improvement in energy efficiency under various effects of power, thermal and performance coupling.

Finally, CHAPTER 8 concludes with a summary of key research contributions of this dissertation and some potential directions for future research.

Collectively, the power management techniques proposed, demonstrated, and evaluated in the course of this thesis substantiates the importance of coordinated power management for the efficient operation of current and future heterogeneous processors. The techniques also expose general principles that govern the management of power and thermal limits. These general principles help place the management of power and thermal capacities on par with execution time and memory space as integral to the scaling of performance across future technology generations.

CHAPTER 2

ORIGIN OF THE PROBLEM AND RELATED WORK

This chapter starts by describing the origin of the compute problem and the shift in trends towards heterogeneous architectures. It provides an overview of the related research in the areas of heterogeneous computing and power management. The chapter concludes with identifying some gaps and needs in managing heterogeneous processors.

2.1 Shift in Compute Paradigm

In this section we give an overview of how the industry saw a shift in computing paradigm from single core to multi-core to heterogeneous architectures due to power, thermal and performance scaling challenges.

2.1.1 End of Dennard Scaling

Innovations in computing industry have been largely driven by improving performance of microprocessors through various advancements in semiconductor fabrication technology and enhancements in micro-architecture. In April 1965, Gordon Moore wrote an article for Electronics magazine titled "Cramming more components onto integrated circuits" [78]. He predicted that the number of transistors on a chip would double every 12 months to 18 months into the near future. This became known as what is popularly called "Moore's law". In 1974, Robert Dennard wrote a seminal paper describing MOSFET scaling rules for achieving simultaneous improvements in transistor density, switching speed and power dissipation [36]. Dennard's scaling rules observe that voltage and current should be proportional to the linear dimensions of a transistor, implying that

power consumption will be proportional to the area of a transistor. This property implies that shrunk MOSFETs will consume less power, and forms the basis of Moore's Law.

Moore's law combined with Dennard scaling became the guiding principle in semiconductor industry by providing designers a roadmap target for effective usage of transistors. This enabled faster and increasing clock speeds leading to higher dynamic power consumption. Increasing number of transistors also increased leakage power, power density and thermal dissipation. Voltage scaling limitations started to arise as lower threshold voltages increased sub-threshold leakage. Last but not the least, design complexity also increased with the emergence of superscalar pipelined architectures, sophisticated cache and branch predictors – all designed to improve single core performance. Clock speeds, power and thermal envelopes started to flatten out. However the demand for increased performance continued and this prompted a shift from the single core paradigm to a multi-core paradigm

2.1.2 Multi-core and Many-core Architectures

Multi-core architectures (CMP) from various companies such as Intel and AMD dominate the computing market today [16][99]. However multi-core has brought out several other challenges, such as high latency and increased power due to complex cache sharing logic, coherency protocols, longer interconnect and evolution of System on Chip (SoC). This puts a ceiling on power and performance efficiencies of general-purpose multi-core processors leading to dark silicon effects. It has limited the number of general purpose cores in a package without dissipating a significant amount of power that exceeds the capability of traditional cooling systems [40]. The International Technology Roadmap for Semiconductors indicates transistor density will continue to increase [55]. Smaller devices yield greater levels of integration, yet this poses an organizational question of how to achieve throughput and efficiency gains with more transistors while respecting constraints

in power, clock frequency, and interconnect delay. Hameed et al. [44] observe ASICs can be over 500x more energy efficient than CMPs for throughput-oriented tasks. A microcosm of the industry shift toward heterogeneity, their work describes a series of transformations applied to a soft-core processor transitioning it from a general-purpose programmable CMP toward an application-specific processor or accelerator.

Accelerators such as GPU, FPGA were initially targeted towards specific applications. For example: GPU was traditionally designed for graphics applications that exhibit high thread-level parallelism. These architectures have provided high degree of parallelism by integrating many simple cores to improve power efficiencies. However, general purpose applications with limited parallelism run poorly on such architectures because of the high overheads over PCI-e interconnects.

This brought the advent of heterogeneous architectures where a number of dissimilar cores are integrated onto the same die [65]. The heterogeneity can be at the functional level where multiple types of cores supporting different ISAs are integrated, such as the integrated CPU-GPU architecture, or it could be at the physical level where cores of same ISA with different physical properties are integrated onto the same die, such as cores supporting high and low power budgets etc.

In this dissertation and research, we focus on multi-ISA heterogeneous processor architectures with integrated CPU & GPU cores, so it is worth understanding such architectures in detail and the general landscape of research. The next few sections describe the advantages and research trends in such processors.

2.1.3 Path to Heterogeneous Computing

We are in the era of heterogeneous computing where the trend towards heterogeneous processors continues with Accelerated Processing Units or APUs, consisting of multiple CPU and GPU cores integrated onto the same die, sharing the same memory subsystem. Both AMD and Intel have examples of such heterogeneous processors.

This trend is enabled by the abundant availability of parallelism in GPUs and their high power efficiencies.

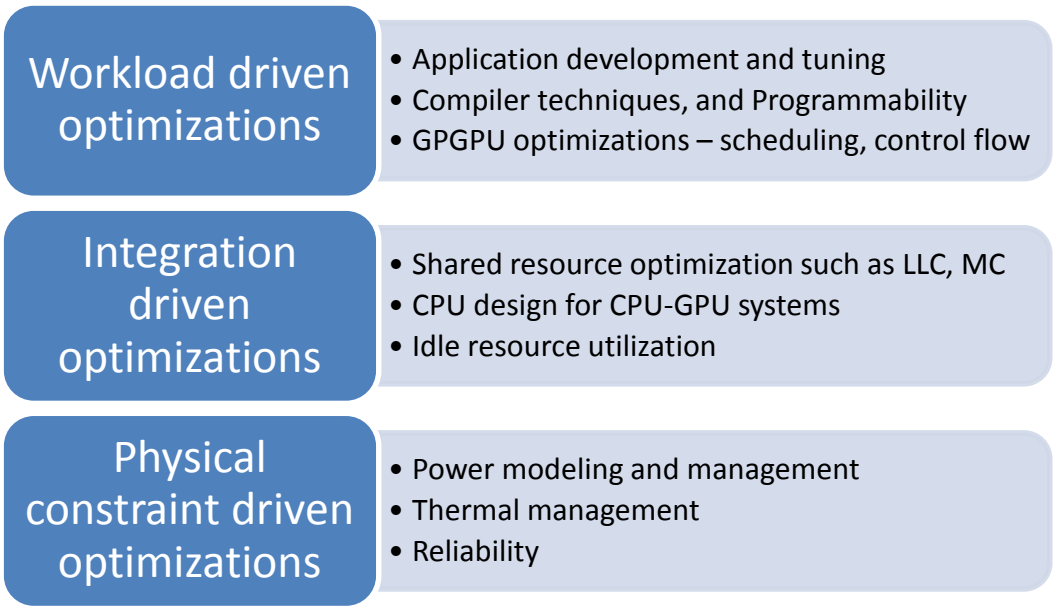


Figure 1: Research efforts in heterogeneous CPU-GPU architecture.

The tight integration of hardware in APUs and reduced latency, coupled with the companion emergence of programming models such as CUDA and OpenCL, facilitates effective and efficient computation on heterogeneous systems. This has also resulted in the emergence of a new class of applications that utilize both the CPU and the GPU computational capabilities to improve application performance [6]. Future processors may also have a unified memory address space between CPU and GPU, reducing latencies associated with data movement, and further improvements in programming models [13]. In the future we expect to see heterogeneous systems with integrated memory e.g., in 2.5D or 3D packages [74][88][119]. However such integrated architectures expose a new set of problems and new research questions. The next section will describe examples of such

research problems and the current state-of-the-art in addressing some of those research questions.

2.2 Research Efforts in Heterogeneous Architecture

Based on a literature survey, we have grouped the major research problems and efforts in heterogeneous architecture into three broad categories as shown in Figure 1. We believe these are some of the key driving factors enabling new research directions in heterogeneous architectures. In the next few sub-sections we will describe some of the current state of the art in each of these categories.

2.2.1 Workload Driven Optimizations

One of the challenges with GPUs is their poor performance for general purpose non-graphics applications. As CPU and GPU cores are integrated, a significant amount of research has been focused on making GPUs more amenable to mainstream computing. This spans everything from application development and tuning to compiler optimizations and programming models to GPGPU micro-architectural optimizations.

The tight integration between CPU and GPU has led to the emergence of a new class of workloads that were previously not possible to run on a GPU, such as irregular applications, and ones that make effective use of all compute resources available in the processor. Recent studies [6] have identified throughput-computing applications as an emergent class of future applications. In [20][54] the authors focus on the challenge of scaling irregular applications such as generalized reductions, irregular reductions and MapReduce using the CPU and GPU together in an integrated architecture. Che. et-al [18][19] present and characterize Rodinia, a benchmark suite for heterogeneous computing. Che et-al also presents Pannotia – a graph library suited for GPU architectures [17].

Integrated CPU-GPU pairings require two different programming models and distinct compilation tool chains to utilize the entire die, thus incurring a significant increase

in software complexity. A considerable body of ongoing work examines the feasibility of compiling applications for multicore CPUs, GPUs, and other accelerators. Consequently new programming models and APIs such as CUDA, OpenCL, and OpenACC have emerged. In [37][97] the authors propose a dynamic compilation framework and run-time support for heterogeneous systems.

Current GPUs suffer from two key shortcomings – loss of performance under control flow divergence and poor scheduling policies, both of which are important for mainstream computing applications. Hence apart from software level optimizations, there has been a lot of research in micro architectural optimizations to facilitate running general purpose workloads on the GPUs in integrated CPU-GPU systems. In [43][82], the authors explore mechanisms for efficient control flow execution on GPUs via dynamic warp or wavefront formation and large warp microarchitectures. GPUs typically use a round-robin warp scheduling policy giving equal priority to all concurrently executing wavefronts or collection of threads. This is beneficial due to high data locality across threads. However as we try to run more and more irregular general purpose applications on the GPU, round robin scheduling may not work due to memory divergence and branch divergence. So the authors of [82] propose two-level scheduling where a set of threads are scheduled based on some grouping that minimizes divergence.

2.2.2 Integration Driven Optimizations

One of the key steps in the development of next generation systems is a range of optimizations targeted towards the efficient use of integrated CPU-GPU and shared resources. In this arena, the CPU+GPU system is examined as a whole to better optimize its components. Rather than being designed for all workloads, Arora et-al in [4] redefine the role of the CPU in the heterogeneous era. They propose that the CPU core design be optimized for workloads that the GPGPU executes poorly such as serial code.

Similarly, there have been research studies to redesign shared resources such as the memory hierarchy and interconnects to account for the different demands of CPU-GPU

architectures and workloads. In [68], the authors propose a thread level parallelism aware last level cache management policy for CPU-GPU systems. Sharing the LLC between CPUs and GPUs brings new challenges due to the different characteristics of CPU and GPGPU applications. The authors demonstrate that effective caching and cache hit rates translate to GPU performance only when there is limited thread level parallelism. Jeong et-al [57] propose memory controller bandwidth allocation policies for CPU-GPU systems by dynamically partitioning off-chip memory based on expected deadlines and Quality of Service (QoS) provided to each of the components.

Another research optimization in CPU-GPU systems has targeted utilizing idle CPU or GPU resources. For example, COMPASS [120] proposes using idle GPU resources as programmable data prefetchers for CPU code execution. Correspondingly, in [123], the authors propose using a faster CPU to prefetch data for slower throughput oriented GPU cores. Similarly, workload partitioning schemes are being proposed such that both CPU and GPU compute resources are utilized by matching sections of the code to the better entity [75][98].

2.2.3 Physical Constraint Driven Optimizations

CPUs and GPUs in APUs share many classic hardware resources such as the memory hierarchy and interconnect. In addition, they also share physical resources such as power and thermal budget for the die. A significant body of work has dealt with the individual phenomenon of managing power or thermals in the context of homogeneous and heterogeneous processors. For example, power and thermal limits typically are addressed using a variety of individual techniques including thread scheduling and migration, dynamic voltage-frequency scaling (DVFS), power gating (PG), etc. The following subsections give an overview of research in both homogeneous processors as well as emerging heterogeneous architectures.

2.2.3.1 Power and Thermal Modeling

Several research works have proposed analytical and performance-counter based models for DVFS and power-performance models and predictions [10][27][50][60][121] in multi-core processors. A large body of work exists in power modeling for GPUs. In [48], Hong et al. develop an analytical power and performance model for a discrete GPU processor. In [72], J. Leng et al develop power models for GPUs that are configurable and capable of cycle-accurate predictions. A number of research efforts have also focused on developing computationally efficient, accurate thermal models for processor architectures [48][51][52]. There has also been a renewed interest in using machine learning and statistical frameworks to construct behavioral models for use in run-times, compilers, and even hardware to make scheduling decisions. Techniques to automate the construction of models of execution time for GPUs using basic machine learning are described in [58] and [63]. In [22][116], authors introduce the notion of machine balance and propose a performance and energy roof-line model that provide insights into performance bottlenecks and the operational intensity of an application in a particular platform. Such techniques focus on model construction and are distinct from model application (e.g., in making power-management decisions).

2.2.3.2 Power Management

There have been many research efforts that have employed DVFS to improve efficiency under a certain constraint such as power or performance. For example: C. Hsu et al. apply DVFS techniques to improve power efficiency through CPU-boundedness detection [49]. J. Li et al. [73] propose a run-time voltage/frequency and core-scaling scheduling algorithm that minimizes the power consumption of general-purpose chip multi-processors within a performance constraint. A. Mclaughlin et-al explored optimizing performance of graph algorithms under power constraints in a GPU [77]. J. Lee et al. [69] analyze throughput improvement of power-constrained multi-core processors by using power-gating and DVFS techniques. Recently, there has been a significant interest in the

power management of GPUs. Lee et al. [70] propose DVFS techniques to maximize performance within a power budget for discrete GPUs. Wang et al. [115] propose a workload-partitioning mechanism between the CPU and GPU to utilize the overall chip power budget to improve throughput. An overview of different power efficiency techniques is presented in [60][96]. Arora et-al investigate power gating mechanisms for CPUs in the context of emerging CPU-GPU applications in a heterogeneous architecture [5]. Several research works have also proposed compiler-driven [122] and control-theoretic [113] approaches for managing power.

There are also many existing studies investigating main memory power management in CPU-memory systems [31][32][33]. For example,[31] applies DVFS to memory controllers and DFS to memory channels and DRAM devices, using a simulation framework. Authors in [32] propose DVFS for main memory and presents evaluations on real hardware. In [33], Deng et-al allocates a power cap to main memory with the aid of a runtime DRAM power model. A few prior works also look at coordinated power management between CPU and main memory. In [34], authors propose runtime techniques to minimize total system energy within a performance constraint for a multi-CPU system, using a simulation framework. Another work [35] tries to reduce system energy by applying coordinated DFVS across multiple memory controllers (MCs), based on the observation of skewed traffic across MCs in multicore server processors. Authors in [41] look at policies for power shifting between CPU and memory, without explicitly investigating performance dependency between CPU and memory. Regarding GPU-memory systems, [47] develops an analytic performance model for GPUs around memory and thread-level parallelism, without power/energy considerations.

In the HPC area as well, there has been considerable power management focused research work on DVFS for multi-core processors [87][100]. Pakin et al. [87] characterized power usage on production supercomputers using production workloads. Laros et al. [67] performed extensive large-scale analysis of power and performance requirements for

scientific applications in supercomputers based on static tuning of applications through DVFS, core, and bandwidth scaling. In [100] Rountree et al. explored energy-performance trade-offs for HPC applications bottlenecked by memory and communication. In [101] and [102], Rountree et al. investigated speeding up the critical path of an application in a multi-processor cluster using slack-prediction and leading-load techniques, respectively. In [9] Balaprakash et al. described exascale workload characteristics and created a statistical model to extrapolate application characteristics as a function of problem size.

2.2.3.3 Thermal Management

Similarly, a large amount of research exists on thermal management in homogenous multi-core processor that evolved originally to prevent harmful thermal capacity violations of peak temperature. Consequently, architectural efforts have focused first on preventing unwanted thermal excursions and have since quickly evolved to balancing the system-level performance impact of such management techniques [14][79]. The range of techniques include i) activity migration [21], ii) power reduction by various forms of throttling [45][118], iii) feedback control [105][106][107][124], or iv) a combination of techniques to balance performance loss against thermal management. These techniques are concerned primarily with managing peak temperatures. That philosophy continues with the advent of multi-core architectures [38] through run-time techniques such as heat-and-run [94] or a combination of design- and run-time techniques [79], while more recent work considers the impact of reliability [24] and relationships to process variation [66]. The management issues naturally evolve to 3D architectures, which exacerbate the thermal management problem [25][110]. Architectural techniques are complemented by efforts in the system software community primarily through managing power dissipation using various scheduling techniques [23][53]. Authors in [104] study thermal management for GPUs. Some recent works also include efforts to couple thermal management, cooling management, and power management [7][8][95]. The preceding are just a few examples of

the extensive knowledge base developed in the past decade or so, and [26][60] provide a thorough overview of the techniques.

However, none of this research focuses on interactions between power, thermal, power management and system performance in a heterogeneous architecture. These interactions will become more pronounced at future technology nodes and are amplified by heterogeneous architectures with diverse elements running general purpose compute applications. Prior works do not address the consequences of tightly coupled heterogeneous systems and how to manage them efficiently. As W. Dally noted, performance scaling in the post-Dennard era will involve improving efficiency of power and energy usage instead of scaling clock speeds [29]

2.3 Summary

The preceding challenges provide motivation for further research to understand and analyze the relationship among performance, power management, thermal capacity, and thermal interactions between the multiple elements in a heterogeneous architecture for the emerging class of general purpose compute applications. It inspires new research directions to manage "on-die integration driven" physical constraints and system performance among all the components in a heterogeneous processor. We believe that this awareness has the potential to exert significant influence over the design of future power-performance management algorithms. The next chapters will provide background information followed by describing the power and thermal behaviors in heterogeneous processors in more detail leading to the proposed research and thesis statement.

CHAPTER 3

BACKGROUND

We are in the era of heterogeneous computing where the trend towards heterogeneous processors continues with accelerated processing units or APUs, which consist of multiple CPU and GPU processing elements, sometimes sharing a unified memory address space, integrated onto the same die. The tight integration of hardware on APUs, coupled with the companion emergence of programming models such as CUDA and OpenCL, facilitates effective and efficient computation on heterogeneous systems. In the future we expect to see heterogeneous systems with massive parallelism and high bandwidth integrated memory e.g., in 2.5D or 3D packages [39][56][74][93][119]. This chapter provides the necessary background on heterogeneous and massively parallel high bandwidth GPU architectures along with their programming models to establish a foundation for the research in this dissertation.

3.1 GPU/APU Programming Model

Post-Dennard performance scaling is achieved by improving efficiency along the entire stack from hardware through programming models and applications. To fully exploit heterogeneity and the available hardware resources, applications must be redesigned, requiring the need for new programming models. In this context, the bulk-synchronous parallel (BSP) programming model has been uniformly adopted for massively parallel GPU architectures that get rid of global cache coherence and memory level consistencies in lieu of coarse and fine-grained thread level parallelism.

Since the original formulation of the BSP model by Valiant [112], several industry initiatives have adopted variants for general purpose programming on GPUs. CUDA [85] and OpenCL [86] are first such examples. This thesis utilizes the OpenCL terminology

although the concepts are applicable to analogous elements of the CUDA programming model. Both CUDA and OpenCL express programs in terms of a series of compute kernels. A host program launches a compute kernel consisting of a 2D/3D grid of work groups (thread blocks) where each workgroup is comprised of a block of work-items (threads). Work-groups share a block of local data storage (LDS) and vector and scalar general purpose registers (VGPR and SGPR). Work-items within a workgroup are also grouped into sets of threads called wavefronts (warps) operating in lock step relative to each other. For example, 64 threads execute the same instruction on different data using the 64 ALUs on an AMD architecture.

The BSP model requires global synchronization among all threads leading to some threads waiting for others to finish. Programming models like OpenCL and CUDA coupled with emerging applications allow for balancing computation among the different compute elements in the heterogeneous CPU-GPU architecture to take advantage of the different performance-power efficiency points of the CPU and the GPU.

Recently AMD (APUs) [83], Intel (Sandy Bridge) [99], and ARM (MALI) [108] have released solutions that integrate general purpose programmable GPUs together with CPUs on the same die. In this computing model, the CPU and GPU share memory and a common address space. Even though they are different microarchitectures they all fundamentally utilize the BSP style of communication and are comparable e to each other. Next sub-section details such architectures.

3.2 GPU Hardware Design

GPUs gain much of their performance from running many parallel compute units (CUs) at a moderate frequency. Figure 2 shows a canonical GPU architecture. They typically hide memory latency be executing many parallel threads with access to high bandwidth memory systems. The GPU itself has many graphics-specific circuits that we do not illustrate, because general purpose compute codes primarily use the programmable

CUs. There are many of these parallel CUs or streaming processing clusters in modern GPUs, and each contains some number of arithmetic logic units (ALUs), which run in a SIMD fashion. CUs on GPUs from AMD, for instance, have 64 parallel ALUs [76]. Similarly, NVIDIA's “Fermi”-class flagship GPU contains four Graphics Processor Clusters (analogous to CUs), each partitioned into four SMs (Streaming Multiprocessors) [84]. If two threads within a wavefront need to execute different instructions, such as taking different branch directions, they use predicated execution and serialize the wavefront through both paths.

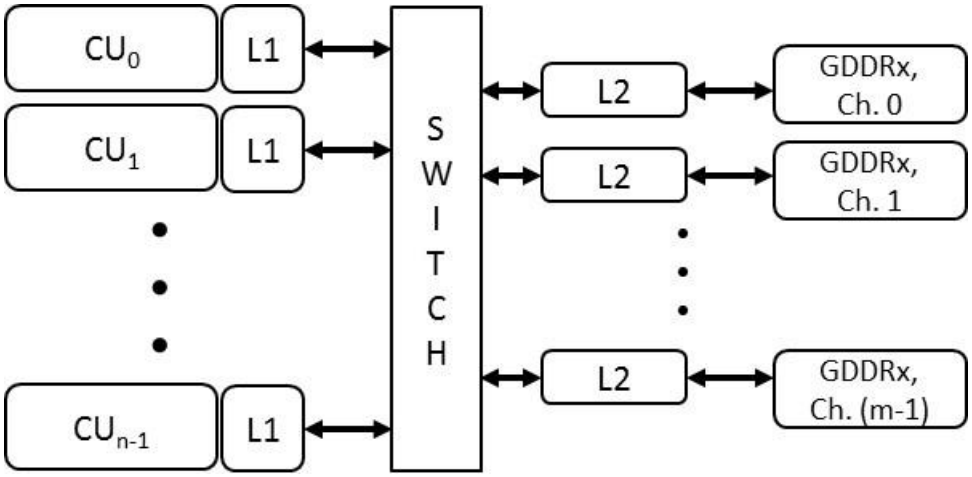


Figure 2: Canonical GPU architecture.

Rather than using out-of-order execution logic like CPUs to avoid delays due to long-latency operations, GPUs use more threads. A wavefront is the basic unit of hardware scheduling. However, there are resources that are shared among work-groups. These resource demands in part govern the number of in-flight wavefronts and hence concurrent execution. Many wavefronts are assigned to a single CU, which uses fine-grained multithreading to mask delays. GPUs have a hardware-managed cache hierarchy between

the CUs and memory system, which is used to hold commonly used read-write data. Each CU has some "private" cache, which is actually shared between all of the threads running on that CU. The L2 cache sits near the memory controllers and is shared across all of the CUs.

The common design described above can be configured in a variety of ways to meet market demands. CU count, L2 size, and number of memory channels can be varied in order to reduce area and costs. The frequency of the chip and the DRAM can be varied in order to control power usage, maintain thermal limits, and hit performance targets. It is worth noting that in a discrete graphics card (dGPU), the GPU is connected to the host CPU through a low bandwidth PCI-e bus, which is why high performance massively parallel discrete GPUs have access to their own high bandwidth memory system co-located with the GPU chip on the same dGPU card. However, heterogeneous processors, such as APUs, put both a CPU and a GPU in the same chip. They replace the PCIe connection with custom high-bandwidth interconnects and let the GPU use the system's DRAM. In future we expect the system DRAM to be integrated inside the same package along with CPU and GPU in an APU. The exact number of CUs and parallel processors that make up the GPU differ between an APU and a dGPU and it typically depends on a variety factors such as the peak compute to memory ratio target for that architecture, the target application space, power and thermal budgets, design and packaging constraints etc.

In this thesis we use measurements from state-of-the-art modern heterogeneous processors and discrete GPUs to analyze interactions between processor physics and performance and evaluate impact of the propose power management techniques. Although there are many simulators and analytic frameworks that simulate power, performance and thermals for different microarchitectures, simulators cannot really model and capture the complex relationships between time-varying real-world workloads, power delivery, thermal transients and steady state fields, and performance with high fidelity. Next, we provide an overview of the specific architectures - AMD Trinity APU and AMD HD7970

dGPU - used in this dissertation. Although we used specific hardware architectures as experimental basis, the core concepts, observations and the insights from this dissertation are equally applicable to other types of architectures following the bulk-synchronous parallel programming model described in Section xxx.

3.3 Heterogeneous Architecture Overview

Systems containing GPUs are, by definition, heterogeneous. The GPU has to communicate with the CPU via low latency on-chip interconnect, such as in APU, or over PCIe, such as in discrete GPU. Access to the memory interface (shared memory or dedicated) brings another dimension to this heterogeneity.

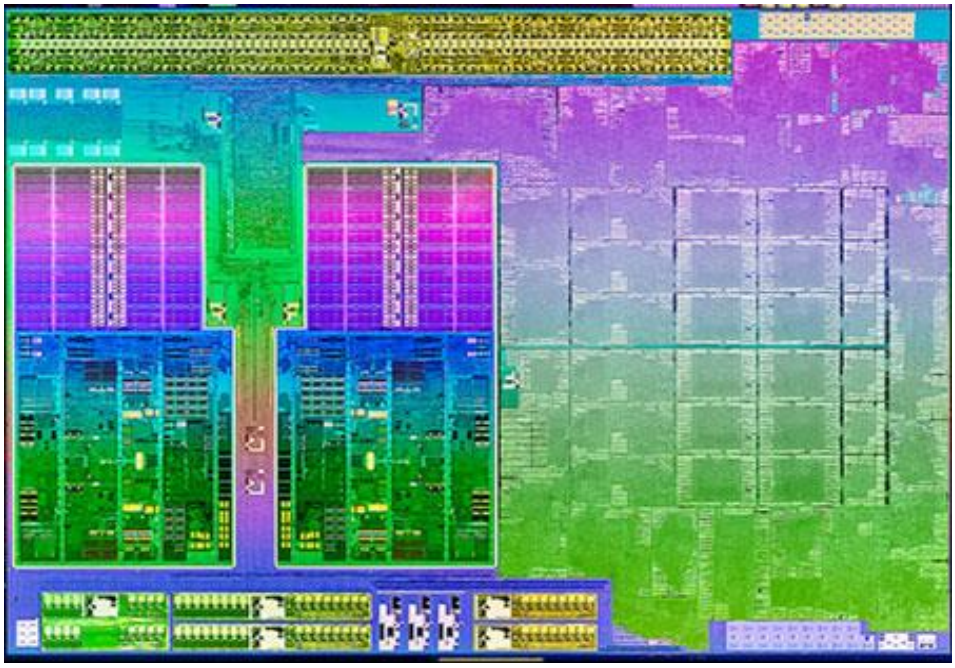


Figure 3: Die shot of AMD Trinity APU [83].

Heterogeneous CPU-GPU processors such as Intel's SandyBridge [99] and AMD's Trinity APU [83] consist of one or more CPU cores in combination with a GPU core. Die

shots of the AMD Trinity and the Intel SandyBridge processors are shown in Figure 3 and Figure 4, respectively. They both contain a number of CPU x86 cores paired with caches and a GPU in addition to miscellaneous other logic such as a memory controller, power controller, and fixed function units such as video encode and decode. The key point to note about both systems is that there are many compute units (CU) in the same package and they are in close proximity to each other sharing the same power and thermal budgets as well as the heat sink solution.

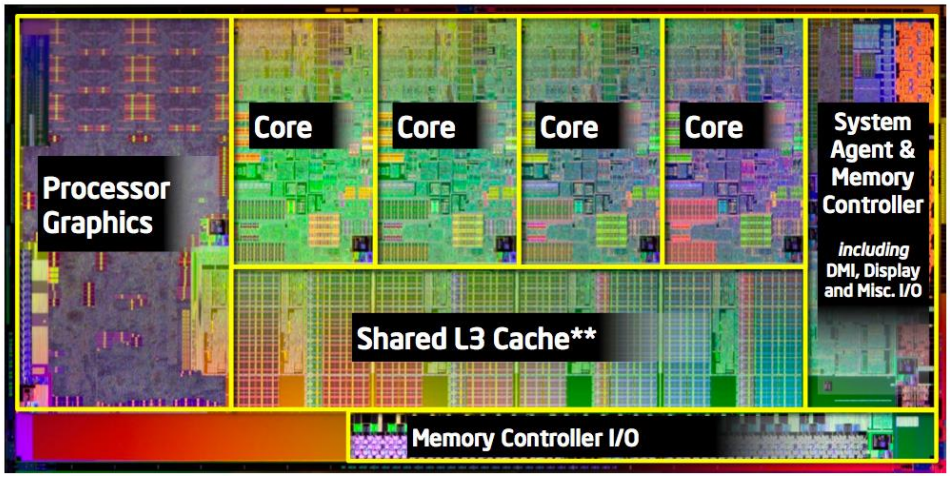


Figure 4: Die Shot of Intel SandyBridge Processor.

3.3.1 Trinity Accelerated Processing Unit

The Trinity APU in Figure 3 contains two PileDriver modules or CPU compute units (CUs), AMD Radeon™ GPU cores, and other logic components such as a NorthBridge and a Unified Video Decoder (UVD). Each CPU module is composed of two out-of-order cores that share the front-end and floating-point units. In addition, each module is paired with a 2MB L2 cache that is shared between the cores. The GPU consists of 384 AMD Radeon cores, each capable of one single-precision fused multiply-add

computation (FMAC) operation per cycle. The GPU is organized as six SIMD compute units (CUs), each containing 16 processing units that are four-way VLIW. The memory controller is shared between the CPU and the GPU.

On the CPU side, there are multiple DVFS states, some of which are software-visible and can be managed either by the OS through the Advanced Configuration and Power Interface (ACPI) specification [1] or the hardware. A few of the DVFS states are only visible to and managed by the hardware – in other words, entrance to and exit from those states are managed only by hardware. On Trinity, DVFS states can be assigned per CPU compute module; however, because the CUs share a voltage plane, the voltage across all CUs is set by the maximum-frequency CU.

The GPU has an independent power plane whose voltage and frequency are controlled independently. However, unlike the CPU, the GPU does not have DVFS states visible to software. Entrance to and exit from these states are managed entirely in hardware with some involvement from the GPU driver.

3.3.2 Tahiti Discrete GPU

The AMD Radeon HD 7970 system is one of the "Southern Island" families of AMD graphics processors, and is illustrated in Figure 5. It features the AMD Graphics Core Next (GCN) architecture and is paired with 3GB of GDDR5 memory organized using a set of six 64-bit dual channel memory controllers (MC) with maximum bandwidth of 264 GB/sec. The processor contains up to 32 compute units or CUs with four SIMD vector units in each CU. There are 16 processing elements (PE) per vector unit, called ALUs, resulting in a single precision FMAC compute throughput of about 4096GFLOPS. Each CU contains a single instruction cache, a scalar data cache, a 16-KB L1 data cache and a 64-KB local data share (LDS) or software managed scratchpad. All CUs share a single 768-KB L2 cache. All CUs in the GPU share a common frequency and voltage plane.

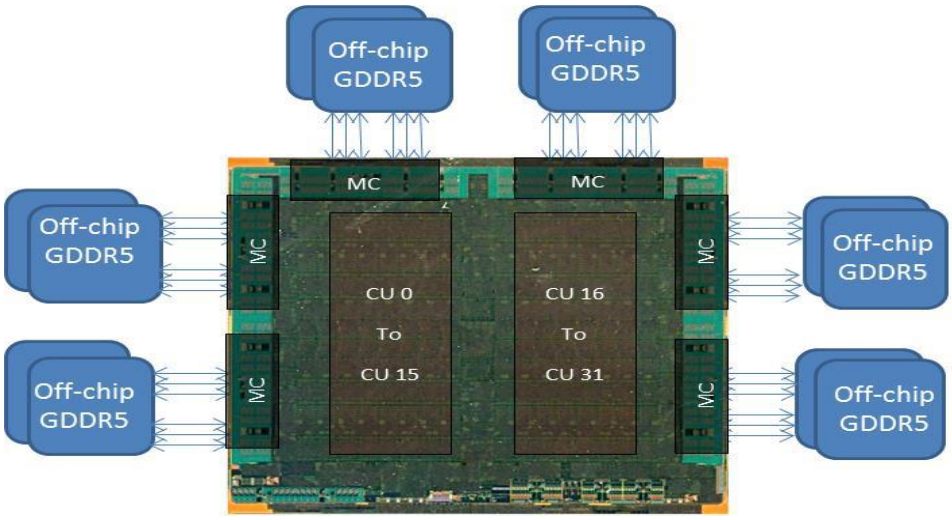


Figure 5: Tahiti GPU architecture [84].

3.4 Power Management in Heterogeneous Processors

This section provides an overview of modern power management techniques on current state of the art heterogeneous processors. In an APU, although the CPUs and the GPU are on independent power planes, they share the same die and system power supply, and hence share the same power and thermal headroom. These heterogeneous processors use a sophisticated power-monitoring and management technology, referred to as Turbo Boost on Sandy Bridge and AMD Turbo CORE on Trinity, to determine the dynamic voltage and frequency scaling (DVFS) states for the CPU and GPU to optimize performance given power and thermal constraints. These technologies use some combination of measured and approximated power and/or temperature values to monitor and guide the power-management algorithm. In addition, unused resources can be

dynamically power gated by the hardware to shift power credits to the active regions of the die.

Typically processor's power consumption is limited by a system's ability to both deliver power to the device and cool it by removing the heat it generates. Processor manufacturers provide system builders with a Thermal Design Power (TDP) figure for their products to allow them to design their systems appropriately. This represents the maximum power draw for reliable operation. There are many factors which can affect TDP, including workloads, voltage and frequency, leakage and ambient temperature. The maximum software-visible voltage and frequency for the processor is defined using a combination of heavy activity and worst-case operating conditions. This corresponds to the thermal design point (TDP) power for the chip. However, across time-varying workloads it is common for the processor to operate well below the TDP power and, therefore, well below the peak temperature allowed. The difference between the current and peak temperatures is the thermal headroom. Thermal headroom can be utilized by permitting the CPU and/or the GPU components to exceed the maximum frequency and TDP power for short periods.

Likewise, in a heterogeneous platform with a dGPU, similar thermal-headroom based power management techniques can be employed to maximize performance under the physical constraints of the chip. Thermal headroom can be used towards boosting the GPU compute units to improve performance. In addition, the off-chip DRAM and compute processor share the same overall board level power and thermal budget. In future with the advent of High Bandwidth Memory (HBM), Hybrid Memory Cube (HMC) and other die stacking memory technologies [93][39][56], heterogeneous architectures with integrated GPU and memory will emerge [131] sharing the package or chip-level thermal design power envelope (TDP). Thermal headroom driven management techniques can be utilized for any of these architectures to improve performance and power efficiency.

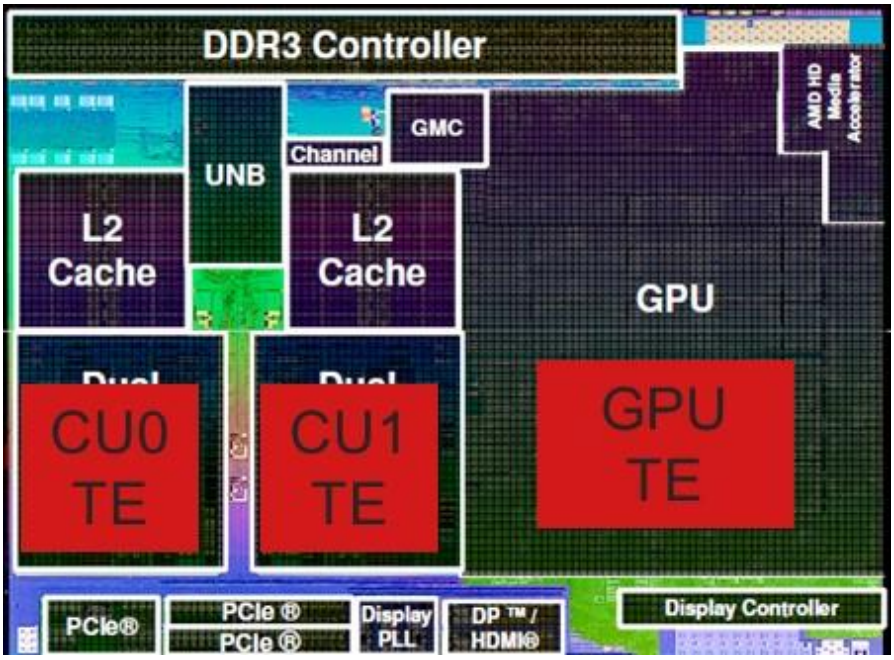


Figure 6: Thermal entities in an AMD Trinity APU.

Power-management algorithms differ in how the timing, extent, and duration of the boosted and non-boosted operation are determined. However, regardless of the specific implementation, it is safe to say that modern processors dynamically manage power across multiple components of the die under fixed power and thermal constraints. For instance, both Sandy Bridge and Trinity processors dynamically manage power allocation across the CPUs and the GPU under a pre-set thermal and power limit. Next we describe the power management algorithms in AMD Trinity APU and AMD Tahiti dGPU used in this dissertation. The concepts and methodologies are applicable to other heterogeneous processors using power and thermal based management.

3.4.1 Trinity Power Management

AMD's Turbo CORE technology uses the Bidirectional Application Power Management, or BAPM, algorithm [83] to manage to thermal limits. BAPM controls the power allocated to each thermal entity (TE) in the processor, as shown in Figure 6. TEs are defined to be any sub-component of the processor that interfaces with BAPM to report its

power consumption and receive its power limits. Once BAPM has assigned power limits, each TE manages its own frequency and voltage to fit within that limit. For the Trinity system evaluated in this paper, BAPM interfaces with the two CPU compute units (CU0 and CU1) and the GPU. At regular time intervals, the BAPM algorithm does the following:

1) Calculates a digital estimate of power consumption for each TE;
2) Converts the power estimates into temperature estimates for each TE; and,
3) Assigns new power limits to each TE based on the temperature estimates.

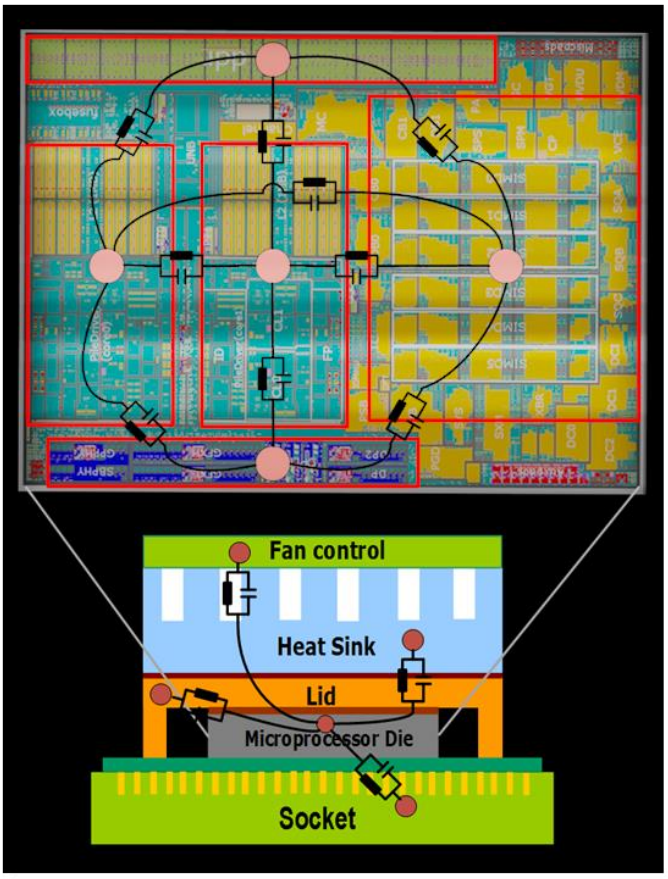


Figure 7: RC network modeling thermal coupling in an APU.

To estimate the temperature across the die, the chip is divided into regions in which local power and thermal properties are calculated and transfer coefficients (represented as

an RC network) are utilized to compute heat transfer among the thermal regions, substrate, and package. Temperatures within each region are computed using numerical methods. Figure 7 shows the RC network model for thermal coupling used in the BAPM algorithm.

The BAPM algorithm is optimized for a fair and balanced sharing of power between the TEs. When thermal headroom is available, BAPM proportionally allocates power to each TE using a pre-set static distribution weight derived using empirical analysis reflecting the individual thermal properties of each TE (i.e., its thermal behavior for a given power). Such static allocation is an effective choice in the absence of dynamic feedback from application execution. When the core reaches its thermal limit, BAPM reduces the allocation of power to all TEs in the system. As a general-purpose state-of-the-practice controller, BAPM is designed to provide reasonable performance improvements without any significant outliers for today's applications.

3.4.2 Tahiti Power Management

The AMD Radeon HD7970, code-named Tahiti, uses AMD PowerTune technology [3] to optimize performance for TDP-constrained scenarios. This enables the GPU to automatically adjust power between its DVFS states, based on power and thermal headroom availability. It also allows for boosting to a higher frequency state when there is headroom. The AMD PowerTune algorithm embedded in the GPU hardware calculates the compute frequency based on an internal assessment of the runtime power draw. When the GPU is in the highest activity or power state and not exceeding the power and thermal limit, it will remain in the highest power state for maximum performance. In the case where AMD PowerTune calculates that the GPU is exceeding power and thermal constraints of the die, the power is dynamically reduced in a gradual manner by reducing the frequency. This works well for managing compute power. However, very little power management exists for off-chip memory which shares the same board level power budget on current GPUs, and same on-die power and thermal envelope in future 3D die-stacking GPUs [74][131].

3.4.3 Memory Power Management

DRAM memory is a significant power consumer in a platform – whether it's the global system memory accessible by both CPU and GPU in an APU or the on-board high-bandwidth memory present in the dGPU card that is accessible by the GPU. One way to change memory power is by dynamically adjusting the memory bus frequency, which controls the memory controller, DDR PHY and the DRAM devices. DRAM power can be further broken down into background, activation/pre-charge, read-write and termination power. Changing memory bus frequency has a different impact on each of these components. Lowering bus frequency lowers background and PLL power, as well as memory controller and PHY power. On the other hand, it can increase read/write and termination energy due to longer interval between array accesses. Further, if frequency is slowed down to a point where memory latency can no longer be hidden through thread level parallelism in the GPU, it can hurt performance significantly and increase the overall energy consumption of the platform. In this dissertation, managing memory power refers to managing the frequency of the memory channel i.e., the bus interface.

3.5 Summary

This chapter presented the necessary background information to support the rest of the thesis. We described modern bulk-synchronous type parallel programming models (BSP) and explained the generalization and applicability of the core concepts and insights from this dissertation across multiple architectures implementing this BSP model. We also provided an overview of the current state-of-the-practice boost algorithms which serve as a reference point of comparison for the novel power management techniques proposed and evaluated in this dissertation.

CHAPTER 4

NEW MANAGEMENT CHALLENGES IN HETEROGENEOUS PROCESSORS

Following the end of Dennard scaling, the major system challenge facing the industry is to sustain performance scaling with Moore's Law while preparing for the transition to post-CMOS technologies. From [29], system performance can be represented by $Perf\left(\frac{ops}{s}\right) = Power(W) * Efficiency\ (\frac{ops}{joule})$. Since power densities remain roughly flat, performance scaling must derive from commensurate improvements in efficiency [29]. This has two important consequences. First, customization in the form of heterogeneity, technology diversity, and architecture asymmetry is the norm. Second, scaling performance is achieved by scaling energy efficiency. Thus heterogeneity and energy efficiency must be concurrent goals.

In heterogeneous processors, multiple components are tightly integrated onto the same die. While this results in the sharing of many classical functional resources such as the memory hierarchy and interconnect, it also results in the sharing of physical resources such as power and thermals by the different components in the package. The different components consume power and thermal capacities in different ways at different times and their consumption is a time-varying function of workload characteristics, architectural and micro-architectural constitution of the various components, as well as the die floor plan. This leads to new and complex multi-function, multi-physics and multi-scale management interactions and challenges which must be understood in order to improve performance and power efficiency of future processors.

This chapter presents an in-depth characterization of the new management challenges arising from the tight integration of different compute elements and memory in

heterogeneous processors. First, it elaborates and explains physical phenomena such as *thermal coupling* that lead to loss of performance and loss of efficiency in heterogeneous processors in Section 4.1. Second, in Section 4.2.1, it characterizes the functional dependency, i.e. *performance coupling* of the different compute cores, e.g., CPU and GPU, in a heterogeneous processor and demonstrates that tight coordination is indeed needed at run-time to manage their power states. Lastly, Section 4.2.2 describes one of the key management challenges that arises with the advent of on-package DRAM integration e.g., die stacks and EDRAM, where compute and memory share the same power and thermal envelope. We explain *performance coupling* between compute and memory and the notion of a *hardware balance* point and show that imbalances in hardware configuration (i.e. compute and memory bandwidth) can lead to significant imbalances in cost (power consumption) vs. benefits (performance and power efficiency) in future systems.

4.1 Thermal Coupling and Thermal Signatures

This subsection presents an analysis of thermal coupling effects and attempts to articulate concepts for describing the thermal behavior of individual micro-architectural components.

4.1.1 Thermal Coupling

Due to the tight on-die physical integration of the CPU and the GPU, heat exchange occurs between the heterogeneous cores. In this research, this physical phenomenon is referred to as *thermal coupling*, where heat generated by one core raises the temperature (and leakage power) of adjacent cores and components.

Figure 8 illustrates the effects of thermal coupling on overall performance using an AMD A8-4555M Trinity APU comprised of two dual-core CPU compute units (CU0 and CU1) and one six-SIMD unit GPU (Figure 3). More details of the APU are provided in Chapter 3.3.1. The left-side y-axis shows measured power relative to time zero, provided

by on-chip real-time measurement capability. The right-side y-axis shows the peak die temperature normalized to the maximum junction temperature.

Initially, the GPU operates at its highest frequency and the CUs are fixed at a low-frequency, low-power state. After the GPU temperature stabilizes, at around 230 seconds, additional power is allocated to CU0 and CU1 due to availability of significant thermal headroom and they enter a higher-power DVFS state (this is automatically performed by the existing chip power management unit). Not only do the CUs increase their power dissipation, but due to thermal coupling and the impact of heat on leakage power, the GPU power also rises. The increase in system power causes an increase in peak temperature and eventually triggers temperature-based throttling of both the CUs and the GPU at around 267 seconds to maintain a steady-state peak junction temperature. This results in a net performance loss for the overall application as the GPU is now running at a lower effective frequency than before and the application's performance is dependent on the GPU

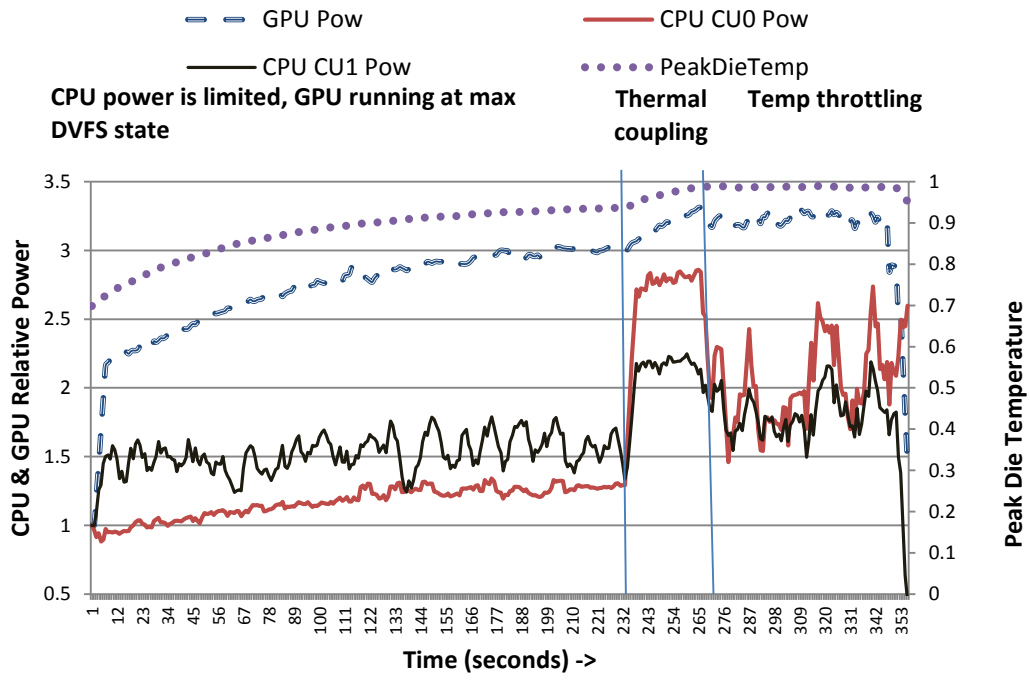


Figure 8: Example of the impact of thermal coupling.

This behavior can be attributed to thermal coupling effects between the CPU and GPU thermal entities (TEs) defined in Chapter 3.4.1. As the GPU warms up (see Figure 8, CU1 has a stronger thermal coupling to the GPU due to its proximity to the GPU (Figure 3), and so its power is initially higher than that of CU0 although they are both relatively low. When they switch to a higher-performing DVFS state, at around 230 seconds, the power in both CUs increases, but thermal effects cause CU0 power to exceed CU1 power. CU0 is on the edge of the die, and its heat is trapped between the edge of the silicon and CU1. The GPU acts as a thermal sink for CU1 due to its larger die area and more distributed heat and, as a result, lower temperature. However, once steady state is reached (more than 267 seconds), CU0 and CU1 temperatures stabilize to roughly equal values.

We conducted two additional experiments to support the preceding discussion on thermal coupling between the CUs and the GPU. In the first experiment, we boosted the CUs to run at a higher power while the GPU executed the same workload at a constant voltage and frequency. We observed the GPU temperature was 6^{o}C higher once the CUs were boosted, indicating thermal coupling between CU1 and the GPU.

In the second experiment, we performed temperature measurements with a high-power, two-thread CPU application. We first pinned the threads to CU0, then pinned them to CU1. The GPU is idle and power is managed by BAPM. When the application ran on CU0, we observed the peak die temperature was higher than when the application ran on CU1, indicating worse heat flow from the CU next to the edge of the die. Further, the GPU and idle CU temperatures rose by 13^{o}C when one of the CUs was active with all others idle, again indicating heat transfer effects.

Better coordination of the CPU and GPU power states could have reduced the effects of thermal coupling, postponed (premature) throttling, and thereby increased overall performance as well as energy efficiency, e.g., joules/instruction. For example, coordinated power management ensures a greater percentage of the heat generated is due to useful work, e.g., instruction execution, and not wasted.

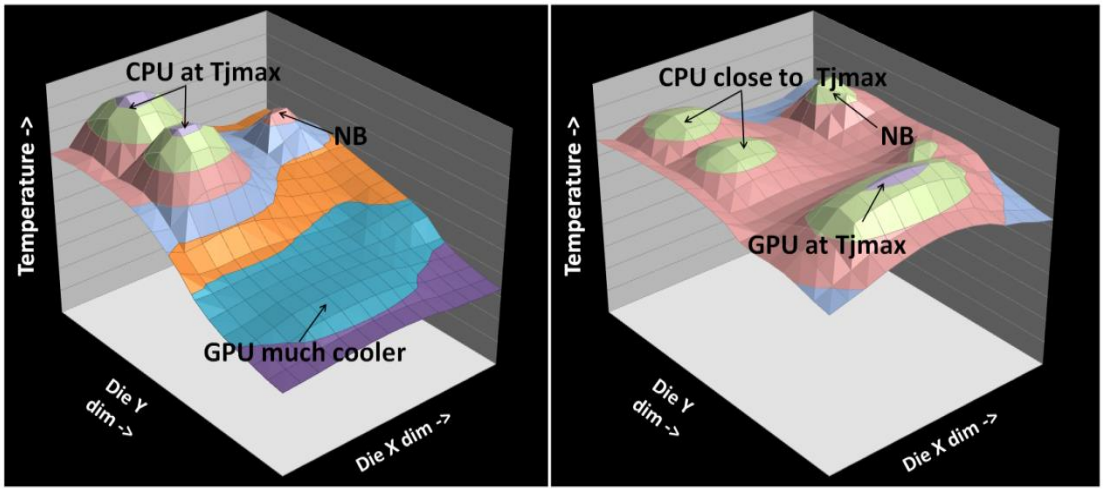


Figure 9: Thermal densities under CPU-centric (left) and GPU-centric (right) workloads.

4.1.2 Thermal Signatures

Section 4.1.1 discussed the heat transfer properties of the TEs on the die. This section details the differences in the thermal characteristics of CPUs and GPUs and how they affect performance-coupled applications.

The thermal signature of a TE reflects its ability to translate power to temperature. It is measured by the distribution of power density across the occupied area. In this sense, the thermal signature of a GPU is quite different from that of a CPU – the latter is more "thermally dense". In Figure 9, we show on the left side a simulated heat map of the Trinity system when running a CPU-centric, L1 cache-resident, high-power workload with an idle GPU. The simulated heat map was constructed by feeding measured power levels and power density while running the workload into a thermal grid model. The right side shows a heat map for *HotSpot* [19], a GPU-centric workload with the serial portion being executed on the CPU. The thermal maps show the steady-state thermal fields produced with the BAPM algorithm across the two CPU CUs, the GPU, and the NorthBridge as labeled in the figure. Tjmax refers to the maximum junction temperature allowed by the die. The

temperature distributions in Figure 9 are steady-state distributions and therefore correspond to the region of Figure 8 after 267 seconds (i.e., after the BAPM algorithm throttled the CPUs and GPU once they reached peak junction temperature).

The thermal characteristics of the workloads vary significantly. The CPU-centric workload shows high heat density in the CPU CUs while the GPU-centric workload shows a wider and flatter temperature distribution across the GPU. The computational area of the CPU, which is where most of the power is consumed, is much smaller than the computational area of the GPU. The complex, out-of-order CPU structures combined with their relatively small areas lead to higher thermal density for the same power and, thus, higher temperatures [52]. The GPU, on the other hand, performs computation across many simple in-order SIMD units that encompass a large area, leading to a lower thermal density for the same amount of power.

There are two consequences to the higher thermal density in the CPU. The first is that the CPU consumes its available thermal headroom more rapidly than the GPU when it is actively doing computation. . Figure 10 shows the thermal time constant of CPU and GPU for a CPU centric workload, where Figure 11 shows the thermal time constant of CPU and GPU for a GPU centric workload.

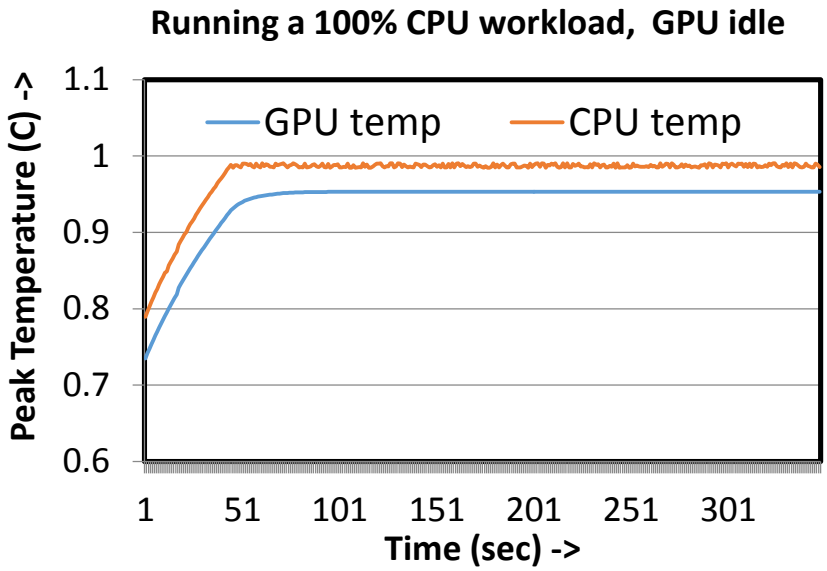


Figure 10: Thermal time constant with a CPU-centric workload.

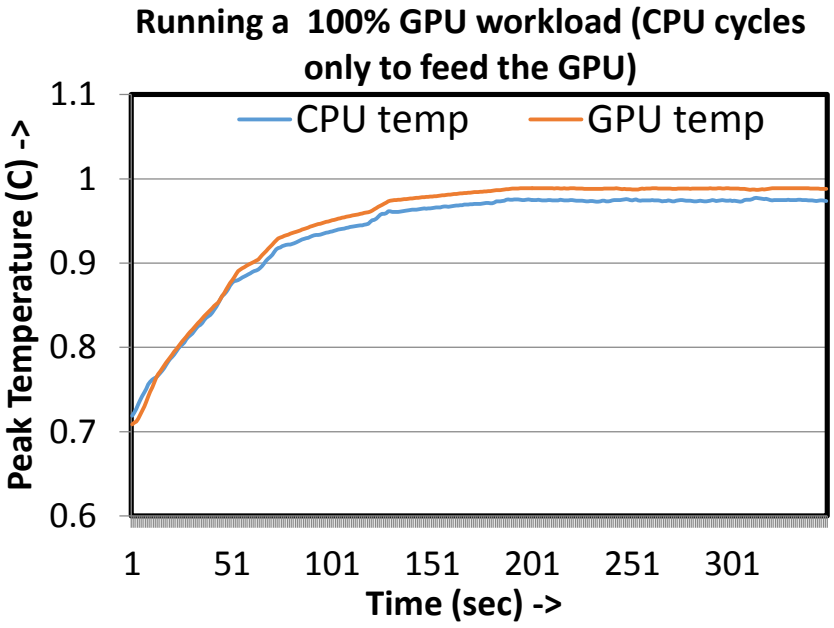


Figure 11: Thermal time constant with a GPU-centric workload.

In our analysis, we observed that the CPU heats up approximately 4X faster that the GPU leading to a much smaller thermal time constant on the CPU. As a result, the GPU

can sustain a higher power boost than the CPU for a longer period before locally reaching the thermal limit. In some cases, this results in sustained power dissipation that is higher than the TDP power. For example, in the simulations shown in Figure 9, the TDP of the APU complex is 19W; the total power for the CPU-centric workload is 18.8W, while the total power for the GPU-centric workload is 19.7W for the same thermal limit.

The second consequence of the higher thermal density in the CPU is the destructive effect of thermal pollution on other components on the die. The rate and extent of thermal pollution depends on the thermal signatures of the entities. The distinct thermal signatures lead to a larger thermal gradient between the CPU and GPU when the CPU is active than when the GPU is active. Heat from the CPU spreads, heating neighboring components, increasing leakage, and accelerating temperature rise. The thermal coupling effects can be seen in Figure 8 and Figure 9.

In a thermally coupled system, the TEs do not influence each other's performance as long as they are all running well below the thermal limit. Power management employs boost algorithms to improve performance by pushing the processor to operate near the thermal limit, reallocating power across the CPU and the GPU. As shown in Figure 8, boosting based on available thermal headroom can sometimes be detrimental to the application performance. The complexity of the power-management task is exacerbated in heterogeneous systems because application performance relies on components with widely varying thermal signatures and coupling. The distinct thermal signatures lead to larger thermal coupling across the die when power management across the die is not coordinated.

4.1.3 Summary

The preceding analysis shows that heterogeneous cores have distinct power and thermal signatures that give rise to new management challenges. Phenomena such as thermal coupling and thermal pollution between compute elements can produce complex interactions with performance and can degrade overall system efficiency if power states of the cores are largely un-coordinated. In future heterogeneous processors, architects must

deal with thermal coupling actively besides power in order to reason about and improve overall performance and efficiency of the system. In the next section we will see how lack of coordinated management can lead to overall poor efficiencies in heterogeneous systems with distinct performance signatures.

4.2 Performance Coupling

As the trend towards heterogeneous processors continues with tightly coupled accelerated processing unit (APU) designs, the companion emergence of modern programming languages based on the bulk synchronous parallel (BSP) model such as CUDA, OpenACC, and OpenCL, is making such processors viable for general purpose and scientific computing. However, the tighter integration of CPUs and GPUs and consequently emerging workloads and programming models result in greater performance dependencies between the CPU, the GPU and the memory subsystem. For example, CPU and GPU memory accesses interact in the memory hierarchy causing interference between reference streams and consequently impacting performance. Further, in the offload model of computation the CPU is responsible for "feeding" the GPU both from the perspective of launching computations as well as providing input data and consuming output data. The rate at which the CPU can perform these functions determines the utilization of the GPU. Too slow and the GPU is underutilized. Too fast and the CPU is stalled. These observations are similar to the relationship between the CPU and the memory system leading to the notions of compute-bound vs. memory bound workloads. Thus the overall performance of the processor is dependent on the interactions between the CPU, GPU and memory. We refer to such performance critical functional dependencies among multiple compute and memory elements as *Performance Coupling.* In the following sub-sections we will describe performance coupling between heterogeneous compute elements and between compute and memory elements.

4.2.1 Performance Coupling between Heterogeneous Compute Elements

The CPU and GPU have fundamentally distinct performance signatures. Modern GPUs contain hundreds of simple in-order ALUs, hardware thread schedulers, and access to fast on-chip and high-bandwidth memories. This translates to excellent peak performance and power efficiency for a wide range of applications. However many applications that do not have enough thread-level parallelism or have significant serial fractions do not map well to the GPU. The bulk synchronous parallel nature of the GPUs leads to poor performance on codes that are control flow intensive or which have irregular memory access patterns. Codes with irregular control flow, irregular memory accesses, and phases with low parallelism are better suited for executing on the complex out-of-order CPU as compared to the GPU. The fact is that in emerging applications both the CPU and the GPU are candidates for hosting different portions of the computation or data processing, i.e., these applications make concurrent use of both the compute engines.

Figure 12 shows an example programming model where the host application on the CPU launches a kernel consisting of a 2D/3D grid of work groups on the GPU where each workgroup is comprised of a block of work-items or threads. The work-items all execute in parallel as long as hardware resources such as registers and data-storage spaces are available on the GPU. The host CPU and the GPU have control and data dependencies between computations executing on the two types of cores.

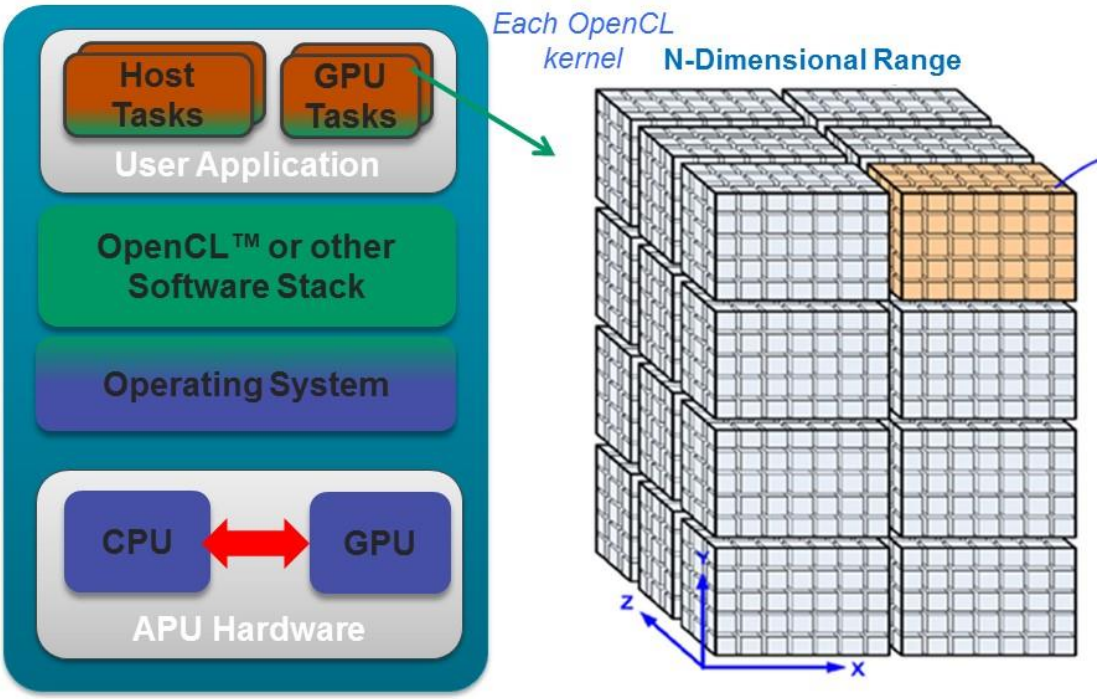


Figure 12: Bulk-synchronous parallel programming model.

Figure 13 provides a canonical illustration of performance coupling in a heterogeneous processor with two CPU cores and a GPU. The CPU is processing computations and placing work in GPU's command queue for processing. CPU0 and CPU1 first execute portions of the computation and prepare the data needed for the GPU to start. Once completed, CPU0 enqueues a compute kernel (work) into the GPU's command queue at which point the GPU can begin the execution. In this example a slow CPU can starve the GPU of data, leading to underutilization of the GPU. If the rate of consumption of the work by the GPU is faster than the CPU can process, the command queue will be mostly empty leading to GPU's starvation. On the other hand, a slow GPU which is not consuming the data generated by the CPU fast enough will lead to over-run of the command queue. Neither conditions are desirable and must be actively managed to maintain an optimal balance in the producer (CPU) – consumer (GPU) relationship. Here for peak GPU utilization, the CPU must provide data to the GPU at a rate appropriate to sustain GPU

performance and vice-versa.

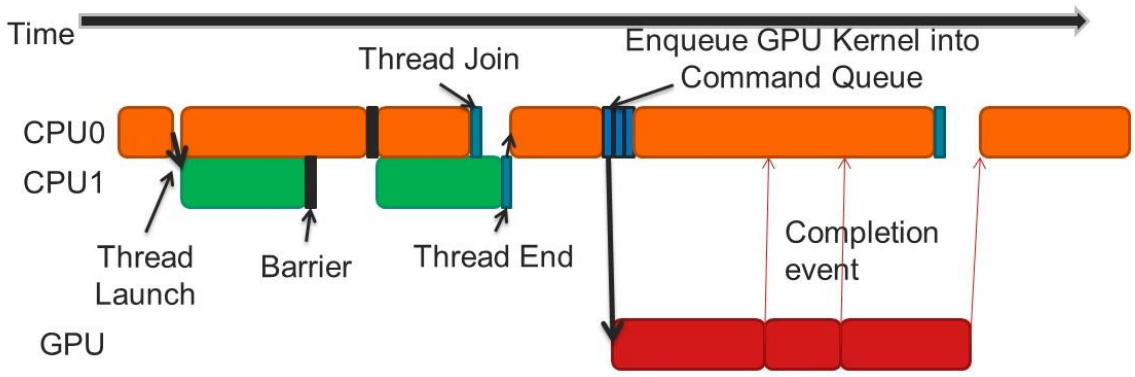


Figure 13: CPU-GPU execution dependency in a heterogeneous processor.

Figure 14 illustrates an HPC application running on an AMD Trinity APU. The figure shows fine-grain communication between the CPU and the GPU on an OpenCL variant of *Lulesh* with 100 node elements per dimension [59]. The x-axis shows time (in milliseconds) and the y-axis shows the CPU utilization as measured by IPC for the multi-threaded CPU, and the GPU utilization as measured by active clock cycles for the data-parallel GPU.

The application is in the start-up phase up to 3200 ms, and the CPU is the primary active component. Subsequently, the CPU primarily plays an assist role delivering data to the GPU for computation leading to low CPU activity (IPC) and high GPU activity. However, there is constant communication between the CPU and the GPU and the performance required of each core is a function of the kernel being run. For instance, the *CalcFBHourGlass* kernel has a higher GPU utilization than the 20+ miscellaneous kernels in the application. The computational demands of the CPU and the GPU vary across program phases, as does the intensity of their interactions.

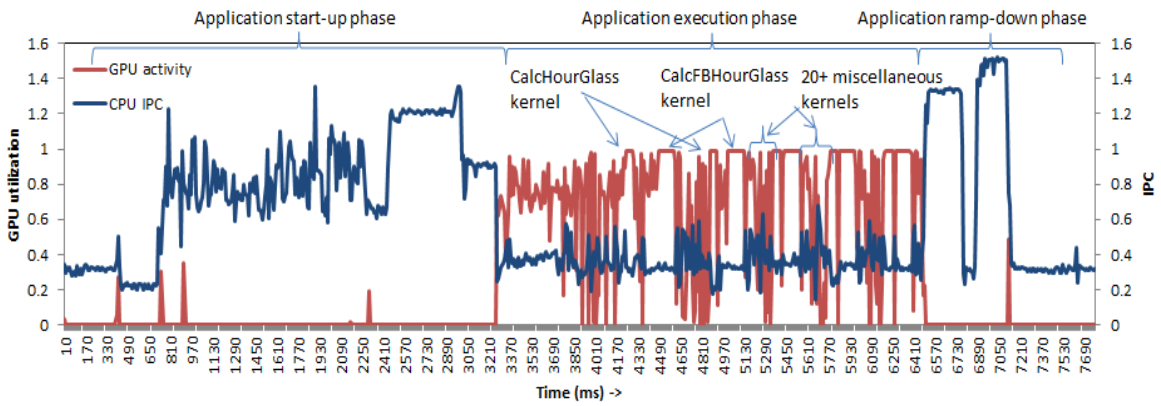


Figure 14: Example phase behavior in an exascale proxy application (Lulesh).

The above analysis attests to the fact that unlike multi-core homogenous architectures in which all cores are identical and the majority of threads are identical, the CPU and GPU differ in both the architecture and execution models. While the former supports asynchronous execution of (relatively) coarse-grain threads, the latter implements a model orchestrating the synchronous execution of thousands of thread blocks or wavefronts, comprising tens to hundreds of fine-grain threads. Consequently, their energy and power behaviors are quite distinct. Further, while the CPU-GPU behaviors are *directly coupled* through the programming model (e.g., through off-load model of kernel execution on the GPU), their executions are *indirectly coupled* via interference within, and competition for, shared on-chip resources such as memory controller and on-chip interconnect.

Managing power states of the CPU & GPU without regard to the scope and intensity of their (coupled) interactions lead to disproportionate reductions in performance when reducing power or energy consumption by moving to lower power states. This again demonstrates that lack of coordination can lead to inefficient power management.

4.2.2 Performance Coupling between Compute and Memory Elements

While the preceding observations reflect the interaction between cores, the DRAM memory system is one of the biggest contributors of overall system power besides compute devices and its power and energy signatures are quite distinct from that of the heterogeneous cores. For example, Figure 15 illustrates the power distribution in an AMD Radeon HD7970 discrete GPU card (dGPU) executing a memory intensive workload *XSBench* [46]. Performance is also coupled between the compute and memory operations and their relative demands must be met to sustain application performance. This distribution of performance and power consumption between compute and memory must operate under a fixed board level power and thermal envelope, while with the advent of on-package DRAM e.g., die stacks and EDRAM [39][56][93][131], they must share an even tighter package power and thermal envelope.

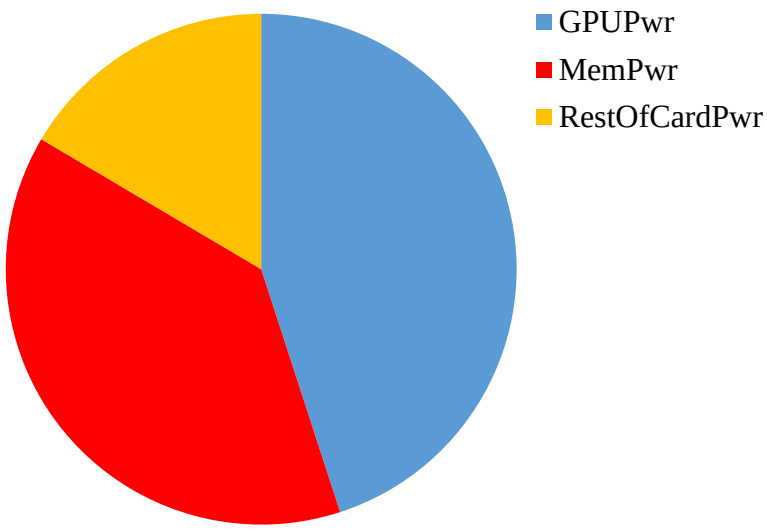


Figure 15: Power breakdown in a typical modern discrete GPU.

In addition to on-die memory integration, future heterogeneous processors are expected to have a significantly larger number of parallel processing cores or compute units

(CU) in order to meet increased compute demands. For example, the national roadmap for HPC now has the goal of establishing systems capable of sustained Exaflop (10^{18} flops/sec.) performance. However, the road to exascale is burdened by significant challenges in the power and energy costs incurred by such machines. Due to dark silicon effects it may not be possible to power all of the transistors in a chip [40]. This accentuates the problem of effectively sharing the power envelope between the memory system and the compute.

Emerging applications are becoming increasingly unstructured and irregular in their memory access patterns. For example, Figure 16 demonstrates the evolution of fine grained parallelism over time for a *breadth-first search (BFS)* algorithm across two different input graphs [77]. The X-axis indicates the iteration count and the Y-axis indicates the amount of parallelism for that iteration as represented by the number of graph nodes per iteration. Due to the synchronization barrier at the end of each iteration, load imbalances and critical paths can cause significant reduction in power efficiency by reducing hardware utilization. Similarly, memory divergence and memory bandwidth access rates also have a large effect on power efficiency.

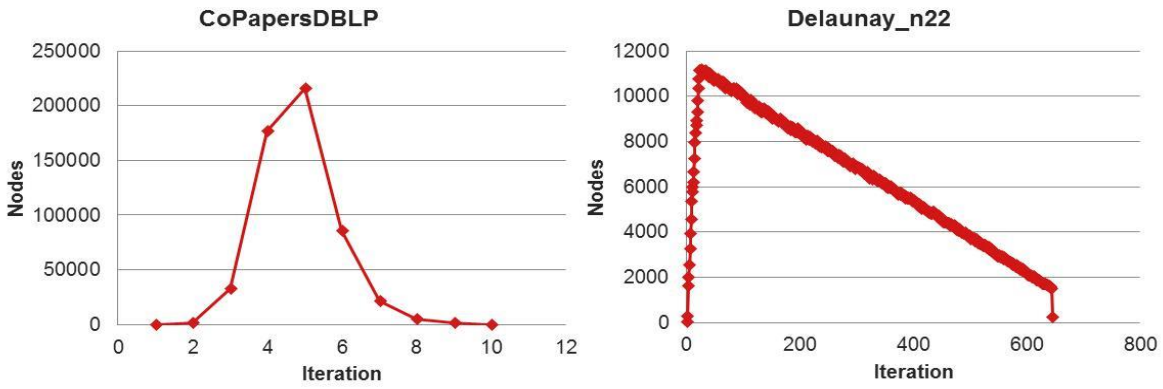


Figure 16: Parallelism over time for two input graphs.

4.2.2.1 Application Characterization

Recognizing the time-varying redistribution of compute and memory demands for emerging applications, the first task is to understand the behaviors of such applications to identify the performance coupling effects between compute and memory, and the trade-offs in shifting power from memory to compute or vice-versa. We specifically focus on analyzing the relationship between the compute-memory behaviors of the hardware platform with that of the applications. For example, the ops/byte behavior of an application (compute operations per byte of memory data transfer) represents the relative demands placed on the GPU cores and memory system. Hardware tunables such as the number of cores, their operating frequency, and the memory bandwidth collectively capture the relative time and power cost of performing operations vs. memory accesses in the hardware platform. The ops/byte behavior of the application is time varying and the ops/byte costs of the platform depend on the specific values of the hardware tunables such as compute frequency or memory bus frequency. For example, we studied the ops/byte behavior of *Graph500* [81] running on an AMD Radeon HD7970 GPU with GDDR5 memory [76]. More details on HD7970 can be found in Chapter 3.3.2. The ops/byte behavior varied from lows of 0.64 ops/byte to bursts of 264 ops/byte.

High ops/byte ratios imply the memory system can be run at lower speeds relative to compute with little to no performance degradation but with lower overall platform power. The time varying behavior of the application implies that a lack of continuous coordination of the power states of the GPU and memory system can lead to imbalances between the power and time costs of compute and memory resulting in significant degradation of performance and power/energy efficiencies. Towards this end, the rest of this subsection presents a detailed characterization of the relationship between application behaviors and settings of hardware tunables for compute throughput and memory bandwidth.

4.2.2.1.1 Experimental Methodology and Terminology

The performance scaling and ops/byte characterization of the GPU hardware is carried out as follows. The total number of active CUs and CU operating frequency produces a peak operation rate (ops/sec) or compute throughput. The memory channel frequency determines the peak memory bandwidth (bytes/sec). The ratio provides the ops/byte value of the platform hardware for a particular combination of the number of active CUs, CU frequency (all operating at the same frequency), and memory bandwidth. The number of active CUs can be varied from 4 to 32, and the CU frequency can be varied from 300MHz to 1GHz, in steps of 100MHz. A specific setting of CU and CU frequency is called the *compute configuration*. Memory bandwidth can be varied from 90GB/s (at 475MHz) to 264GB/s (at 1375MHz) in hardware, in steps of 30GB/s (150MHz). A specific setting is called the *memory configuration*. The total number of combinations of compute and memory configurations is approximately 450. Each combination reflects a specific value of ops/byte *delivered* by the platform hardware and a specific balance between compute and memory bandwidth. It also reflects a specific balance between power devoted to computation vs. memory access. If there is significant imbalance between demanded ops/byte of the application and what the platform delivers, execution time and energy inefficiencies result.

4.2.2.1.2 Performance Scaling Trends and Hardware Balance

We observe that GPGPU kernels show a number of common scaling patterns across the hardware configuration state space we have explored. Some patterns are limited by the available computational resources, others by the memory bandwidth, and still others can be affected by both limitations. We also demonstrate a few non-intuitive scaling patterns, such as performance losses from shared cache thrashing and performance plateaus from a lack of parallelism.

Compute-Bound: Figure 17 shows an example performance scaling surface for a compute-bound kernel. Here Y (vertical) axis shows performance normalized to the

slowest hardware configuration that we studied. The X (horizontal) axis shows the various CU frequencies that we studied (keeping the number of active CUs fixed at maximum possible value), and the Z (depth) axis shows the memory bandwidth settings we used. We define compute-bound kernels as those whose performance can be improved by giving them more compute resources. Such resources can be of two types – the number of active CUs and the core frequency. Typically, the performance of compute-bound kernels is not limited by the available memory bandwidth because the memory footprint of such kernels is relatively small and/or exhibited good caching behaviors.

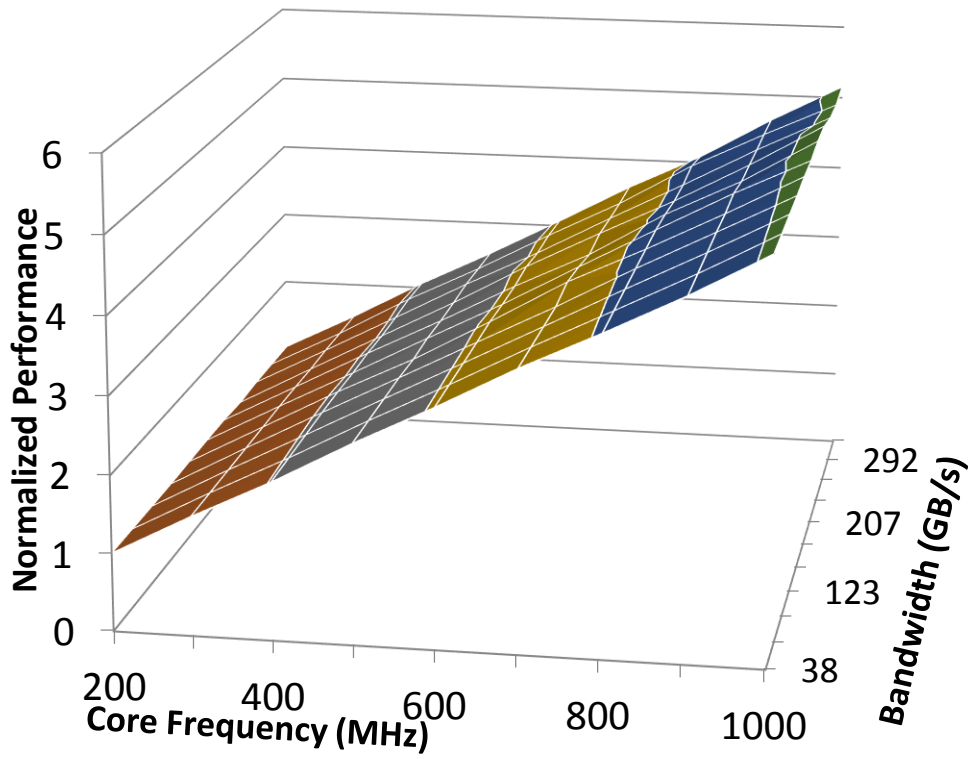


Figure 17: 3D plot for the performance scaling of a compute bound kernel with compute frequency and memory bandwidth.

We observed that a kernel can be compute-bound for two primary reasons. First, there are kernels that do not have enough parallel work, but are (compute) latency sensitive.

The performance of such kernels improves as core frequency is scaled but is insensitive to the number of CUs. Second, there are kernels that exhibit significant parallelism allowing them to achieve higher performance with either higher frequency or CU count. These kernels also have low memory footprint and/or good locality that make them less sensitive to memory bandwidth. *MaxFlops* and *FFT* are both examples of such kernels. The former is a computational throughput microbenchmark from the SHOC suite [30] that is commonly used in the GPGPU community to stress the hardware against its compute limits. The latter is a more traditional high-compute GPGPU application which calculates the Fast Fourier Transform [2].

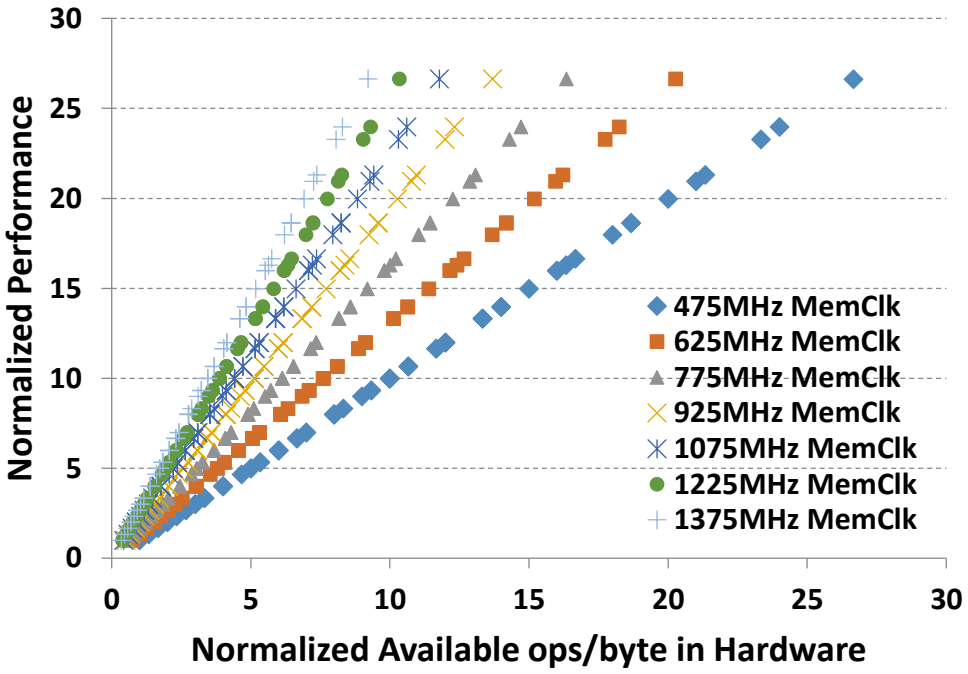


Figure 18: Performance scaling of MaxFlops with available ops/byte in hardware.

Figure 18 describes the corresponding ops/byte behavior of the *MaxFlops* application against platform ops/byte. The X-axis shows the ops/byte provided by the hardware. Each curve in the figures corresponds to a fixed memory bandwidth

configuration. Each point on the curve is a different compute configuration with increasing CU frequency and number of CUs as we move to the right, i.e., increasing ops/byte of the platform. The Y-axis shows performance (i.e., 1/execution time). Both the X and Y axes are normalized to those of a minimum hardware configuration with 4 active CUs, 300MHz compute frequency and 90GB/s memory bandwidth.

As we can see from Figure 18, increasing compute throughput results in linear increase in performance for a fixed memory bandwidth. Also, for the same compute-to-memory bandwidth ratio in the platform (i.e. same ops/byte value on the x-axis), higher available memory bandwidth means higher available compute throughput and hence higher performance for this benchmark. However, it is clear that maximum performance (at 27 normalized performance) is achieved at multiple memory configurations. All these points are at the same compute configuration—maximum 32 CUs and maximum 1GHz compute frequency. However, the most energy-efficient point is the rightmost point at 27 normalized ops/byte of x-axis, which corresponds to the lowest memory bandwidth. This is because *MaxFlops* is not memory sensitive—running at the lowest memory bandwidth does not hurt performance, but significantly improves energy efficiency.

Memory-Bound: Memory-bound kernels can generally be described as those that are primarily or solely affected by the available memory bandwidth between the GPU and its DRAM. In our tests, this is controlled by DRAM bus frequency, which can also affect memory latency. Nonetheless, most GPU kernels are latency tolerant due to multithreading: they only begin to lose performance when a lack of DRAM bandwidth causes the latency of an access to scale with the number of accesses.

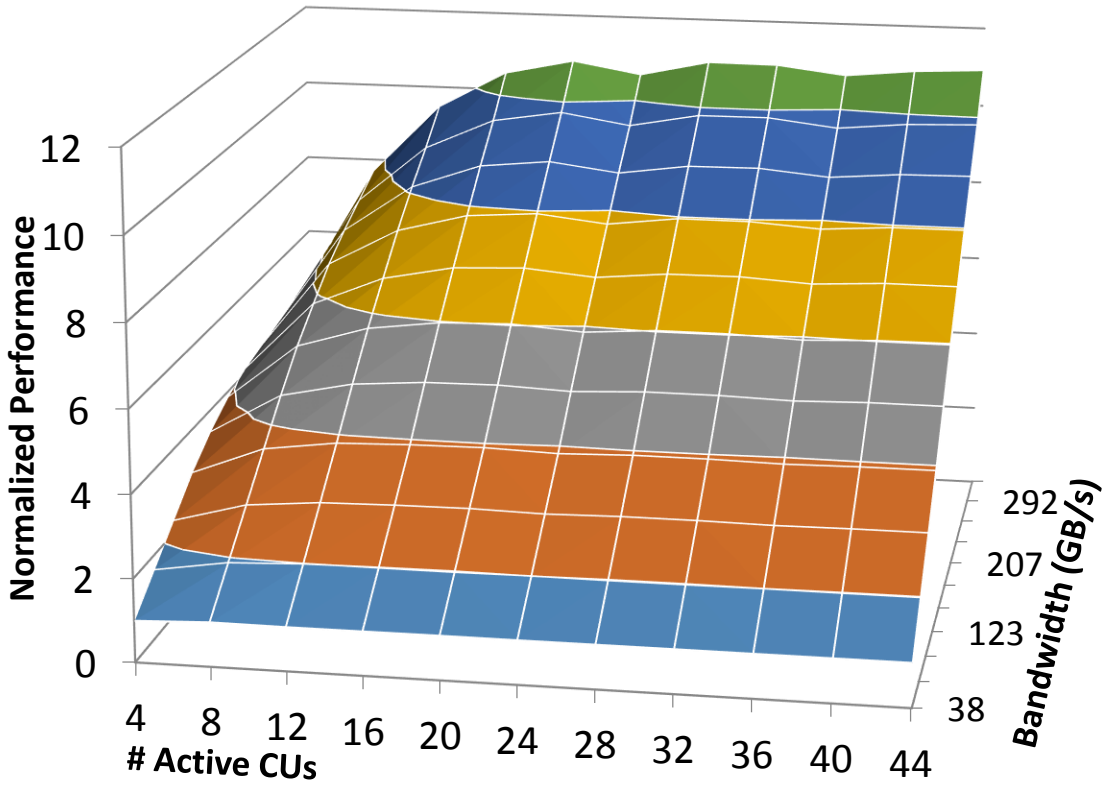


Figure 19: 3D plot for the performance scaling of a memory bound kernel (DeviceMemory) with active compute units (CUs) and memory bandwidth.

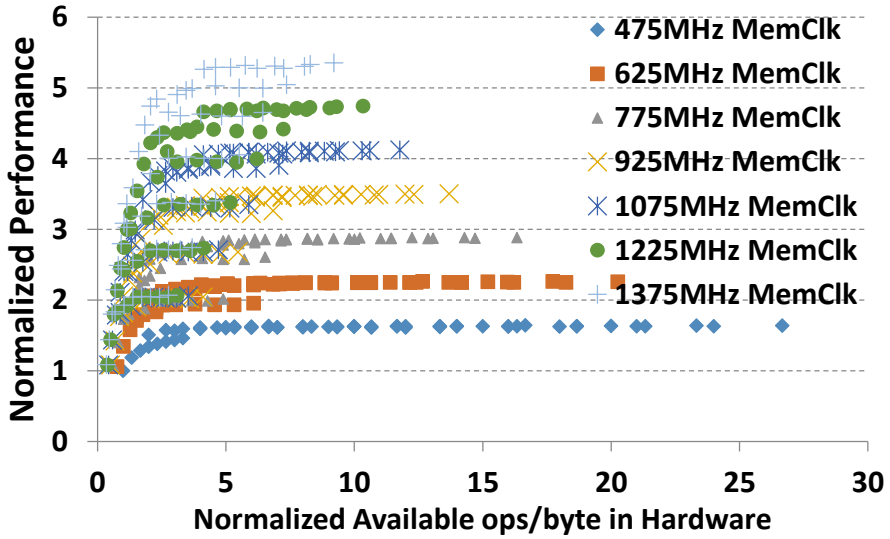


Figure 20: Performance scaling of DeviceMemory with available ops/byte in hardware.

Figure 19 shows how the kernel *readGlobalMemoryCoalesced* from the *DeviceMemory* microbenchmark scales as CU count and memory frequency are varied. In most operating regions (i.e. above 12 CUs), the added compute capabilities do not help. *DeviceMemory* is a benchmark from the SHOC suite [30] that is commonly used in the GPGPU community to stress the GPU hardware against its memory limits. DRAM bandwidth controls the performance of this kernel. Now consider the corresponding ops/byte characteristics of this application in Figure 20. We observe that for each value of memory bandwidth, increase in compute throughput does not lead to improved performance beyond a hardware ops/byte of around 5. Hardware configurations with normalized ops/byte of 5 are balanced configurations where compute throughput just saturates the available memory bandwidth. Each memory configuration has a different balance point (the knee of the curve) corresponding to a specific compute configuration. The optimization problem is the selection of the specific balance point that maximizes power and energy efficiencies with minimal impact on performance. Any other combination of compute and memory configurations either wastes power and/or leaves additional performance gains unexploited.

Balanced: Balanced kernels are those whose performance depends on both type of hardware resources – compute (core frequency and/or CU count) and memory bandwidth. As we vary the available compute-to-bandwidth ratio in the GPU hardware, these kernels have some optimal ratio that maximizes performance. Making less bandwidth available causes a kernel to stall waiting for values to return from memory. Using fewer computational resources causes it to become a bottleneck for instruction throughput.

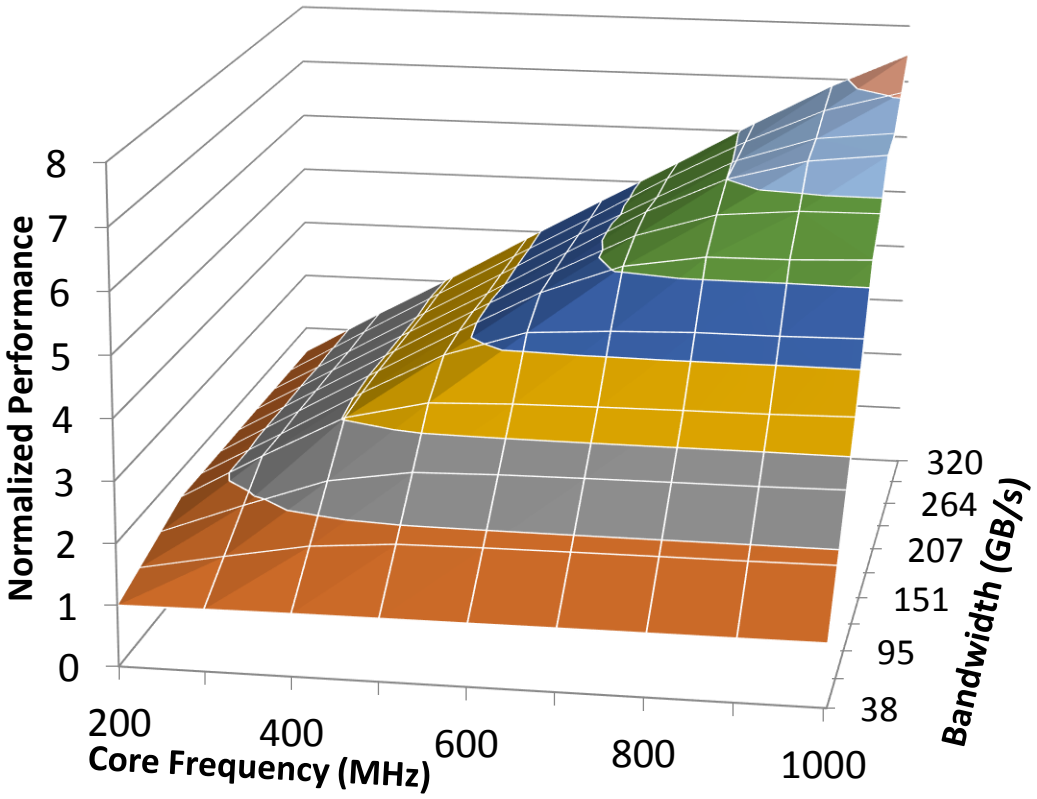


Figure 21: 3D plot for the performance scaling of a balanced kernel (MatrixMultiplication) with compute frequency and memory bandwidth.

Figure 21 illustrates how a balanced kernel scales. This shows that for balanced kernels, there is a "hill" in the performance curve at some compute-to- bandwidth ratio. Moving away from that curve in either direction causes the performance gains to quickly drop off. For such kernels, without increasing both core frequency and memory bandwidth, the performance gains quickly drop to zero. If, however, the kernel has enough bandwidth, the performance scales linearly with added computational power. Figure 21 presents data from the *mmmKernel* kernel in *MatrixMultiplication* [2], but is generally representative of a number of other balanced kernels. It is important to note that not all balanced kernels have the same preferred ratio between compute and memory, meaning that the "hill" in the

curve may occur at different locations. This type of scaling is often seen in kernels that are well designed for modern GPU configurations.

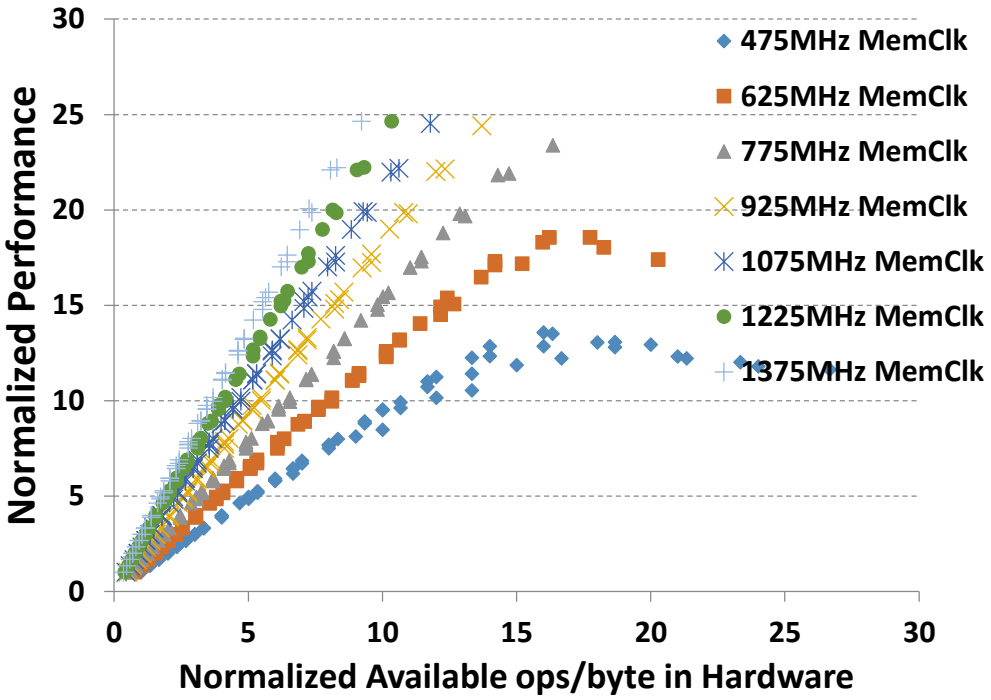


Figure 22: Performance scaling of LUD with available ops/byte in hardware.

Finally, in Figure 22 we show the ops/byte behavior of a balanced application *LUD* and its relation to platform ops/byte balance. *LUD* is a representative scientific application from the Rodinia benchmark suite [18][19] that performs matrix decomposition. The application may be compute-bound or memory-bound depending on the choice of compute and memory configurations. For higher values of memory bandwidth the application remains compute bound across all configurations. For such applications, the best hardware balance point corresponds to the configuration that is the highest and rightmost. For *LUD* this is achieved when normalized hardware ops/byte is at around 15, where compute throughput most effectively matches memory bandwidth demands.

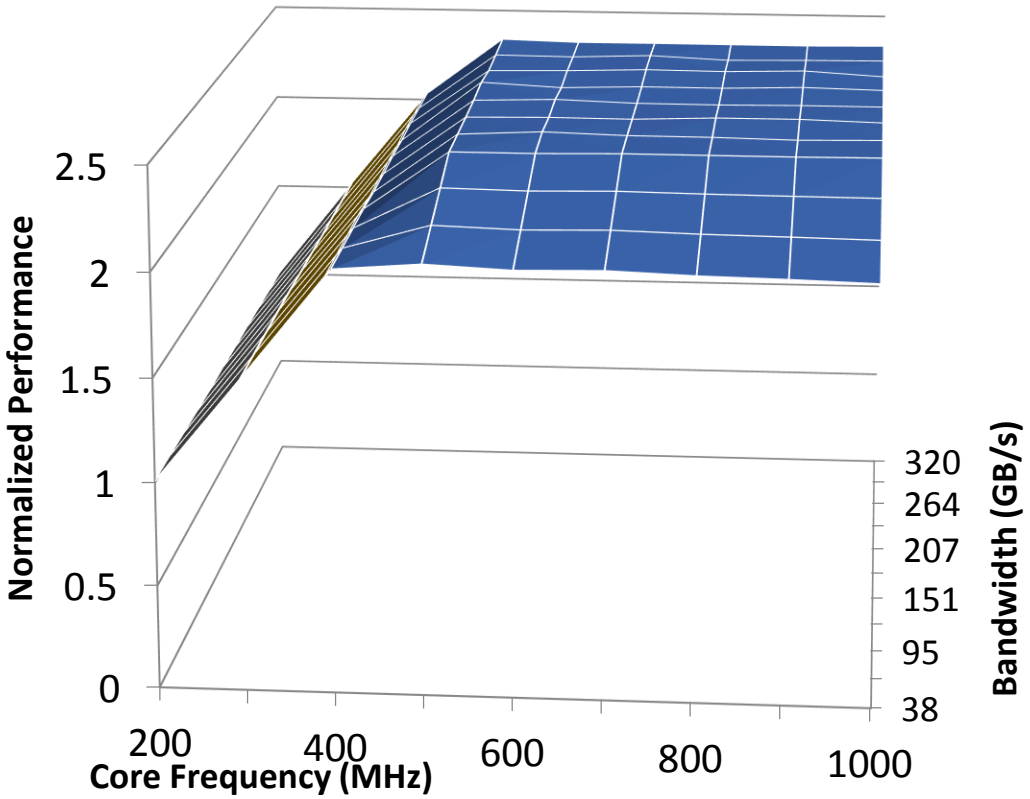


Figure 23: 3D plot for performance of a kernel that plateaus eventually and does not scale.

Other representative scaling patterns: In addition to the above scaling trends, we found a few kernels whose performance changes very little with any of the hardware resources or stops scaling beyond a certain point. Such behavior often happens because of programming errors, interconnect bandwidth limitations or algorithmic limitations such as being completely serialized or a single thread doing all the work, or not having enough work. An example scaling surface is demonstrated in Figure 23 for *Blackscholes* [2].

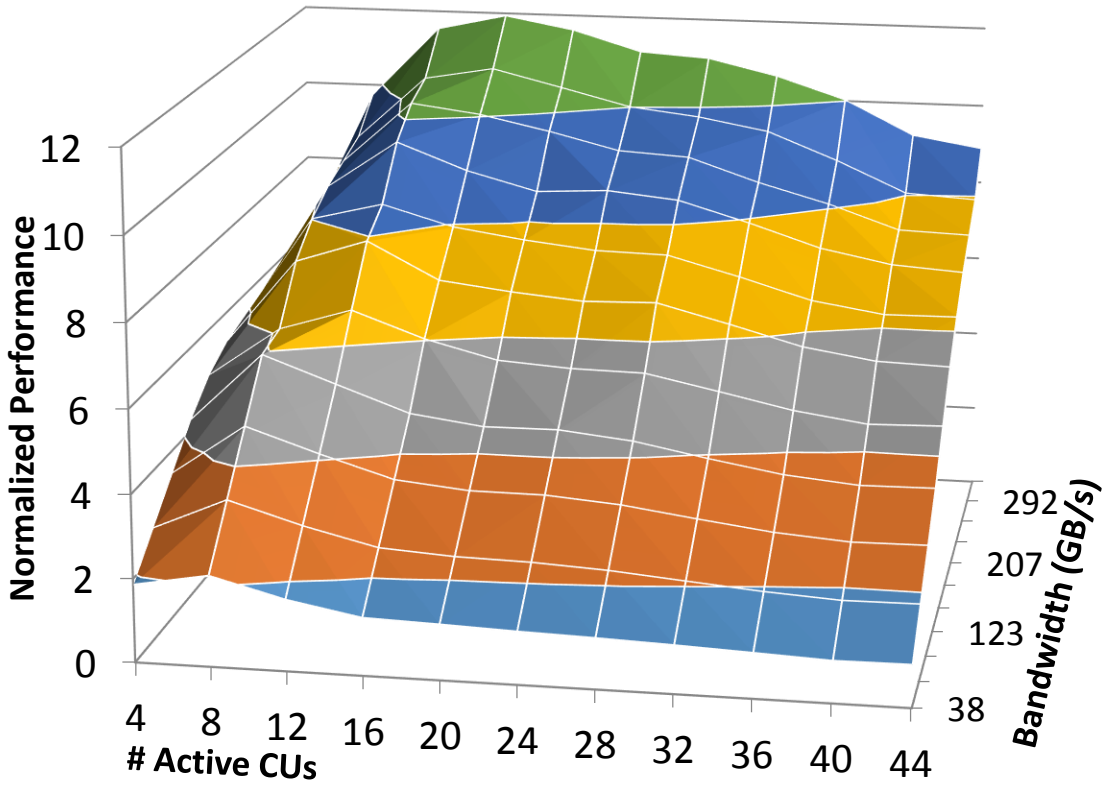


Figure 24: 3D plot for performance of a kernel that peaks and then falls off as more resources are added.

Another interesting scaling trend is seen as described in Figure 24, where the kernel shows initial improvement in performance as more resources (number of CUs in this example) are added. But the performance *peaks* and then starts falling as further resources are added. The kernel *btreefind* from the *BPT* application is an example of such performance scaling [28]. Kayiran et al. and Lee et al. have seen similar peaks in simulation, which our experiments corroborate [61][71].

We found that this problem commonly occurs due to destructive interference among threads for the shared L2 cache. In the GPU hardware used in our experiments, the L2 cache is shared across all the active CUs. As more CUs are added, more threads are activated. This means that each thread gets a smaller share of the L2 capacity if it does not

share data with other threads. At some point, the shared L2 size becomes insufficient to hold the working set of active threads running in the CUs. At this point, memory requests from various CUs interfere with each other, causing cache thrashing. Total bandwidth demanded by the application remains the same, but more accesses are now pushed to the DRAM system. This leads to reduced GPU performance. Since increasing memory bandwidth helps to reduce the L2 miss penalty, these programs benefit from running with higher memory bandwidths.

In general, the above studies point out that the optimal *hardware balance* point, at which the compute and memory demands match the platform costs, varies across applications and application phases. It also varies across different hardware platforms.

4.2.2.2 Power Reduction Opportunities

In this section we characterize the power reduction opportunities in choosing the optimal hardware balance point. We measure total power at the GPU board level using a power instrumentation setup at the PCIe connector interface from mother board to the GPU board. Further details of this setup are described in Chapter 7.5. This includes the power of the GPU, off-chip DRAM memory, and the remaining board components. In Figure 25, the X-axis indicates the available ops/byte in hardware under a constant memory bandwidth of 264GB/s (i.e., fixed memory configuration). The Y-axis shows the cost of a compute configuration, i.e. effect of changing the compute configuration on overall board power for memory intensive *DeviceMemory*. Each line represents a CU count (4-32) and each point on a line shows increasing CU frequency. Relative to the compute configuration of the lowest possible setting of 4CUs and 300MHz, the overall board power scales by about 70%. Similarly, Figure 26 indicates the power savings possible through reduction of memory bandwidth by adjusting the memory bus frequency for a compute intensive application - *Maxflops*. Note that the memory bandwidth variation was performed at a fixed voltage as the memory system voltage could not be controlled in our experimental setup.

Therefore, in general the differences would be even greater. Here, the number of active CUs and compute frequency were kept fixed at the default configuration of 32 CUs and 1GHz respectively. We see up to a 10% power reduction between operating at the lowest memory frequency of 475MHz (90GB/s) compared to the memory frequency of 1375MHz (264 GB/s). This data underscores the power saving opportunities associated with balance points.

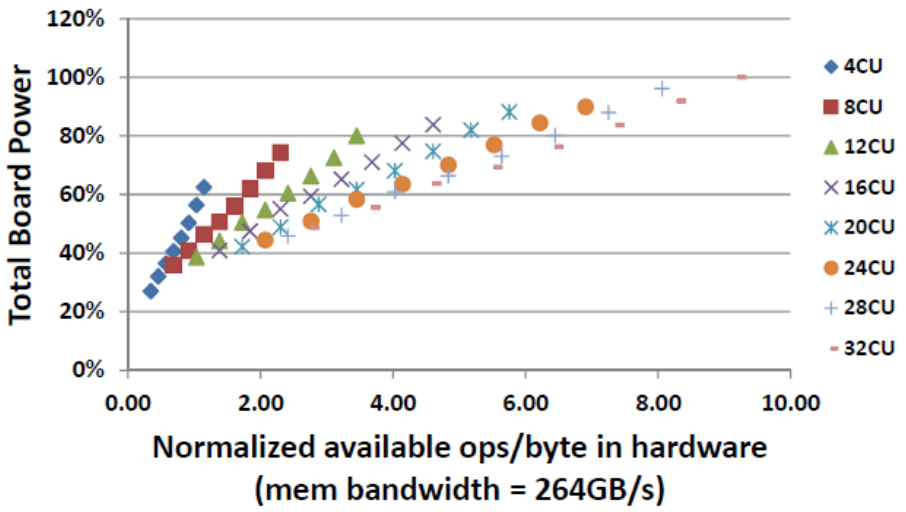


Figure 25: ***DeviceMemory*'s GPU card power across compute configurations at constant 264GB/s memory bandwidth.**

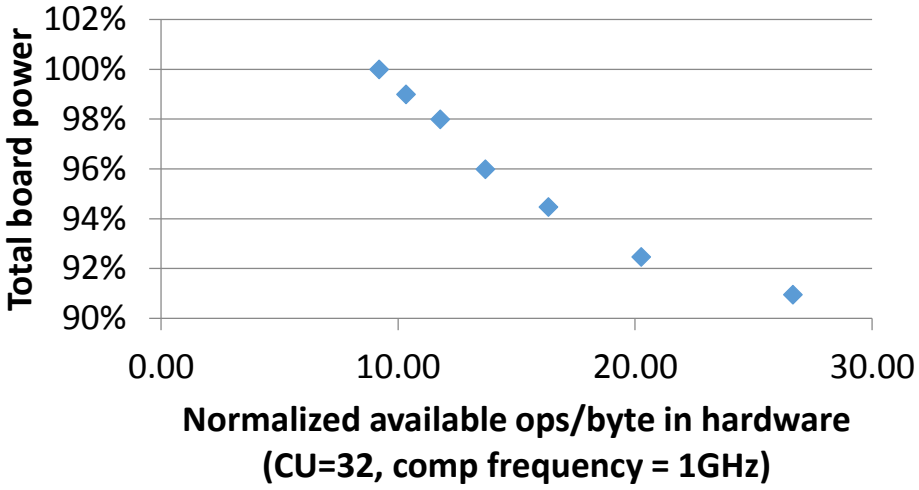


Figure 26: ***MaxFlops*'s GPU card power across memory bandwidth configurations at 32CUs and 1GHz compute frequency.**

4.2.2.3 Summary

From the above study we find that in general, hardware balance can be achieved with many different combinations of compute and memory configurations. The optimization question is the selection of the *optimal hardware balance* – the balance point that maximizes performance and improves energy efficiency and is a function of the time-varying workload and platform configurations. Lack of coordination between the power states of compute and the memory system leads to unwarranted energy and performance inefficiencies. Thus, it is necessary to monitor the power and energy consequences of the compute and memory interactions so that the power states of the compute and memory elements can be cooperatively managed and coordinated.

4.3 Summary of Key Management Challenges

The preceding analysis shows that there are a variety of complex power, thermal and functional interactions between the CPUs, GPUs, and the memory subsystems that affect the performance, power consumption and energy efficiency of a heterogeneous processor. This section summarizes the new management challenges in heterogeneous systems and forms the foundation for the rest of the work in this thesis.

- ***Thermal Coupling and thermal signatures:*** Power and thermal capacity are shared resources in heterogeneous processors. The distinct thermal signatures of the CPU and GPU necessitate careful management and allocation of thermal capacity. For example, we see that higher thermal density in CPU leads to faster consumption of thermal headroom. We also find that significant thermal coupling and thermal pollution occurs from the active to the idle components of the die. To be effective, power management algorithms must understand the consequences of thermal

coupling and coordinate the power states between the diverse compute elements to maximize performance under power and thermal constraints.

- ***Performance Coupling between Heterogeneous Compute Elements:*** Strong performance coupling exists between the CPU and GPU in heterogeneous processors due to performance dependencies arising from their tight physical integration, emerging programming models and emerging workloads with time-varying redistribution of compute intensities between the CPU and GPU. However, unlike multi-core homogenous architectures in which all cores are identical, the CPU and GPU differ in both architecture and execution models. Hence their power and energy behaviors are quite distinct. To be effective, power management algorithms that determine the DVFS states of the CPU and the GPU must be cognizant of these effects, their interrelationships, and their combined effect on performance.
- ***Performance Coupling between Compute and Memory Elements:*** Compute and memory behavior are fundamentally performance coupled. If we ignore this coupling in managing platform power, significant energy/power is wasted. The time-varying relative demands of the applications must be matched with the relative compute and memory costs of the hardware platform to achieve hardware balance. Hence to retain the most power efficient operation we need a runtime power management infrastructure that should coordinate power states of the processor (GPU) and the off-chip memory system so that they are in balance

Overall, these new challenges clearly point towards the need for chip-scale coordinated power management, as illustrated conceptually in Figure 27, to achieve performance scaling and energy efficiencies under a chip-wide TDP envelope and power budget. The efficiency with which we can manage these problems will depend on 1) identifying a key set of run-time metrics and sensitivity predictors that can be used to manage these effects, and 2) developing online coordinated power management techniques

to optimize system level global metrics in heterogeneous CPU-GPU-memory processors such as performance and energy efficiency under global constraints such as thermal limits and power budgets. The rest of this thesis focuses on development, implementation and evaluation of the run-time monitoring and management metrics, predictors and algorithms to cooperatively manage the processor physics and functional consequences of tight integration in heterogeneous processors.

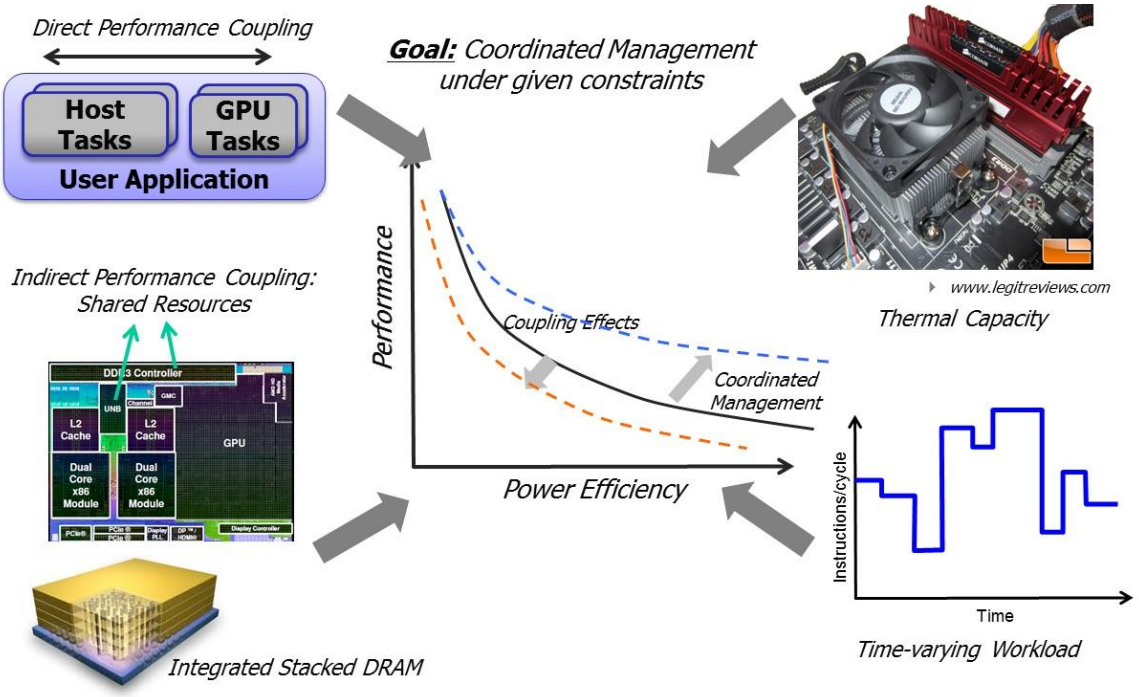


Figure 27: Chip-scale coordinated power management.

CHAPTER 5

THERMAL COUPLING MANAGEMENT

In CHAPTER 4 we described how thermal coupling effects could lead to interactions with the power management algorithms that can increase power and performance inefficiencies. Effective use of thermal capacity can be realized with power management algorithms that consider such thermal coupling effects. This chapter presents a dynamic power management approach called Cooperative Boosting (CB) to coordinate power allocation between the CPU and GPU in order to maximize performance under the thermal and power constraints of the processor package.

5.1 Overview

Modern, high-performance client processors are composed of heterogeneous cores that are managed to create a compelling user experience. Power management is a critical piece of the user experience, with the goal of allocating power adaptively across cores to produce the best performance outcome within a fixed processor power and thermal envelope.

The maximum power for a processor (i.e., the thermal design point (TDP)) is set based on running a heavy workload under worst-case conditions [99]. It is an upper bound for the sustainable power draw of the core and is used to determine the cooling system required. Under normal operating conditions, however, not all components are active at the same time or to the same extent, leaving thermal headroom in the system. Power-management technology such as Intel's Turbo Boost [99] and AMD's Turbo CORE [83] take advantage of the thermal headroom to increase the active cores' frequencies until either the maximum performance state or the thermal limit is reached.

As discussed in Chapter 4.1, modern boost algorithms greedily boost the frequencies of the CPU or GPU cores to utilize all of the available thermal headroom and thereby improve performance. These boost algorithms seek fairness through allocation of power across cores in proportion to expected performance benefits. This works well for many applications in which the type of computation dictates the component that requires boosting. For graphics applications, the GPU is the obvious choice, as is the CPU for many control-divergent, general-purpose applications. However, for applications that require cooperative execution of both CPU and GPU cores, these boost algorithms can break down and degrade, rather than improve, performance. This occurs due to the tight interactions between performance coupling and thermal coupling.

To this end, this chapter makes the following contributions:

- Demonstrates the interactions between thermal coupling and performance coupling and their effect on system performance using hardware measurements and analysis from a state-of-the-art heterogeneous client system.
- Identifies run-time metrics to capture these interactions between processor physics and performance dependencies.
- Proposes a cooperative boosting (CB) algorithm for the coordinated management of power states of the CPU and GPU to optimize performance, and,
- Provides a detailed, measurement-based analysis of the performance of CB in comparison to a state-of-the-practice boost algorithm for exploiting thermal headroom across a range of benchmark applications.

The rest of the chapter is organized as follows: Section 5.2 examines the interactions between performance coupling and thermal coupling. Section 5.3 presents run-time metrics to capture these interactions. Section 5.4 describes the power management algorithm, Cooperative boosting (CB), to balance the needs of performance coupling with the thermal coupling effects. Finally, Section 5.5 provides the experimental setup and Section 5.6 presents a detailed evaluation of CB on a modern heterogeneous processor.

5.2 Greedy vs. Needy Power Management: Interactions between Thermal Coupling and Performance Coupling

As discussed in CHAPTER 4, for emerging compute applications both the CPU and the GPU are candidates for hosting many components of the application. For such applications, the CPU and the GPU are performance-coupled. While thermal coupling places an upper bound on performance potential, performance coupling establishes the minimum performance interaction needs that must be met. This section explores the performance-coupled nature of compute applications and the limits placed by thermal coupling by demonstrating the sensitivity of performance to the CPU performance states. In the following experiments, we statically fix the highest-performing (i.e. frequency) CPU P-state permitted by the power-management algorithm, which we denote as the *P-state limit.* The local CPU power controller may change the P-state to a lower-performing P-state based on the thermal state, but it cannot exceed the P-state limit. Figure 28 presents the impact of CPU P-state limits for *Binary Search, HotSpot, and Needle* [2][18][19].

In Figure 28, the x-axis is labeled with the CPU P-state limit for that experiment. In addition, we show results for the baseline, which refers to the default Trinity power-management system (Section 3.4.1). Limiting to Pb0 (highest boost state as indicated in Table 1) means all P-states are available to the power-management controller, which is the same as the baseline case and hence is not shown separately. The left-side y-axis refers to the stacked bar charts, and it shows the percent of time the GPU spends in low-, medium-, and high-DVFS states. The right-side y-axis shows speed-up and GPU utilization normalized to the baseline results. We define GPU utilization as the ratio of time during which at least one of the SIMD units in the GPU is active, to the total execution time. These data were collected on the Trinity system hardware described in Section 3.3.1 and Section 3.4.1, with corresponding DVFS table being shown in Table 1.

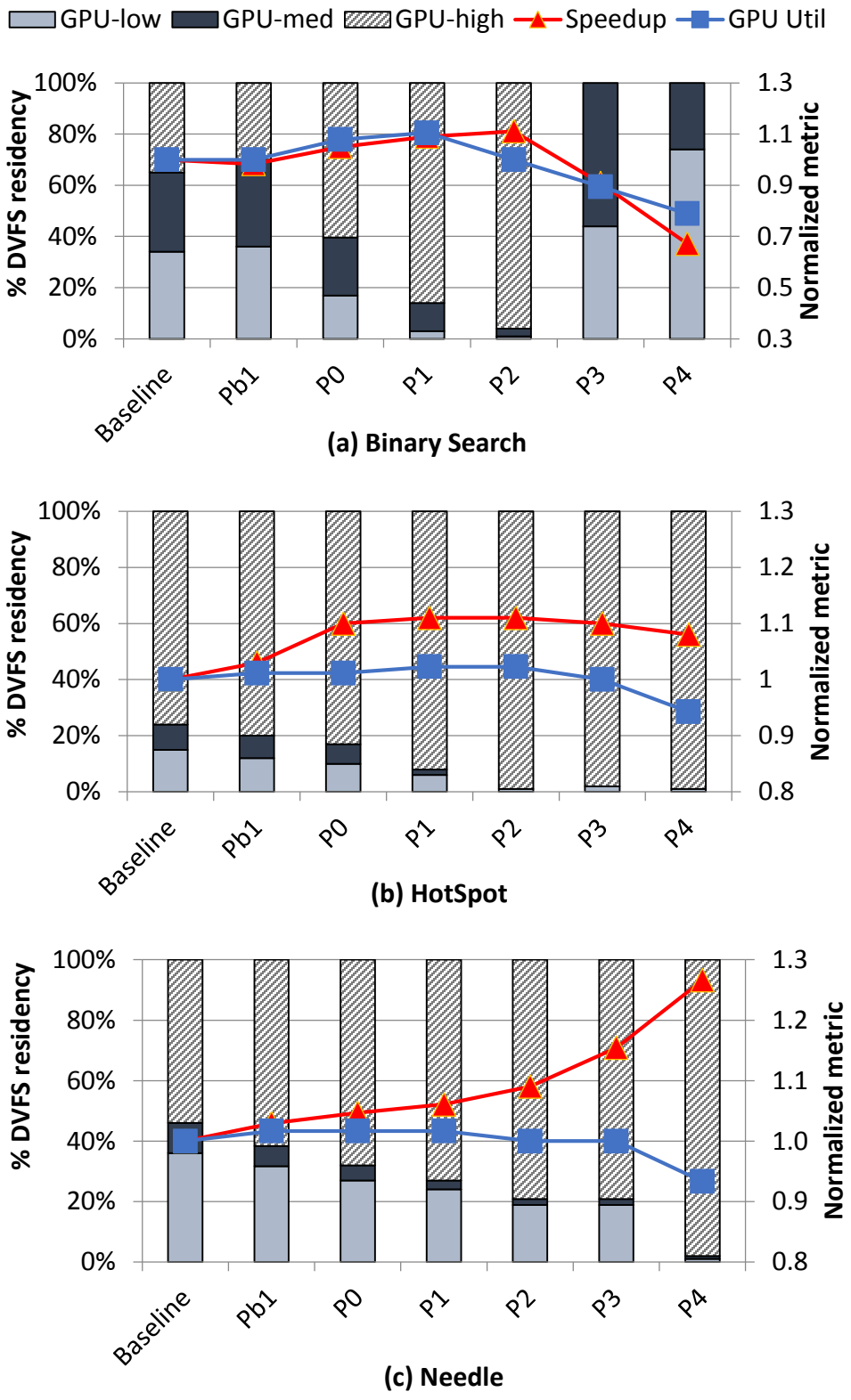


Figure 28: Impact of CPU P-state limit on performance, GPU residency, and GPU utilization.

For *Binary Search* and *HotSpot*, as the CPU maximum frequency decreases (moving to the right), the application transitions from being limited by thermal coupling to being limited by performance coupling. As the frequency decreases from Pb1 to P2, the GPU spends a larger portion of time in its higher-frequency performance states, indicated by GPU-high. In addition, speed-up increases, indicating that the GPU is utilizing the extra thermal headroom to improve performance. However, as the CPU frequency decreases beyond P2, we see a marked reduction in overall performance because performance coupling begins to dominate. GPU utilization decreases beyond P2, indicating that the GPU is being starved by the slower CPU. In the case of *HotSpot*, the thermal headroom permits the GPU to continue operating at its highest-performance state when active. However, for *Binary Search*, the local GPU power controller reduces the GPU frequency because of a significant drop in GPU utilization. For *Needle*, the GPU is thermally limited by the CPU across all P-state limits. Performance improves by 27% when the CPU operates at its lowest-performance P-state. However, performance coupling becomes dominant at a CPU P-state limit of P4 because GPU utilization starts to decrease.

Greedy vs. Needy Power Management: In Figure 29, we demonstrate further how CPU power impacts GPU performance for the application *Needle*. The figure shows the percentage of time spent in different GPU DVFS states and peak die temperature under various levels of CPU power during the entire program's execution. We see that for applications utilizing both CPU and GPU cores, as is the case with ***Needle***, greedy boost algorithms that consume all available thermal headroom can degrade rather than improve performance due to interactions between performance and thermal coupling. Boosting without accounting for these functional and physical dependencies can result in premature throttling of the GPU cores as seen in Figure 29a, which can negatively impact performance. In the case of ***Needle***, lowering CPU power by limiting CPU P- states eliminated premature throttling and improved performance (Figure 29b and Figure 29c),

but as seen from Figure 28, that may not always be the case. The apparent solution of throttling the CPU cores to mitigate thermal coupling effects becomes counterproductive if the CPU units become too slow to fully utilize the GPU, i.e., they are limited by performance coupling. Further, emerging applications are more likely to concurrently use the CPU and the GPU as first-class computational engines, increasing the importance of power-management solutions that balance performance and thermal coupling effects.

As is evident from the preceding analysis, during any time interval there is an optimal CPU operating frequency (and, equivalently, P-state) for each application depending on its thermal and performance coupling characteristics. We refer to this P-state as the *critical P-state* and the corresponding frequency as the *critical frequency*. We observe that the critical P-state is a time-varying function of the workload and our goal is to have the CPU always operating in the critical P-state. Our approach is to first define a measurable performance metric that is sensitive to the CPU and GPU frequencies. By tracking the behavior of this metric, we can periodically determine and set the CPU to its critical P-state.

Microsoft® Windows® OS Power Management [117] using ACPI provides a capability for managing the CPU P-state based on application requirements. However, in experiments with ACPI, the lower-performing P-states were never utilized. There are a number of shortcomings here. First, the OS uses the highest utilization among all cores as the metric to determine the P-state of all cores, while most of the applications analyzed have varying degrees of core utilization. Second, ACPI does not consider the performance requirements of an application, while our analysis shows that applications experience phases that require higher CPU performance. Finally, ACPI does not include any concept of performance coupling or thermal coupling; therefore, it cannot be used readily to regulate to either requirement.

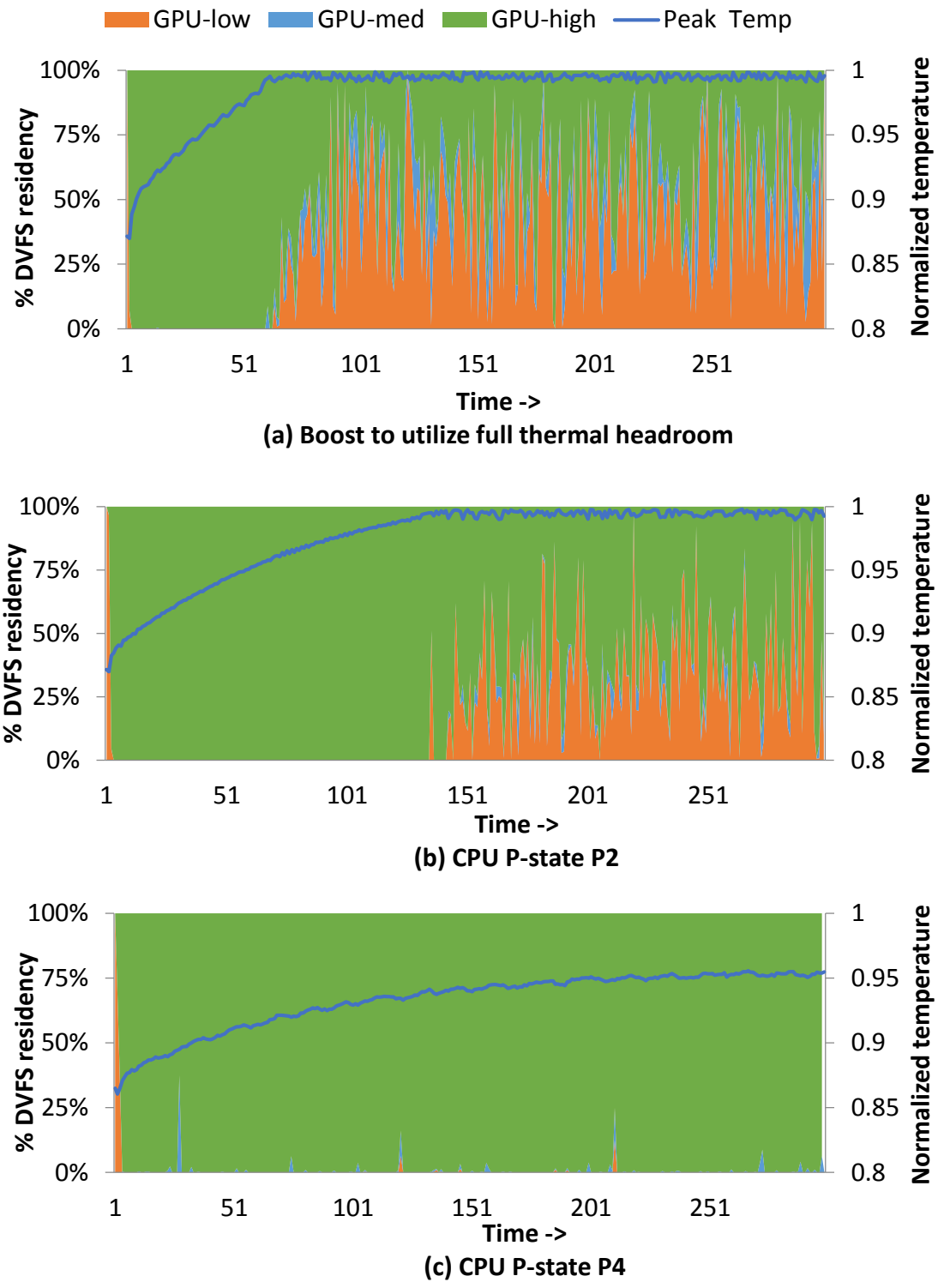
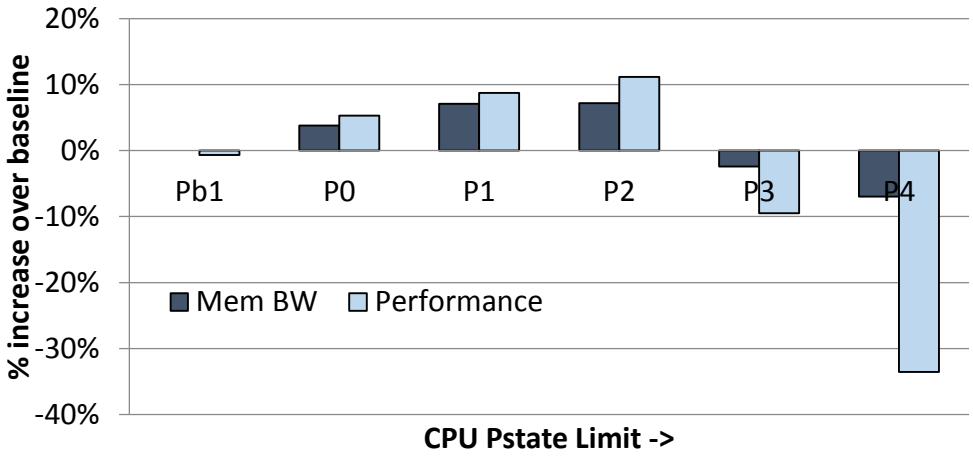


Figure 29: Thermal throttling in Needle with greedy boost algorithm and CPU P-state limiting. GPU-high, GPU-med, and GPU-low refer to high medium and low GPU operating frequencies, respectively.



(a) Binary Search

(b) Hotspot

(c) Needle

Figure 30: P-state limit effects on GPU memory bandwidth.

5.3 Run-time Metrics

In this work, we propose to use the gradient of the GPU memory access rate as a proxy for CPU-GPU performance coupling [68]. The GPU is a data parallel execution engine with massive thread-level parallelism and significant number of memory accesses. When the CPU transitions to a lower-performing P-state, the GPU frequency and hence the memory access rate increases due to decreasing thermal coupling effects. However, if the CPU operation drops below the critical frequency, the GPU is starved by the slower CPU and correspondingly the GPU memory access rate drops. This observation can be used as a starvation hint to transition the CPU to the critical P-state. In Figure 30, we illustrate the impact of P-state-limiting on GPU memory bandwidth for *Binary Search, HotSpot,* and *Needle*. Memory bandwidth tracks performance for these three benchmarks, indicating the applicability of this metric. In addition, to deal with phase changes in the applications that require high CPU frequency, we also use retired instructions per clock (IPC) of the CPU as a measure of the application's sensitivity to CPU frequency [49].

5.4 Cooperative Boosting (CB)

Based on the preceding analysis, we see that the power-management problem is one of determining the critical P-state – the state that mitigates the negative effects of thermal coupling while providing sufficient power for performance-coupled operation. This section describes our CB algorithm for the dynamic determination of the critical P-state. Figure 31 shows the architectural block diagram of the implementation of our scheme.

5.4.1 Structure

The CB algorithm operates as a decision layer on top of the baseline Trinity power-management system (from now on referred to as the baseline). The baseline was designed to optimize the average case behavior across a wide range of applications with a fair allocation of power to both the CPU and the GPU by utilizing all of the available thermal

headroom. The CB algorithm enhances such thermal headroom-based management techniques by tailoring its behavior to i) the asymmetry of the thermal behavior of the CPU and GPU and ii) the phase behavior of applications – specifically, thermal and performance coupling over short intervals. Such optimization becomes increasingly important as emergent workloads are making more balanced use of the CPU and the GPU. The goal of CB is to determine when thermal coupling effects are detrimental and to set the critical P-state limit under such cases. This is the highest-performing P-state the baseline system is permitted to use for the CPU. The voltage and frequency for the GPU is managed by the baseline and is not directly managed by CB; thus, CB essentially controls the CPU limits under which the baseline power-management system operates. Figure 32 illustrates the CB algorithm flow.

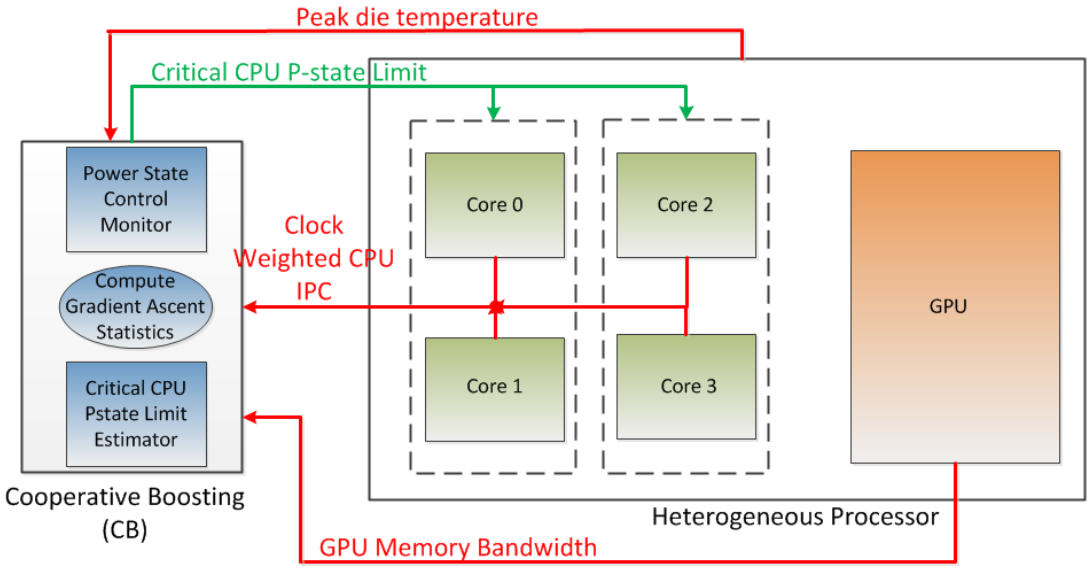


Figure 31: Block diagram of the Cooperative Boosting (CB) framework.

CB monitors temperature and performance metrics at intervals of 10 ms and modifies the CPU P-state limit at intervals of either 10 or 500 ms to account for frequent workload phase changes and relatively slow thermal response times, respectively. Periodic

Cooperative Boosting Algorithm

At beginning

```
If (Peak_Temp > Temp_Threshold) {
   EnableCB();
   Prev_PStateLimit=P0;
}
```

Every 10 ms

```
for i=0, i<Core_Count; i++ {
   IPC[i] = ReadIPC(i);
   Active_Clks[i] = ReadActiveCoreClock(i);
}
Weighted_IPC = ComputeWeightedIPC(IPC,Active_Clks)
IPC_Gradient = Weighted_IPC – Prev_Weighted_IPC
Prev_Weighted_IPC = Weighted_IPC;
Peak_Temp = ReadPeakTemp();
GPU_Mem_BW = ReadGPUMemBW();
Short_Term_BW = ComputeShortTermBW();
Long_Term_BW = ComputeLongTermBW();

If (CB_Enabled && (IPC_Gradient > =IPC_Threshold)) {
   Prev_PstateLimit = CPU_PStateLimit;
   UnsetPStateLimit();
}
```

Every 500 ms

```
If (CB_Enabled && (IPC_Gradient < IPC_Threshold)) {
   CPU_PStateLimit = Prev_PStateLimit;
   BW_Gradient = Short_Term_BW – Long_Term_BW;
   If (BW_Gradient >= BW_Threshold) {
      Last_Good_PState = CPU_PStateLimit;
      CPU_PStateLimit++; /* Until P4 is reached */
   }
   Else {
      CPU_PStateLimit = Last_Good_PState;
   }
}
```

Figure 32: Cooperative boosting algorithm.

enforcement of P-state limits at short intervals can cause dampening of natural workload behavior, whereas long intervals can cause inaccuracy in measurements. Therefore, we decouple the monitoring and the control intervals in CB. These intervals are chosen

carefully to account for the long thermal rise times, shorter performance intervals and workload activities, and overheads of P-state change. In practice, the intervals can be adjusted based on RC time constants of the die, floor plan, and process technology. The critical P-state limit can be computed at any granularity (per core, per CU, or for the entire CPU). In this paper we apply the same critical P-state limit to all CPU cores due to a shared voltage plane in the Trinity system we used for our measurements

5.4.2 Algorithm

The CB algorithm operates in three major steps: i) being invoked, ii) determining and setting the critical P-state limit, and iii) damping control to prevent oscillations. In the beginning, the processor starts with the highest-performance-boost P-states for the CPU and the GPU, and power and temperature are managed by the baseline. At intervals of 10 ms, we determine if the processor is thermally limited and if CB should be applied. If so, power management moves into CB mode.

The second step is determining which CPU P-state limit to apply in CB mode. This involves both instrumentation and decision-making. CB samples peak die temperature, per-core retired IPC, and memory bandwidth usage at every 10-ms monitoring interval. Note that although the algorithm in Figure 32 and the discussion in Section 5.3 refer to the GPU memory bandwidth, the implementation of CB uses a combined CPU and GPU bandwidth measurement since hardware restrictions prevent GPU-only bandwidth measurements while CB is enabled. Through evaluation, we found that this did not hinder the performance of the algorithm due to the overwhelming dominance of the GPU in memory bandwidth usage.

Each core's IPC is weighted by the number of active clock cycles seen by the core during the sampling period, and the aggregate IPC for the CPU is the sum of the weighted IPCs for all four cores. For memory bandwidth, in addition to monitoring memory bandwidth at each 10-ms interval, CB also keeps a short-term and a long-term moving average of memory bandwidth to track how the bandwidth changes over time. Because P-

state limiting to reduce thermal coupling effects is made at intervals of 500 ms, the short-term average is computed over the last 500-ms interval while the long-term moving average is computed over the last five such intervals. Bandwidth gradients are computed by comparing the short-term moving average with the long-term moving average.

The CPU P-state limit may be established by observing changes in the CPU IPC or GPU memory bandwidth (these metrics were advocated as proxies to detect performance-coupled operation in Section 5.2). CB utilizes the gradient of memory bandwidth to determine the critical P-state for the CPU. If the gradients are positive, then the workload benefits from shifting power to the GPU. In this case, the algorithm moves the CPU P-state limit to a lower performance state. The converse occurs when the gradient is negative. Over time, the controller is trying to move the CPU to the critical P-state.

If the workload enters a CPU compute-intensive phase, as indicated by a high-CPU IPC phase, the current P-state limit is saved and the control part of CB is suspended by disabling P-state limiting. The check for CPU IPC changes occurs at 10-ms intervals to capture frequent phase changes and data dependencies. When the CPU workload exits the compute-intensive phase, CB operation is resumed at the saved P-state limit. This dampens multiple transitions through performance states arising from a short burst of high-power CPU phases that would otherwise re-initialize the CPU performance state to the highest performance state. Finally, to prevent oscillation between a pair of P-state limits, we employ a damping mechanism such that a new P-state limit is weighted towards the previous P-state limit after more than a threshold number of transitions.

To encompass non-performance-coupled applications that may have a constant CPU IPC (such as SPEC CPU2006 applications), we use an absolute average IPC in conjunction with IPC phase changes for CPU-centric workloads with no activity on the GPU. Although CPU-centric workloads are not the focus of this paper, we show that our CB algorithm can sometimes improve the performance of these applications by limiting the performance state when the application is memory-bound.

5.5 Experimental Set-up

We perform all measurements and analysis on an AMD A8-4555M Trinity APU with 19W TDP. Base CPU frequency is 1.6 GHz, with AMD Turbo CORE frequency up to 2.4 GHz. The CPU DVFS state table is shown in Table 1. The GPU frequency is 320 MHz with AMD Turbo CORE frequency of 423 MHz [130]. We use four, 2-GB DDR3-1600 DIMMs. Hardware performance counters for IPC, memory bandwidth, etc., are monitored using performance libraries running in Windows OS. A maximum cap on the CPU P-state limit is implemented using model-specific registers as described in [12].

Table 1: HW- and SW-managed DVFS states for the CPU compute units on the Trinity A8-455M APU.

	P-state	Voltage (V)	Freq (MHz)
HW Only	Pb0	1	2400
	Pb1	0.875	1800
SW-visible	P0	0.825	1600
	P1	0.812	1400
	P2	0.787	1300
	P3	0.762	1100
	P4	0.75	900

We evaluate three different boost algorithms. The baseline is the BAPM algorithm, which is the state-of-the-practice algorithm in the Trinity power-management system described in Section 3.4.1. The second is the CB algorithm described in Section 5.4.2. Third, we evaluate the behavior of a static P-state-limit algorithm in which a fixed P-state limit is applied throughout the entire run of the application. This means that the CPU can enter a lower-performing (but not higher) P-state than the P-state limit. We refer to this as

the static PX limit scheme, where PX is one of the performance states (e.g., P1, P3, etc.). For CB, P-state limits are applied according to the algorithm described in Section 5.4.2. Although CB can be implemented in any layer such as hardware, power-management firmware, or system software, we implement CB at the system software level by layering it on top of the baseline.

Table 2: Summary of benchmarks used for CB evaluation.

BM (Description)	Problem Size	Type
NDL **(Needleman-Wusch [18])**	4096x4096 data points, 1K iterations	GPU
LUD (**LU decomposition [19])**	512x512, data points, 500 iterations	GPU
HS **(HotSpot [18])**	1024x1024 data points, 100K iterations	GPU
SRAD **(Image Proc [18])**	502x458,500K iteration	GPU
BF (**BoxFilter SAT [2])**	1Kx1K input image, 6x6 filter,10K iterations	GPU
MM **(Matrix Mult [2])**	2Kx2K, 10K iterations	GPU
FAH **(Folding at Home [42])**	Synthesis of large protein: spectrin$	GPU
CFD **(Computational fluid dynamics [18]**)	200K elements, 20K iterations	GPU
BFS **(Breadth first search [15])**	1M nodes, 1K iterations	GPU
BS **(Binary Search [2])**	4096 inputs, 256 segments, 1M iterations	GPU
KM **(Kmeans [18])**	819200 points, 34 features, 1K iterations	Mixed
BP **(BackProp [18])**	252,144 input nodes, 10K iterations	Mixed
Viewdle **(Haar facial recognition [114])**	Image 1920x1080, 2K iterations	Mixed
Mcf **(CPU2006 [109])**	4 threads, Ref input	CPU
Lbm **(CPU2006 [108])**	4 threads, Ref input	CPU
Perl **(CPU2006 [109])**	4 threads, Ref input	CPU
Pvr (**CPU2006 [109])**	4 threads, Ref input	CPU
Gcc **(CPU2006 [109])**	4 threads, Ref input	CPU

For CPU and GPU power and temperature, we use the digital estimates provided by the power-management firmware running in the Trinity system, accuracies for which are described in [83]. For all schemes, we run the benchmarks for at least a few minutes to reach a thermally stable steady state. A fixed-time cool-down period is applied before each run to eliminate any variations in starting temperature. We also run many iterations of the application and take an average across those to eliminate run-to-run variance in our hardware measurements.

We use 18 applications, summarized in Table 2. These are a mix of both state-of-the-art and emergent applications. Eight of them are from Rodinia (*NDL, LUD, HS, SRAD, CFD, BFS, KM,* and *BP* [18][19]), three are from the AMD APP SDK (*BF*, *MM,* and *BS* [2]), two are stand-alone (*FAH* [42] and *Viewdle* [114]), and five are from SPEC CPU2006 (*Mcf, Lbm, Perl, Pvr*, and *Gcc* [109]). We selected the applications to represent i) GPU-centric operation (where GPU is used as a compute accelerator with CPU feeding the data to the GPU), ii) CPU-GPU mixed workloads (where computation is more balanced between CPU and GPU although the fraction of work division may not be the same), and iii) CPU-centric workloads (where computation is done only on the CPU and the GPU is unused). All GPU applications execute one or more parallel kernels for multiple iterations to reach steady-state thermals. The SPEC CPU applications are run with four threads, one on each core.

We report performance, power, and energy efficiency as defined by the energy-delay^2 product (ED^2) [67]. We show all values normalized to the baseline scheme, which is the default Trinity power-management system. Average total power (CPU and GPU) and average energy efficiency are also measured over the entire run-time of an application.

5.6 Results

In this section, we present performance, power, and ED^2 results for CB and the static P-state limit algorithm. All results are shown relative to the baseline BAPM

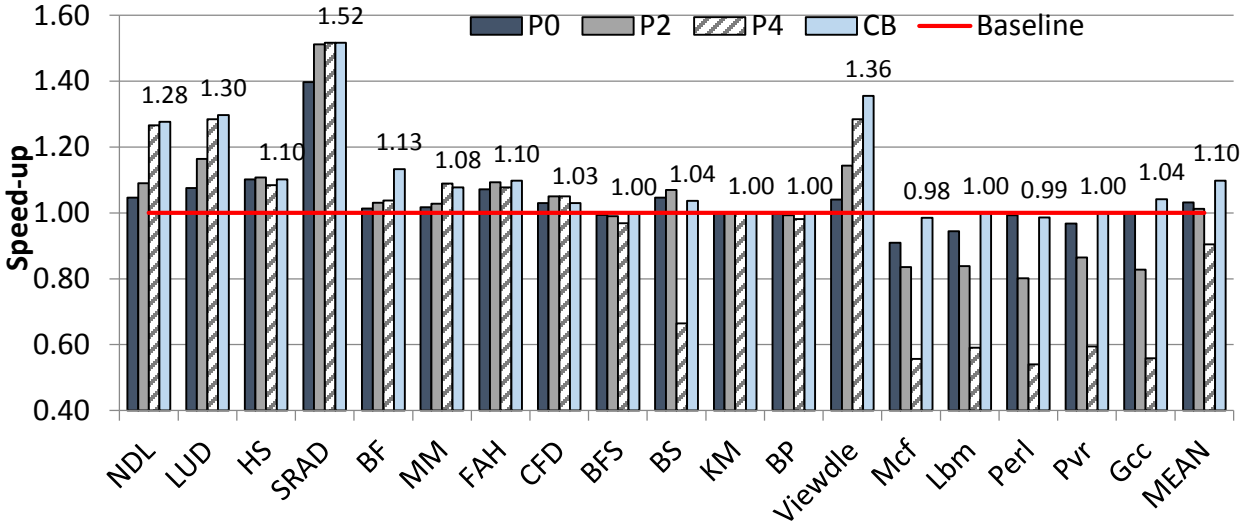


Figure 33: Performance results with static limits and CB.

algorithm described in Section 3.4.1, and all performance and power numbers are measured results from running the applications in Table 2 on real hardware.

5.6.1 Performance

Figure 33 illustrates the speed-up of CB and static schemes. Across the 18 applications, we see a 10% speed-up with CB, a 3% speed-up with P0 (the highest-performance software-visible P-state), a 1% speed-up with P2, and a 10% performance loss with P4. For the performance-coupled workloads (i.e., GPU-centric and CPU-GPU mixed workloads), the average speed-up with CB is 15%. The static schemes clearly demonstrate good performance gains compared to the baseline for certain types of workloads but impose a high performance penalty for others, motivating the need for dynamic schemes.

GPU-centric applications such as *NDL, LUD, MM,* and *SRAD* improve in performance compared to the baseline with both CB and static. In general, these applications have low CPU IPC and are not very sensitive to CPU performance in the frequency ranges explored. Both CB and static P4 limiting show comparable gains, with performance improvement as high as 52% in *SRAD*. Thermal coupling dominates these

applications at all CPU frequencies because they have high activity in the GPU and, hence, high power requirements. The critical P-state for the CPU is at a lower frequency than the lowest P-state P4 available in our part. These workloads reach the peak temperature quickly, and high-performance CPU P-states result in excessive thermal throttling without a commensurate application performance improvement.

Similar thermal coupling effects occur in applications such as *HS, BS,* and *FAH*. However, here we reach the critical CPU P-state before the lowest P-state limit of P4. At P2, thermal and performance coupling effects are balanced and we see the maximum performance gains. Decreasing CPU frequency beyond P2 causes performance coupling to dominate over thermal coupling and degrades performance by 3%, 34%, and 1%, respectively, for *HS*, *BS*, and *FAH* at P4. CB achieves comparable results to the critical P-state of P2.

Applications such as *KM, BFS, BP,* and *CFD* see minimal to no benefits compared to the baseline with static or CB schemes. *KM, BFS,* and *BP* never reach the peak junction temperature, and so CB never invokes P-state limiting. Although *KM* has high-IPC phases, it is primarily memory-bound and its performance stays relatively flat with CPU frequencies. *BP* has serial phases between parallel kernels requiring significant CPU-GPU communication. *BFS* has a high control flow divergence with low GPU activity. In both *BFS* and *BP*, CB results in the same performance as the baseline, whereas static P4 limit shows performance degradation up to 3% due to performance coupling. Although *CFD* is heavily memory-bound, it reaches the peak temperature due to high activity and a relatively high compute-to-memory ratio in the GPU; as a result, it shows a slight improvement of 3-5% compared to the baseline scheme using static limiting and CB. Performance gains from reducing thermal coupling effects flatten out beyond P2 as the memory-related stall time of the kernel starts to dominate.

In balanced workloads such as *Viewdle*, a face-recognition application, both the CPU and the GPU are utilized heavily for computation. Thermal coupling is dominant at

the higher CPU frequencies, and so static P-state limiting to both P2 and P4 improves performance compared to the baseline. CB, however, outperforms all static P-state-limiting schemes by dynamically adjusting to the critical P-state based on application needs. *Viewdle's* IPC varies periodically from low to high, and it is sensitive to CPU frequency during high-IPC phases. CB dynamically shifts power to the CPU during high-IPC phases and to the GPU during low-IPC phases, thereby limiting the impact of thermal coupling while providing the required power for performance coupling. Section 5.6.2 provides further insights in *Viewdle's* performance. We see similar behavior with *BF*, which is an image-filtering application with frequent CPU communication phases between the horizontal and vertical passes in the image blur filter. CB performs 13% better than the baseline and 9%-12% better than any of the static schemes in the case of BF.

Finally, we analyze the performance of CPU-centric, non-performance-coupled applications such as *Perl* and *Pvr*. As we see in Figure 33 the baseline does very well for these workloads and static limiting significantly degrades performance. CB largely performs as well as the baseline, indicating that CB is a well-rounded approach for multiple usage scenarios. Although analyzing multiple non-performance-coupled applications (e.g., a CPU-centric app and a GPU-centric app) running together was not the focus of our research, we believe CB will perform as well as or better than the baseline because CB tries to limit CPU power only when it is not needed.

5.6.2 Thermal and Performance Coupling Analysis

In Figure 34 we illustrate how CB mitigates the effects of thermal coupling in the case of *BS*. The y-axis indicates the measured peak temperature normalized to Tjmax. With a static limit of P4, the application heats the chip to a value less than the peak. CB, on the other hand, does not initially restrict the baseline algorithm; instead, it tries to find the critical P-state for the CPU once we approach the peak temperature threshold. As power is shifted from the CPU to the GPU, peak die temperature decreases because the GPU is able to sustain a higher power boost for a longer period due to its lower thermal density, as

described in CHAPTER 4. Further, the effects of thermal coupling become less dominant because the CPU is running at a lower P-state. As a result, the GPU residency in the high-performance state increases significantly compared to the baseline, thereby improving application performance. Moreover, the short variations in temperature result from the fact that CB constantly adjusts the critical P-state based on workload phases. This helps balance performance and thermal coupling effects.

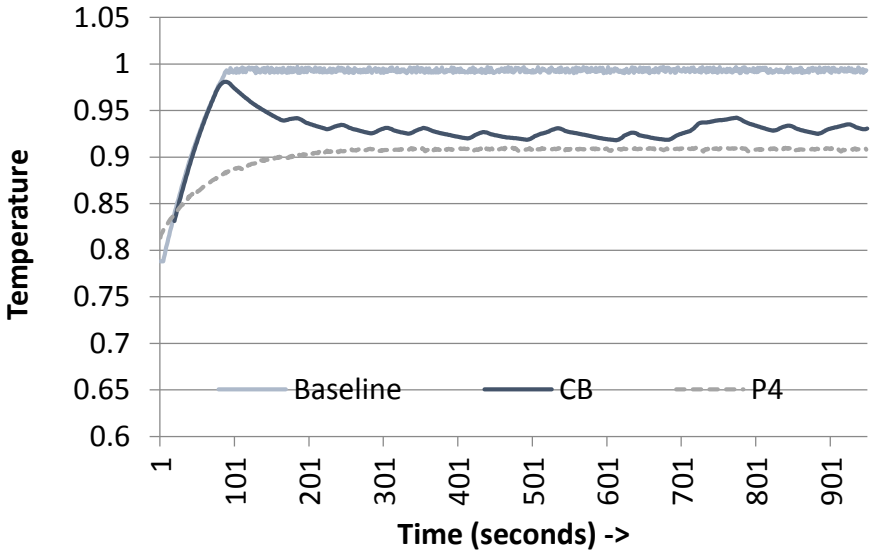


Figure 34: Thermal behavior of Binary Search with CB.

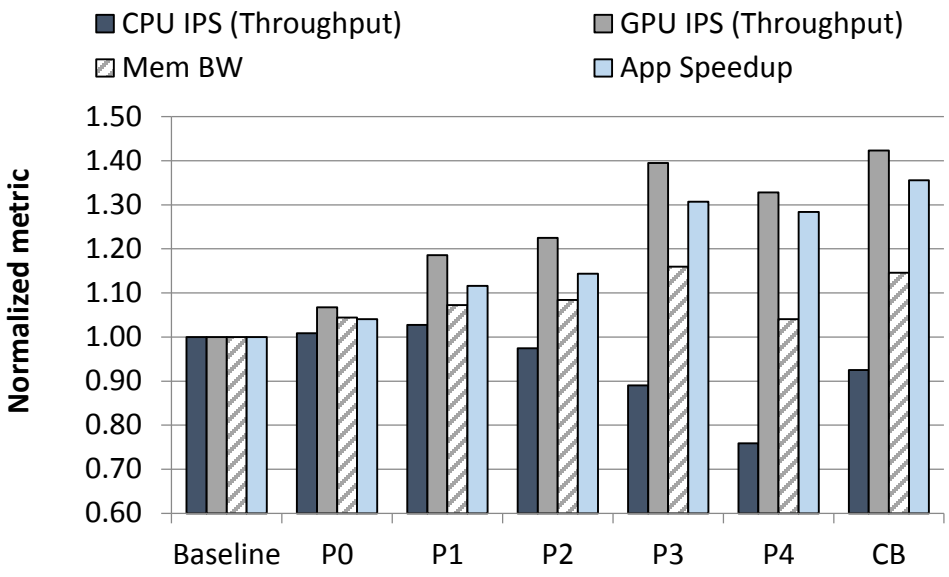


Figure 35: Viewdle performance analysis with CB.

Figure 35 provides further insights into the performance of *Viewdle* in terms of instructions per second (IPS), memory bandwidth, and speed-up. As we apply CPU P-state limiting with lower-performing P-states, CPU IPS understandably drops. However, the GPU IPS continues to increase, and so does utilized memory bandwidth due to the GPU's ability to sustain higher frequencies because of the reduction in thermal coupling. For P-state limiting beyond P3, both GPU throughput and memory bandwidth drop due to performance coupling effects. However, with CB, the CPU P-state limit is managed dynamically to balance performance and thermal coupling effects: GPU throughput and speed-up increase by 42% and 36%, respectively, compared to the baseline.

In Figure 36, we illustrate how CB mitigates the effects of thermal coupling when running *Needle*. The left-side y-axis shows GPU residencies in the different performance states. The right-side y-axis shows the measured peak temperature normalized to Tjmax. In the baseline case (Figure 36 (a)), we see a considerable residencies in the medium and

low GPU frequencies once temperature reaches the steady state to maintain performance within the maximum thermal limits. GPU frequency throttling occurs because of thermal coupling and heat transfer effects from the CPU to the GPU as both CPU and GPU are run at their maximum frequencies during the initial ramp-up stage due to availability of thermal headroom. However, as shown in Figure 36 (b), CB tries to find the critical P-state for the CPU once we approach the peak temperature threshold. Once invoked, CB starts shifting power to the GPU. Because *Needle* is a high-power workload, we see a slight temperature-based throttling initially, after which the temperature decreases and power shifts from CPU to GPU. This allows boosting of the GPU to higher frequencies for a much longer period, thereby improving application performance.

Because CB is designed to mitigate detrimental effects of thermal coupling in thermally limited situations, it effectively lowers the peak operating temperature of the processor opportunistically compared to the baseline (2% lower on average across all applications). Although temperature is not a direct optimization goal for CB, lower peak temperatures have many additional benefits: i) increased TDP power budget to achieve more performance within a fixed thermal envelope; ii) lower cooling costs within a fixed power budget; iii) lower leakage power and, hence, lower overall energy; and/or, iv) improved reliability through increased mean-time-to-failure rates.

5.6.3 Power and Energy

The power saving achieved with CB over the baseline is illustrated in Figure 37, which shows an average power savings of 8% across all applications and an average of 10% across performance-coupled GPU-centric and mixed workloads. These power savings

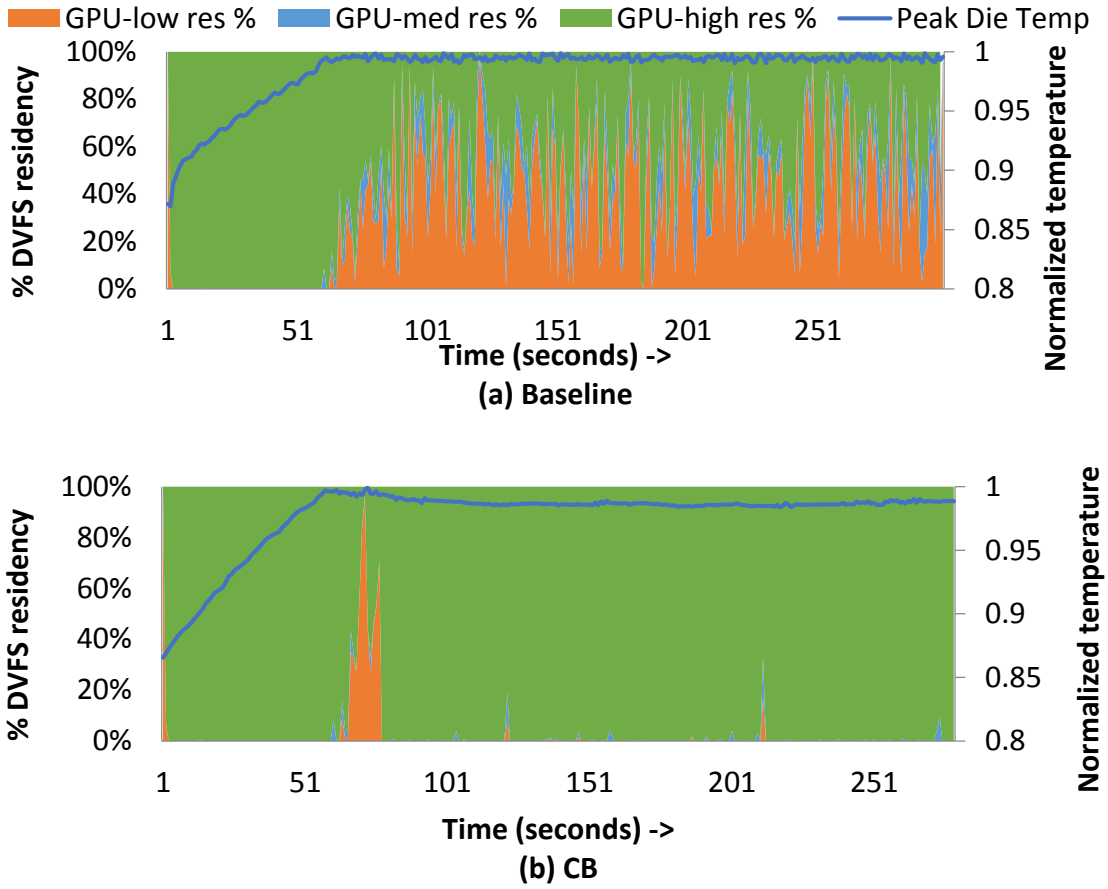


Figure 36: Thermal throttling in Needle with CB.

come from two factors: 1) CPU power consumption is reduced due to its operating at the critical frequency, 2) as a result of the distinct thermal signatures between CPU and GPU and a higher power efficiency of the GPU, only a fraction of the CPU power savings is utilized towards boosting the GPU to its highest operating frequency. Highest power reduction is seen in *BS*, where we see a 5% reduction in average peak temperature and,

hence, leakage power during run-time. *BFS*, *BP*, and *KM* never reach their peak temperatures, so power savings are minimal because CB does not limit P-states under such cases and allows both CPU and GPU to take full advantage of boosting. We also achieve a small amount of power savings in the SPEC CPU2006 workloads, up to 11% with *Mcf* because CB continuously tracks high-IPC compute-bound phases. When the workload encounters memory-bound phases, a P-state limit is applied to lower the frequency; this limit has little to no performance impact but it saves power [49].

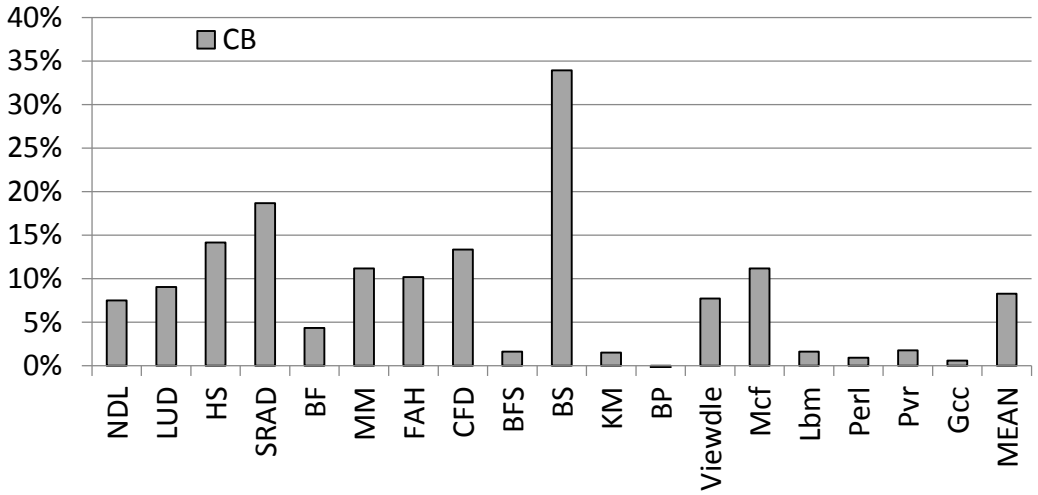


Figure 37: Reduction in power for CB relative to baseline.

Figure 38 shows the ED^2 product (lower numbers signify improvement over the baseline). With CB, we see an average energy-efficiency improvement of 25% (up to 76%) across all applications, and 33% across performance-coupled GPU-centric and mixed CPU-GPU workloads. Interestingly, a static limit of P4 (the lowest-performing P-state) performs 30% worse than the baseline, but we see an improvement of about 10% with static limits of P0 and P2 due to reduction in thermal coupling and a large reduction in power at those states. However, as shown in Figure 33, a fixed static P-state of P0 and P2 results in

significant performance outliers for CPU-centric workloads; hence, it is not a viable solution. CB, however, can achieve similar or better results for performance and energy efficiency than any static scheme without requiring any offline profiling or user intervention.

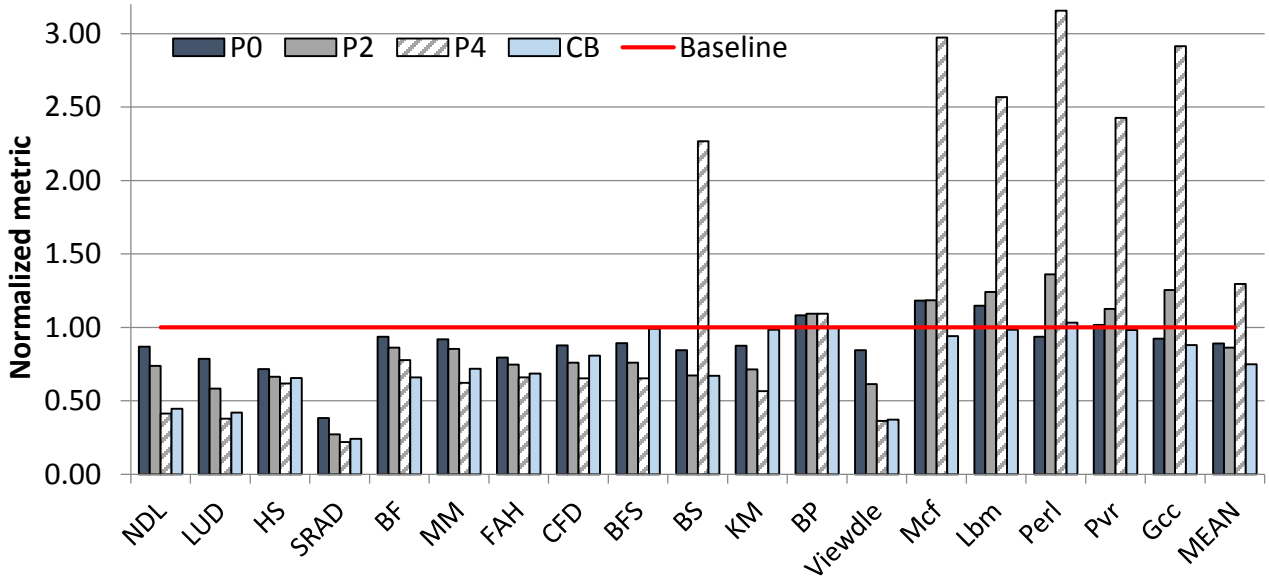


Figure 38: Energy-delay^2 product normalized to baseline.

5.6.4 Summary

In this section, we summarize our results and insights. First we show that workloads with high GPU activity are more sensitive to thermal coupling with the CPU. The baseline can degrade performance while both CB as well as static P-state limiting shift a greater portion of the power to the GPU, reduce thermal coupling, and improve performance. For applications with tight performance coupling with the CPU, CB finds the critical P-state and thus performs better.

For applications with very low GPU utilization such as those with high control flow divergence, thermal coupling may not be a factor since these workloads tend to run much

cooler. While the baseline does not hurt performance, static schemes can degrade performance significantly by amplifying the low GPU utilizations when the CPU P-state is fixed below the critical P-state. However, CB dynamically detects when an application is not thermally limited and stops limiting CPU's P-state under such cases. This allows CB's performance to track baseline for such workloads.

Balanced workloads that actively utilize both the CPU and GPU are particularly susceptible to thermal coupling effects. CB outperforms the baseline and static schemes by continuously tracking the time-varying critical P-state during execution. CB uses only the power it needs, and thus reducing thermal coupling without impacting performance coupled operation. This is one of the fastest growing classes of future workloads [75][114].

For non-performance coupled CPU-centric workloads, greedy boosting approaches work well while static schemes understandably perform poorly since performance scales with frequency. CB performs largely as well as the baseline since the critical P-state tends to be the highest performance state. However, CB delivers slightly better performance for memory bound workloads by detecting memory bound phases and adjusting the critical P-state, which builds up thermal credits for compute phases that need higher performance state.

In summary, CB is a well-rounded technique that can be used to dynamically manage power, performance and thermals across a wide range of applications. Although for a given application one can profile the critical P-state limit statically offline, such an approach is impractical and often detrimental if the goal is to support a variety of applications including emergent and as yet unanticipated ones. CB improves over current headroom based greedy boost algorithms by balancing the needs and dependencies of CPU and GPU performance with the effects of thermal coupling.

5.7 Conclusions

This chapter addressed the complex relationships between power, thermals, and performance in a heterogeneous system running diverse applications. We described and explored thermal entities with varying thermal signatures and demonstrated the relationship between thermal coupling and performance coupling through detailed empirical analysis. Based on our analysis, we proposed a cooperative boosting solution that balances the effects of thermal coupling with the requirements of performance coupling to determine the critical frequency of operation. We compared the CB algorithm with a state-of-the-practice boost algorithm and static power-limiting methods for a varied set of homogeneous and heterogeneous benchmarks. We showed on hardware that CB achieves an average 10% speed-up (up to 52%) and an average 8% power reduction (up to 34%) compared to the other algorithms, resulting in a 25% (up to 76%) improvement in the ED^2 product. *This research work was published at International Symposium of Computer Architecture (ISCA) 2013 [89]*.

CHAPTER 6

CPU-GPU PERFORMANCE COUPLING MANAGEMENT

In CHAPTER 4 we characterized the interactions between the CPU and GPU emphasizing the coupled nature of their interactions – both direct coupling via programming model and algorithmic interactions and indirect via interference at shared hardware resources. Our concern now is making this coupled operation as energy efficient as possible with minimal performance impact. This chapter examines the challenge of improving energy efficiency of a heterogeneous processor consisting of an integrated CPU-GPU for high-performance computing (HPC) applications. This component of the research work is targeted towards the mainstream high performance computing space, which has uncompromising performance requirements while remaining subject to the goal of minimizing energy consumption. Consequently the goal here is to maximize energy efficiency with minimal to no compromises in performance on tightly coupled heterogeneous architectures.

6.1 Overview

Efficient energy management is central to the effective operation of modern processors in platforms from mobile to data centers and high-performance computing (HPC) machines. However, HPC systems are unique in their uncompromising emphasis on performance. For example, the national roadmap for HPC now has the goal of establishing systems capable of sustained exaflop (10^{18} flops/sec.) performance. However, the road to exascale is burdened by significant challenges in the power and energy costs incurred by such machines.

Many current HPC systems use general-purpose, multi-core processors such as Xeon from Intel and AMD Opteron™ that are equipped with several power-saving

features, including dynamic voltage and frequency scaling (DVFS). More recently, driven in part by demand for energy efficiency, we have seen the emergence of such processors with attached graphics processing units (GPUs) acting as accelerators. As of November 2012, four of the top ten and 62 of the top 500 supercomputers on the Top500 list were powered by accelerators [128][129].

As shown in CHAPTER 4 the tighter integration of CPUs and GPUs in heterogeneous processors along with the emergence of companion programming models is leading to greater performance dependencies between the CPU and the GPU. For example, CPU and GPU memory accesses interact and interfere in the memory hierarchy, while they share a chip-level power budget and thermal capacity. In CHAPTER 5 we addressed the problem of maximizing performance while operating under the constraints of thermal coupling (under a fixed thermal capacity) and still meeting the requirements of performance coupling (performance dependency). In this chapter we focus on maximizing energy efficiency with minimal performance loss while still meeting the performance coupling requirements between the CPU and GPU integrated in the same package. An effective power management solution that determines the DVFS states of the CPU and the GPU must be cognizant of these performance coupling effects, their interrelationships, and their combined effect on performance, i.e., return (performance) on investment (frequency and power allocation).

To this end, this chapter makes the following contributions:

- We empirically characterize the CPU and GPU frequency sensitivity of compute applications in a performance-coupled architecture. The analysis exposes several opportunities for improving energy efficiency without degrading the performance of the application.
- We identify a key set of CPU and GPU run-time parameters that reflects the frequency sensitivity of the application and use regression techniques to construct an analytic model of frequency sensitivity.

- We propose *DynaCo* – a coordinated, dynamic energy-management algorithm using online frequency-sensitivity analysis to coordinate the DVFS states of the CPU and the GPU. DynaCo is implemented on a state-of-the-art heterogeneous processor.
- Using measurements on real hardware, we compare DynaCo to a commercial, state-of-the-practice power- and performance-management algorithm for several OpenCL Exascale proxy applications and other HPC applications, demonstrating that significant improvements in energy efficiency are feasible without sacrificing performance when CPU-GPU power management is coordinated.

The rest of the chapter is organized as follows: Section 6.2 analyzes and identifies behaviors that have a substantive impact on frequency sensitivity of the CPU and GPU in an integrated APU. Section 6.3.1 presents run-time metrics to capture these interactions, followed by a description of the power management algorithm, DynaCo, to balance the power allocation between performance-coupled CPU and GPU in Section 6.3.2. Finally, Section 6.4 provides the experimental setup and Section 6.5 presents a detailed implementation and evaluation of DynaCo on a modern heterogeneous processor

6.2 Sensitivity Analysis and Opportunities

Figure 39 shows the peak temperature normalized to the maximum junction temperature allowed for each CU and the GPU for *miniMD* as the application runs on a 100W TDP processor. Processors with such a thermal design power package are commonly found in HPC clusters [126]. Although temperature tracks power and inversely tracks performance, it never reaches the peak thermal limits. This means that the performance of the CUs and the GPU are not constrained by temperature, and therefore they generally run at their maximum frequency. However, just because they can run at their maximum frequency does not mean that they should; there has to be a reasonable return in performance for the increase in frequency and higher power.

We characterize this return on performance with the notion of frequency sensitivity

– a measure of the improvement in performance for a unit increase in frequency. Frequency sensitivity is a time-varying function of the workload on a target processor. In general, the frequency-performance function is unknown. Thus, the idea is to measure the frequency sensitivity of an application periodically and determine whether it is productive (efficient) to change the frequency. While Rountree et al. [102] developed a frequency-sensitivity predictor for homogeneous CPUs, the problem in APUs is more complex due to shared resources and subtle CPU-GPU interactions.

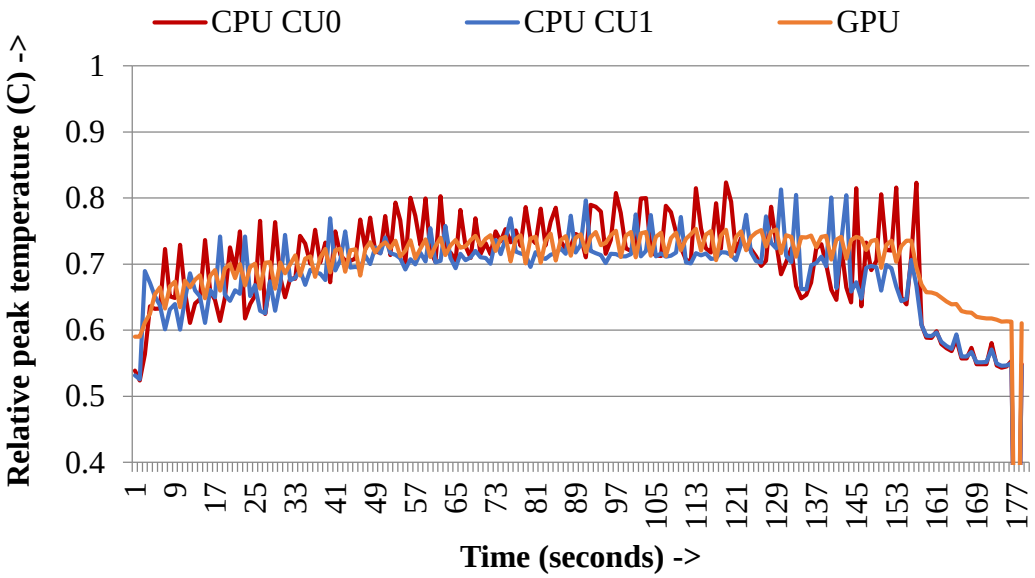


Figure 39: Thermal profile of miniMD running on GPU.

The rest of this section identifies and categorizes behaviors that have a substantive impact on frequency sensitivity of the components. All results are based on hardware measurements on an AMD A-Series APU, described in Chapter 3.3. This understanding is used in Section 6.3 to develop a model of frequency sensitivity for tightly coupled heterogeneous processors and to use the model to guide DVFS decisions.

6.2.1 Shared Resource Interference

The memory hierarchy is a key determinant of performance, and the CPU and the GPU share the Northbridge and memory controllers. The extent of interference at these points (which is time-varying) has a significant impact on the effectiveness of DVFS for the CPU or the GPU.

Figure 40 (left bar) breaks down the CPU and GPU memory access rates, normalized to peak-DDR bandwidth with 75% bus efficiency, of one of the main computation kernels (*neighbor*) in *miniMD* [15]. The kernel is run iteratively in the application for steady-state duration. Figure 40 (right bar) breaks down the average CPU DVFS state residency for the active CPU time under BAPM, which shows that the kernel DVFS residency is entirely in the hardware managed CPU boost states (Pb0 and Pb1). We observe that this kernel saturates the overall shared-memory bandwidth primarily due to the high rate of memory references from the GPU. The CPU portion of memory demand, which is captured by looking at last-level cache L2 miss rates, is relatively small. Further (not shown), the CPU IPC of this kernel is higher than a typical memory-bound application.

Power- and performance-management schemes that determine the CPU DVFS state independent of interference at shared resources might conclude that the CPU voltage-frequency can be boosted within thermal limits to improve performance. This is, in fact, what the BAPM algorithm does. However, the application performance is memory bandwidth-limited due to the GPU memory demands, so scaling up the CPU voltage-frequency has little performance benefit and will degrade energy efficiency (discussed in Section 6.2.3). *The lesson here is that we need online measurements of chip-scale global interactions to make good decisions regarding the CPU or the GPU DVFS state*.

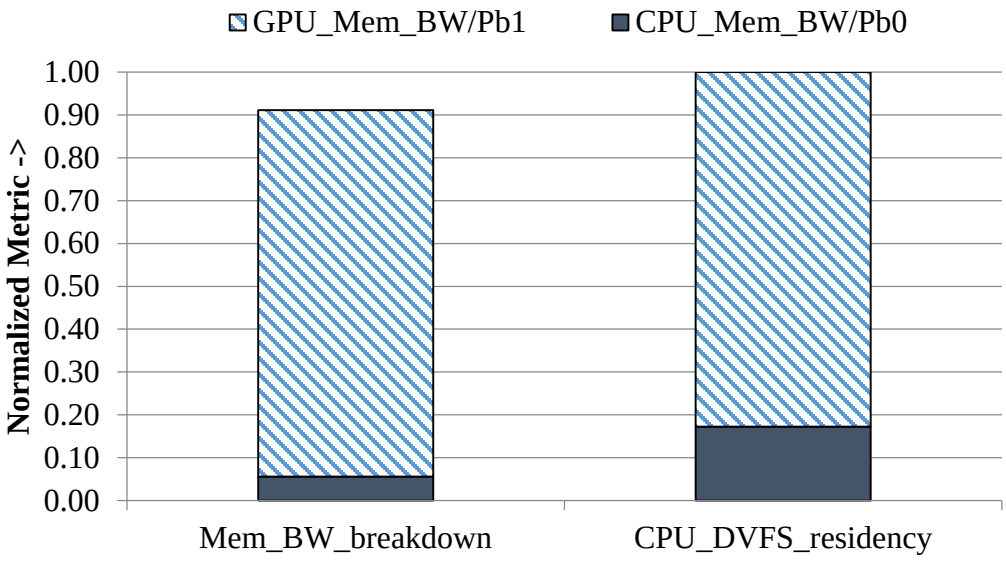


Figure 40: Break-down of memory interference between CPU and GPU and corresponding CPU DVFS residency.

6.2.2 Computation and Control Divergence

GPUs are exceptional execution engines for data-parallel workloads with little control divergence. However, performance efficiency degrades significantly with increasing control divergence. That does not imply that lower-frequency states should be used for control divergent applications. Consider the *Breadth-first Search (BFS)* graph application from the Rodinia benchmark suite [18]. Figure 41 illustrates GPU frequency sensitivity for *BFS* (left bar). Execution times are measured at the lowest and highest frequencies. We compute frequency sensitivity as the ratio of the difference in execution times to the difference in frequencies. The figure also shows the GPU ALU compute utilization (right bar). While GPU ALU utilization and computation are fairly low, GPU frequency sensitivity is quite high. This is due to the high control flow-divergent behavior of the kernels in *BFS*, which leads to low utilization. However, higher-frequency operation leads to faster re-convergence, and thus shorter execution time.

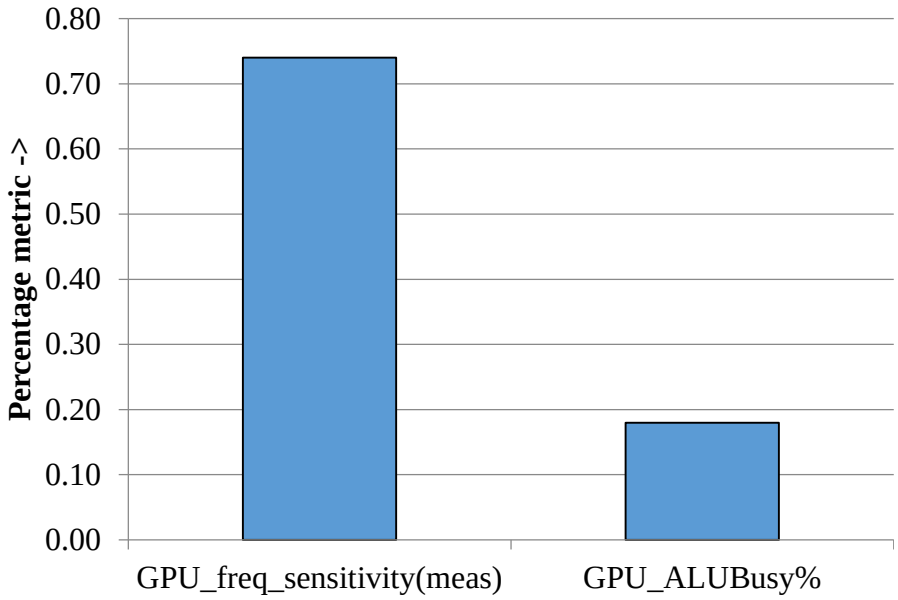


Figure 41: GPU frequency sensitivity to control divergence.

Conventional cores that extract instruction-level parallelism from a single thread correctly associate low IPC with low frequency sensitivity. The converse is true here due to the bulk-synchronous parallel-processing nature of GPU kernels. Control flow serializes the execution of threads in a thread block. The correct analogy with traditional core execution is the observation that higher-frequency operation will speed the serial sections of code and, therefore, the application as a whole. In this case, the greater the serial fraction or divergence, the greater the speed-up. *The lesson here is that control flow-divergence measures should be captured in the compute behavior when determining frequency sensitivity*.

6.2.3 Performance Coupling and Kernel Sensitivity

Each application has phases that vary in their frequency sensitivity due to the type of their activity rates and the degree of performance-coupling between CPU and GPU. This is also true of HPC applications. While computations are offloaded to the GPU, there are control and data dependencies between computations executing on the CPU and the GPU

cores. For example, for peak GPU utilization, the CPU must deliver data to the GPU at a certain rate; otherwise, the GPU will starve, resulting in a reduction in overall performance. Such performance-coupling between the CPU and the GPU cores is accentuated by the tighter physical coupling due to on-die integration and the emergence of applications that attempt a more balanced use of the CPU and the GPU. Hence, any cooperative energy-management technique must balance such interactions against energy/power savings.

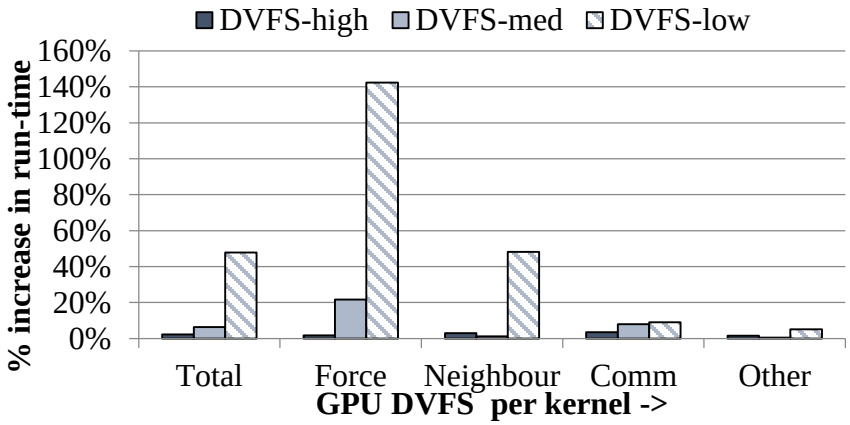


Figure 42: Percent increase in kernel run-time due GPU DVFS changes relative to the baseline (BAPM).

Figure 43: Percent increase in kernel run-time due CPU DVFS changes relative to the baseline (BAPM).

Here we evaluate the opportunities to save energy of an Exascale proxy application from the Mantevo suite called *miniMD* [15]. In particular, we characterize the frequency and resource sensitivity at the kernel granularity for both the CPU and the GPU power states. We have observed this behavior in other HPC applications as well; however we present only *miniMD* results here. Figure 42 illustrates the GPU frequency sensitivity for the main *miniMD* kernels by measuring the impact of frequency on the speed-up of each kernel. The x-axis records the GPU DVFS states for each kernel. The y-axis shows the increase in run-time from the baseline BAPM case as GPU frequency is reduced. Because we are not thermally limited, the baseline algorithm runs the GPU at the highest frequency.

We can observe many interesting behaviors in Figure 42, with the key insight being that different kernels in *miniMD* have different resource requirements and their relative sensitivities to GPU frequency reflect those needs. One of the main computation kernels, *Force*, scales very well with GPU frequency and performs the best at the highest-frequency GPU DVFS state. This is because of the heavy compute-bound nature of the kernel. The *Neighbor* kernel shows high sensitivity to GPU frequency when going from low to medium frequency; however, *Neighbor* sees little to no performance benefit at the highest GPU frequency because the *Neighbor* kernel becomes memory bandwidth-limited at the highest GPU frequency. Communication-limited and other fine-grained, relatively short kernels labeled "*Other*" seem to be less sensitive to GPU frequency. There is a 6% increase in total run-time at the medium GPU DVFS state, with the *Force* kernel being the main contributor to the slow-down.

Consider the frequency sensitivity of the CPU for each of the *miniMD* kernels executing on the GPU (recall the performance-coupling between the CPU and the GPU) illustrated in Figure 43. The *Force* and *Neighbor* kernels do not scale well with CPU frequency. The memory-bounded behavior of *Neighbor* makes it insensitive to CPU frequency with minimal performance loss at the lower CPU DVFS state of P4. The GPU

compute-intensive nature of *Force* makes it less dependent on CPU frequency; however, decreasing CPU frequency beyond P2 starts starving the GPU. On the other hand, fine-grained, shorter kernels such as *Communication* and *others* have higher data dependencies on the CPU and are tightly performance-coupled. Launch overhead, combined with the relatively small kernel timings compared to the actual execution time, make these kernels more tightly performance-coupled to CPU frequency and less GPU frequency-sensitive. *The lesson here is that the frequency-sensitivity metric in an APU needs to account for performance-coupling effects.*

Table 3: APU frequency sensitivity analysis of various performance metrics.

Metric	Description	Correlation Coefficient to GPU FS (meas)	Correlation Coefficient to CPU FS (meas)
WeightedALUBusy	ALUBusy weighted by GPUClockBusy.	0.85	-0.62
ALUInsts PTI	Compute instructions per thousand instructions.	0.78	-0.54
ALUBusy	The percentage of GPUTime ALU instructions are processed.	0.76	-0.54
ALUFetchRatio	The ratio of ALU to fetch instructions. If the number of fetch instructions is 0, then 1 will be used instead.	0.57	-0.31
L2 cache miss/cycle	Level 2 cache miss rate to main memory for CPU.	0.13	-0.41
ALUPacking	The ALU vector packing efficiency (in percentage).	0.11	-0.22
GPUClockBusy	GPU utilization: Ratio of time when at least one of the SIMD units in the GPU is active compared to total execution time.	0.06	-0.13
FetchUnitBusy	The percentage of GPUTime the fetch unit is active.	-0.28	-0.01
FetchUnitStalled	The % of GPUTime main memory fetch/load unit is stalled.	-0.49	-0.15
WriteUnitStalled	The % of GPUTime main memory write/store unit is stalled.	-0.51	0.12
Writes to memory PTI	Main memory writes per thousand instructions.	-0.60	-0.28
Fetch from memory PTI	Main memory reads per thousand instructions.	-0.62	-0.23
Global_MemUtil	Aggregated CPU-GPU memory bandwidth consumed during theoretical peak bandwidth.	-0.63	-0.56
ClockWeightedUPC	Retired micro-operations (includes all processor activity) per cycle weighted by each core's active clocks.	-0.83	0.70

6.3 Run-time System for Energy Management: Metrics, Models and Management Algorithm

The first step is to develop a predictor for the frequency sensitivity of an application. Specifically, at any point in time we need to be able to predict the performance sensitivities of the execution of a kernel on the GPU to the frequency of the CPU and the frequency of the GPU – the sensitivities may be different. As we observed in Section 6.2,

this sensitivity analysis must account for indirect interactions between the CPU cores and the GPU, e.g., in the shared memory system.

The second step is to encapsulate this into an energy-management algorithm that periodically computes the frequency sensitivity and, in response, adjusts the DVFS states of the CPU cores and the GPU. In this section, we derive a frequency-sensitivity predictor and use it to construct a run-time energy-management scheme. Our goal is to develop a simple and practical predictor that can be implemented efficiently in a dynamic run-time algorithm with minimal hardware overhead and complexity.

6.3.1 Frequency Sensitivity Correlation and Run-time Metrics

We develop frequency-sensitivity predictors to capture the dominant behaviors described in Section 6.2 for the GPU and the CPU. First, we selected performance counters that are indicators of frequency sensitivity. Modern processors provide hundreds of exposed performance counters, which makes the selection quite challenging [11]. We used three Exascale proxy applications (*miniMD, miniFE*, and *Lulesh*), each consisting of many different kernels [15][46][59]. We also utilized six scientific applications from the Rodinia benchmark suite: *Needleman-Wunsch, HotSpot, LU Decomposition (LUD), Speckle-reducing Anisotropic Diffusion (SRAD), Computational Fluid Dynamics (CFD),* and *BFS* [18][19]. The chosen applications have a wide range of characteristics ranging across coarse- and fine-grained kernels, compute- and memory-boundedness, different degrees of CPU-GPU performance-coupling, and degrees of divergent control flow.

Using application analysis and a profiling tool called CodeXL, we measured the execution times and the corresponding values of a set of performance counters/metrics at kernel boundaries over a range of CPU and GPU frequencies [127]. We initially attempted to find correlation across multiple sample points in a single application trace but found that minor discrepancies in phase alignment with performance metric traces can cause large variations in correlation. Hence, we looked for alignment only at the kernel granularity in an application. We performed a correlation analysis between each performance

counter/metric and the CPU or GPU frequency sensitivity, measured as the ratio of the difference in execution times to the corresponding differences in frequency. We computed the correlation coefficients using linear regression (shown in Table 3). These performance counters/metrics were derived from a set of more than 40 hardware performance counters in the CPU, GPU, and Northbridge selected based on the insights gained from Section 6.2. Coefficient values greater than 0.5 or less than -0.5 are considered a strong positive or negative correlation, respectively [11]. These values are highlighted in Table 3.

Second, we calculated overall GPU or CPU frequency sensitivity based on the following analysis. As expected, ClockWeightedUPC (retired micro-ops per active core clock cycle) shows high correlation for CPU frequency sensitivity, as does GPU ALU activity and ALUBusy for the GPU. This captures the compute behavior of an application in either type of core. However, to capture the compute behavior for normal operations as well as control-divergent applications, we weighted the ALUBusy metric with GPUClockBusy (as defined in Table 3, note the improvement in correlation between line 3 and line 1 in Table 3).

As Figure 41 shows, graph algorithms have a high degree of control-flow divergence; thus, some SIMD engines are idle and waiting for a thread to finish executing before all threads can re-converge and proceed. This produces poor ALU throughput, making it appear that the GPU is lightly utilized. However, when ALUBusy is weighted with the actual GPU clock activity, we get a higher rate of ALU activity for the active period and better correlation. Similar accounting has been done for CPUs [11]; however, unlike the CPU, which is latency sensitive, the GPU's massively parallel bulk-synchronous computation creates a complex inter-relationship between control behavior and power.

GPU frequency sensitivity shows a strong negative correlation to CPU UPC (retired micro-ops per cycle). This includes all processor activity (instructions, exceptions, interrupts, microcode assists, etc.) Similarly, CPU frequency sensitivity shows a strong negative correlation to GPU ALUBusy. This is because of the data and execution

dependencies between the GPU and CPU. As the computation becomes more balanced and distributed between the CPU and GPU, we expect the correlation coefficients to change. However, CPU and GPU performance still will be closely coupled in their interactions and dependencies. Therefore, a GPU frequency-sensitivity predictor needs to account for CPU UPC as a way to measure GPU's performance-coupling to CPU. Similarly, CPU frequency sensitivity in a heterogeneous architecture needs to account for GPU ALU activity.

We found a better correlation between frequency sensitivity and aggregated memory bandwidth (Global_MemUtil) compared to the localized memory access metrics such as L2 cache misses in the CPU or memory fetch/write stalls in the GPU. This is largely because of the disparity in memory-bandwidth demand between the CPU and the GPU while accessing a shared resource, as shown in Figure 40.

Based on the preceding analysis, we summarized a key set of performance metrics below to use in our run-time energy-management scheme to determine frequency sensitivities in a performance-coupled heterogeneous architecture. We determined CPU and GPU frequency sensitivities as weighted linear regression functions of these combined metrics to capture performance-coupling, core compute behavior, and global memory interference. The correlation coefficient using this combination of metrics improved to 0.97.

$$WeightedALUBusy = \frac{ALUBusy}{GPUClockBusy}$$

$$Global\_MemUtil = \frac{AggregatedMemBW}{TheoreticalPeakMemBW}$$

where

$$TheoreticalPeakMemBW = (DDRClockSpeed) * (8\ bytes\ per\ clock) * (Total\ DDR\ channels)$$

$$ClockWeightedUPC = \frac{\sum(Total\ no.of\ retired\ uops[i] * UnhaltedCoreClocks[i])}{\sum UnhaltedCoreClocks[i]}$$

Although the set of applications analyzed here uses an offload model for computation, in which kernels run on the GPU with periodic synchronization points between CPU and GPU, similar performance metrics (WeightedALUBusy, Global_MemUtil, and ClockWeightedUPC) may be utilized in the case of more concurrent computation across CPU-GPU with higher degree of performance coupling; however, the weights associated with the metrics in the linear regression equation may change to reflect the tighter performance-coupling between the CPU and GPU.

6.3.2 DynaCo: Coordinated Dynamic Energy Management Scheme

We propose a run-time energy-management scheme called *DynaCo* based on the online measurement of the frequency sensitivity described in Section 6.3.1. DynaCo is implemented as a system software policy layered on top of the baseline AMD A-Series power-management system (BAPM).

The energy-management algorithm is partitioned into a monitoring block that samples the performance counters every 10 ms to coincide with the operating system timer tick for minimizing overheads, and a decision block that computes frequency sensitivities using measurements described at the end of Section 6.3.1. The CPU and GPU DVFS states are then configured. In general, DynaCo periodically determines whether the CPU and the GPU frequencies are high or low. In each case, the energy management algorithm embodies the following logic:

1) **High GPU sensitivity, Low CPU sensitivity**: Shift power to the GPU (i.e., boost the GPU to maximize performance).
2) **High GPU sensitivity, High CPU sensitivity**: Distribute power proportionally based on their relative sensitivities.
3) **Low GPU sensitivity, High CPU sensitivity**: Shift power to the CPU (i.e., boost the CPU to maximize performance).

4) **Low GPU sensitivity, low CPU sensitivity**: Reduce power of both the CPU and the GPU by using lower-power states.

Algorithm 1: Dynamic scheme (DynaCo1-levelTh)

```
1: while TRUE do
2:   if (Global_MemUtil >= DDR_bus_efficiency) then
3:              /* Case: Memory is bottleneck */
4:     SetGPUFreqState(GPU-med);
5:     SetCPUFreqState(CPU-low-power_P2);
6:   end if
7:   else        /* Case: Memory is not bottleneck */
8:     if(ClockWeightedUPC >= UPC_Threshold) then
9: /* CPU frequency sensitive, consider GPU sensitivity */
10:       if (WeightedALUBusy>= HIGH) then
11:                   SetGPUFreqState(GPU-high);
12:           SetCPUFreqState(CPU-base);
13:       else
14:       if (MEDIUM<= WeightedALUBusy<HIGH) then
15:               SetGPUFreqState(GPU-med);
16:           SetCPUFreqState(CPU-boost);
17:       else
18:                   SetGPUFreqState(GPU-low);
19:           SetCPUFreqState(CPU-boost);
20:       end if
21:     else
22:     if(ClockWeightedUPC < IPC_Threshold) then
23:/* CPU frequency insensitive, consider GPU sensitivity */
24:          SetCPUFreqState(CPU-low-power_P2);
25:       if (WeightedALUBusy>= HIGH) then
26:            SetGPUFreqState(GPU-high);
27:       else
28:       if (MEDIUM<= WeightedALUBusy<HIGH) then
20:              SetGPUFreqState(GPU-med);
30:       else
31:                   SetGPUFreqState(GPU-low);
32:       end if
33:     end if
34:   end if
35:    Sleep.time(SAMPLING_INTERVAL);
36: end while
```

Figure 44: DynaCo-1levelTh pseudo-code.

Algorithm 2: Dynamic scheme (DynaCo-multilevelTh)

```
1: while TRUE do
----lines 2 through 21 in Algorithm 1----------
22:    if(ClockWeightedUPC <  UPC_Threshold) then
23: /* CPU frequency insensitive, consider GPU sensitivity */
24:      if (WeightedALUBusy>= HIGH) then
25:                    SetGPUFreqState(GPU-high);
26:      else
27:       if (MEDIUM<= WeightedALUBusy<HIGH) then
28:              SetGPUFreqState(GPU-med);
29:      else
30:                    SetGPUFreqState(GPU-low);
31:      end if
32:      SetCPUFreqState(CPU-low-power_Pstate);
33:      Compute_ MemAccessRate_gradient();
34:      if (gradient>=Mem_threshold) then
35:       if(CPU-low-power_Pstate<=Pmin) then
36:            CPU-low-power _Pstate++;
37:       end if
38:       else
39:       if (CPU-low-power >CPU-base+1) then
40:            CPU-low-power _Pstate--;
41:       end if
42:      end if
43:    end if
44:  end if
45:  Sleep.time(SAMPLING_INTERVAL);
46: end while
```

Figure 45: DynaCo-multilevelTh pseudo-code.

Because HPC applications are mostly uncompromising with respect to performance loss, we propose two energy-management algorithms – one more aggressive than the other in attempting to reduce power but with potentially higher performance degradation. In the less aggressive variant, *DynaCo-1levelTh* (Figure 44), we limit the lowest-frequency P-state to P2; the CPU is not permitted to go to a lower-frequency state. Thus, in this case, there is potential to lose some power-saving opportunity. In the more aggressive version,

*DynaCo-multilevelTh (*Figure 45), the CPU is allowed to use all of the low-power P-states during low-sensitivity phases by analyzing gradients in memory access rates. In both versions, the GPU is handled similarly and allowed to use all DVFS states. In Figure 45, we show DynaCo-multilevelTh for only the portions in which it is different from DynaCo-11evelTh. For our analysis, the GPU-high and -med thresholds for GPU WeightedALUBusy were set to 80% and 30%, respectively, based on GPU utilization and variations in workload intensity of graphics and HPC benchmarks. UPC_threshold was set to 0.4 based on empirical observations across a wide range of workload characteristics in this architecture. The CPU and GPU DVFS settings are described in Section 6.4. Pmin is the lowest available CPU P-state.

The key observation is that when there is significant coupling/interaction between the CPU and the GPU, having the lowest CPU P-states can lead to significant power savings but significant performance degradation. At lower levels of coupling, significant power savings can occur with little performance degradation. The choice of algorithm depends on the degree of coupling, which can be time-varying, and the degree to which performance degradation is acceptable. For example, if a HPC application has little communication overhead between the CPU and GPU, such as a compute-offload application in which the serial fraction of the code is insignificant compared to the total execution time, both DynaCo schemes may provide similar performance but DynaCo-multilevelTh will provide better power and energy savings.

6.3.3 Summary

The preceding analysis shows that HPC applications exhibit varying degrees of CPU and GPU frequency sensitivity for a variety of subtle and non-obvious reasons. Overall, the results in this section clearly point towards the need for a set of metrics for energy management that can predict CPU-GPU frequency sensitivity in a tightly coupled heterogeneous architecture. Using these metrics, we envision extending BAPM with

frequency-sensitivity information to augment its functionality. We describe the model, its application, and results with measurements on real hardware in the following sections.

6.4 Experimental Set-up

We used the AMD A10-5800 desktop APU with 100W TDP as the baseline for all our experiments and analysis. Base CPU frequency is 3.8GHz, with boost frequency up to 4.2GHz. The GPU frequency is 800MHz for the highest DVFS boost state [125]. Table 4 and Table 5 show the DVFS state table for the CPU and the GPU. We used four 2-GB DDR3-1600 DIMMs with two DIMMs per channel. Hardware performance counters for the CPU and GPU were monitored using CPU and GPU performance counter libraries running in Red Hat Linux OS. We set specific CPU DVFS states using model-specific registers as described in [12]; to set a specific GPU DVFS state, we send memory-mapped messages through the GPU driver layer to the power-management firmware.

Table 4: CPU DVFS states for AMD A10-5800 APU.

	P-state	Volt. (V)	Freq (MHz)
HW-only	Pb0	1.475	4200
	Pb1	1.45	4000
SW-visible	P0	1.363	3800
	P1	1.288	3400
	P2	1.2	2900
	P3	1.075	2400
	P4	0.963	1900
	P5	0.925	1400

Table 5: GPU DVFS states for AMD A10-5800 APU.

P-state	Volt. (V)	Freq (MHz)
DVFS-high	1.275	800
DVFS-med	1.2	633
DVFS-low	0.9375	304

Although our DynaCo scheme can be implemented in any layer such as hardware, power-management firmware, or system software, we implemented it as a run-time system software policy by layering it on top of the baseline AMD A-Series power-management system. For CPU and GPU power and temperature, we used the digital estimates provided by the power-management firmware running in the AMD A-Series processor described in Section 3.4.1, the accuracies of which are detailed in [83]. For all schemes, we ran the applications for several iterations to reach a thermally stable steady state. We took an average across those multiple iterations to eliminate run-to-run variance in our hardware measurements.

Table 6: Application datasets used for DynaCo evaluation.

Application	**Problem Size**
miniMD	32 x 32 x 32 elements
miniFE	100 x 100 x 100 elements
Lulesh	100 x 100 x 100 elements
Sort	2,097,152 elements
Stencil2D	4,096 x 4,096 elements
S3D	SHOC default for integrated GPU
BFS	1,000,000 nodes

We selected the applications used in our experiments based on their relevance to future high-performance scientific computing. We evaluated seven OpenCL applications in this paper: *miniMD, miniFE, Lulesh, S3D, Sort, Stencil2D,* and *BFS*. *MiniMD, miniFE,*

and *Lulesh* are proxy applications representative of HPC scientific application characteristics in the Exascale time-frame. A sub-set of benchmarks (*S3D, Sort, Stencil2D, BFS*) are from the Scalable Heterogeneous Computing (SHOC) benchmark suite [30] that represents a large portion of scientific code found in HPC applications. We analyzed all applications on a single node to explore energy-saving opportunities using our run-time schemes. These applications and the associated datasets are described in Table 6.

MiniMD is a molecular dynamics code derived from its parent code, LAMMPS [15]. It has two main computational kernels. The first is the L – J potential function, or *force* kernel, and the second is the neighbor-binning algorithm, or *neighbor* kernel. Other kernels include communication kernel *atom_comm* and miscellaneous small kernels to integrate the atom forces and build the neighbor's list for each atom based on proximity and other variables.

MiniFE provides an implementation of a finite-element method [46]. It provides a conjugate gradient (CG) linear system solver with Jacobi preconditioning. The three main kernels in the CG solver are *matvec*, which performs matrix vector operations; *dot*, which performs the dot product of two matrices; and *waxpy*, which computes the weighted sum of two vectors.

Lulesh [59] approaches the hydrodynamics problem using Lagrangian numerical methods. The two main computation kernels in *Lulesh* are *CalcHourGlassForces* and *CalcFBHourGlassForces*.

SHOC consists of a collection of complex scientific applications and common kernels encapsulated into benchmarks that represent a majority of the numerical operations found in HPC. We use *Sort*; which sorts an array of key-value pairs using a radix sort algorithm; *Stencil2D*, which uses a nine-point stencil operation applied to a 2D dataset; *S3D*, which is a turbulent combustion simulation; and *BFS*, which is a graph traversal algorithm.

We report performance, power, and energy efficiency (energy-delay^2 product) for the two variants of DynaCo algorithm. We picked ED^2 because it has been widely used in HPC analysis [67] [111] and it captures the importance of both power and performance, the latter being critical for HPC. The power and energy results include CPU, GPU, memory controller power, and a fixed IO-phy power budget. All results were obtained from real hardware and are normalized to the baseline BAPM discussed in Section 3.4.1. All averages represent geometric means across the applications.

6.5 Results

This section describes the results from the two DynaCo schemes for the AMD A-Series APU and compares them with the state-of-the-practice power-management algorithm BAPM. We also compare our DynaCo schemes with an ideal static scheme that picks the best DVFS state for each kernel as determined through offline profiling and analysis by performing an exhaustive state-space search. Offline techniques provide a good basis for comparison to evaluate the effectiveness of run-time techniques but are impractical as power-management strategies.

6.5.1 Performance, Power, and Energy

Figure 46 shows the performance impact of DynaCo-1levelTh, DynaCo-multilevelTh, and ideal static schemes compared to the baseline for all seven HPC applications. The y-axis represents the increase in run-time compared to a baseline value of 1.0, (lower is better). We see an average run-time increase of 0.78% across all the applications using DynaCo-1levelTh, with up to 2.58% maximum slow-down in the case of miniMD.

DynaCo-multilevelTh sees an average run-time increase of 1.61% across the same set of applications, with a worst-case slow-down of 4.19%. The ideal static scheme measures an average slow-down of 1.65%, with the worst case being 5.2% in *miniMD*. This illustrates the efficacy of the run-time schemes in optimizing energy efficiency under

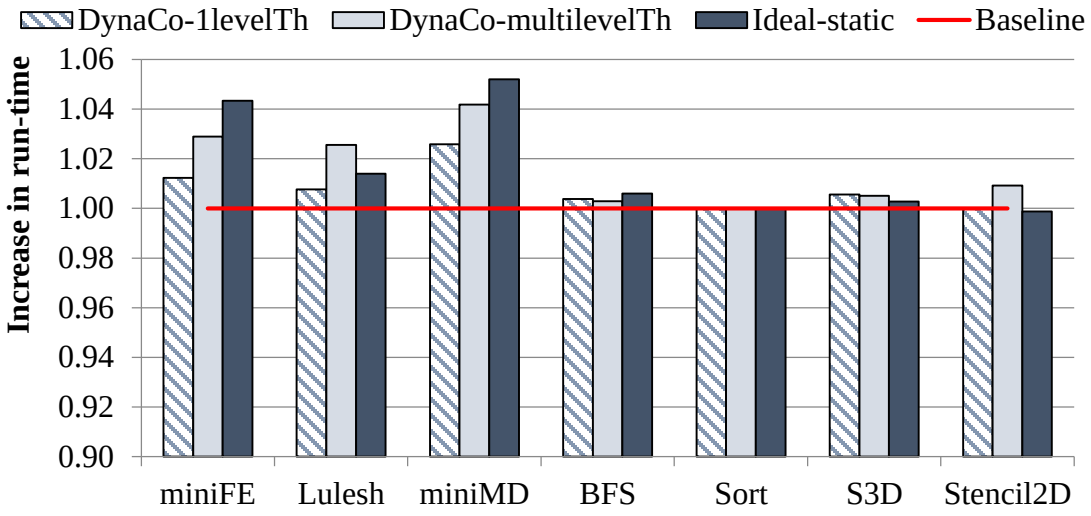


Figure 46: Performance impact of DynaCo.

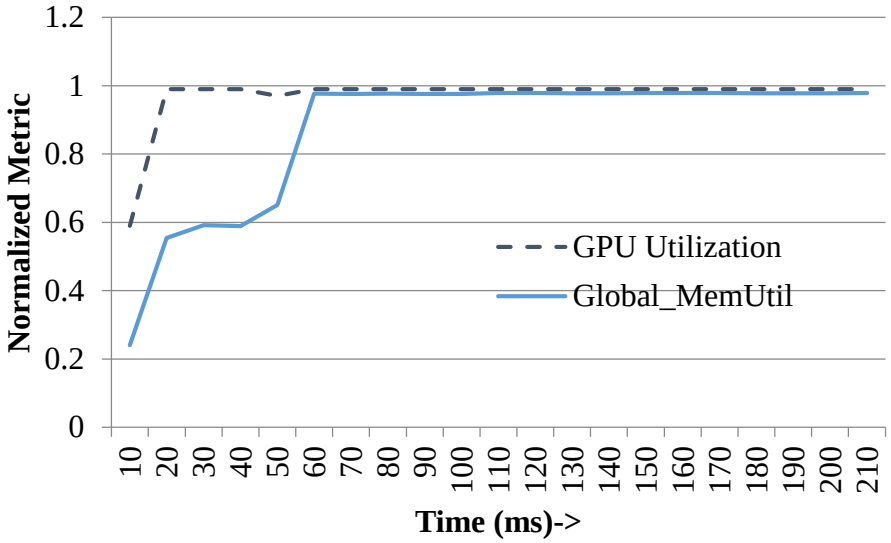


Figure 47: Phase variation within MATVEC.

strict performance constraints. Ideal-static picks the best CPU and GPU DVFS states at a kernel-level granularity, and it is unable to detect fine-grained phase changes in a kernel. Hence, it penalizes short, high-frequency sensitive phases in a kernel that overall have low sensitivity.

As expected, we see much tighter performance control with DynaCo-1levelTh compared to DynaCo-multilevelTh and ideal-static because it does not utilize the lowest-frequency states of the CPU. Since it always fixes the low-power P-state for the CPU to P2 during phases of low CPU frequency sensitivity, it also removes the slight variability in performance over time when the algorithm is adapting dynamically to find the best low-power P-state. On the other hand, DynaCo-multilevelTh provides better energy efficiency gains, as we will see next, with slightly more performance degradation but still within reasonable bounds for most HPC applications. We attribute the relatively higher performance loss in *miniMD* to the impact of variability in kernel phases shorter than our 10-ms sampling interval limitation.

The more aggressive DynaCo-multilevelTh outperforms ideal-static in miniFE and miniMD because a run-time adaptive scheme is able to take advantage of the phase behavior in a kernel, whereas the static scheme based on profiling makes power-state decisions only at kernel-level granularity. Figure 47 shows an example phase behavior of the *matvec* kernel in *miniFE* for a single iteration. The y-axis shows GPU utilization and normalized memory-bandwidth utilization compared to the practical peak-DDR bandwidth. *Matvec* performs a sparse matrix-vector product and, in general, is heavily memory bandwidth-limited due to the large number of indirect memory references and register spills to global memory in the code. However, about 19% of the time it is compute-intensive without saturating memory bandwidth. This behavior is observed in every invocation of *matvec* in *miniFE*, a significant fraction of the application's total run-time. During this 19% compute-intensive phase, DynaCo boosts the GPU to its highest DVFS state while the profiling-based ideal static scheme fixes the GPU frequency to GPU-med due to this kernel's overall low GPU frequency sensitivity.

In Figure 48 we evaluate the ED^2 gains using DynaCo during the entire run-time of the application. All data are normalized to a baseline of 1.0 (lower is better). Average energy efficiency improves by 24% using DynaCo-1levelTh compared to the baseline, with

up to 32% savings in *Sort* and *S3D*. DynaCo-multilevelTh sees an average improvement of 30%, with up to 47% savings in *S3D*. Ideal-static achieves an energy-efficiency gain of 35%. We observe that 70-80% of the savings came from CPU scaling and the remainder came from GPU scaling.

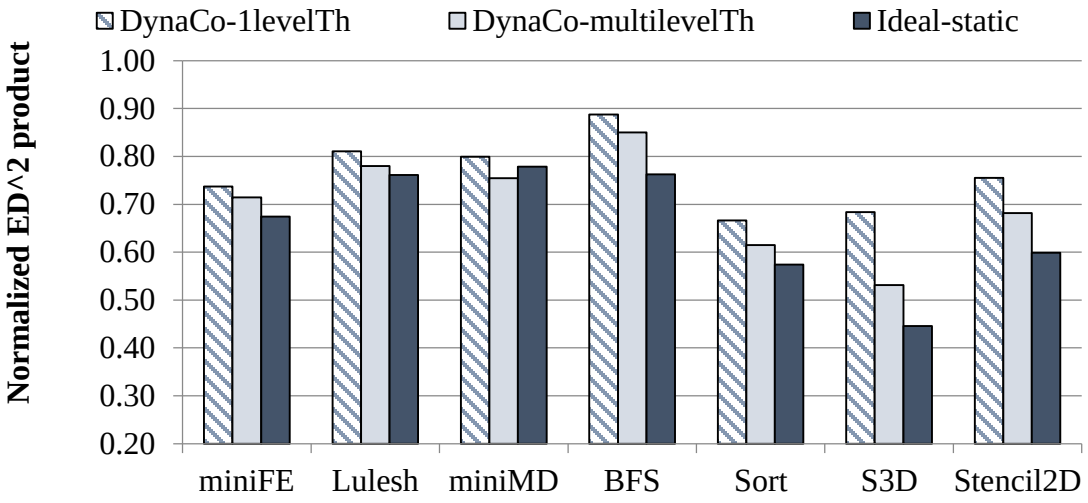


Figure 48: Energy efficiency with DynaCo.

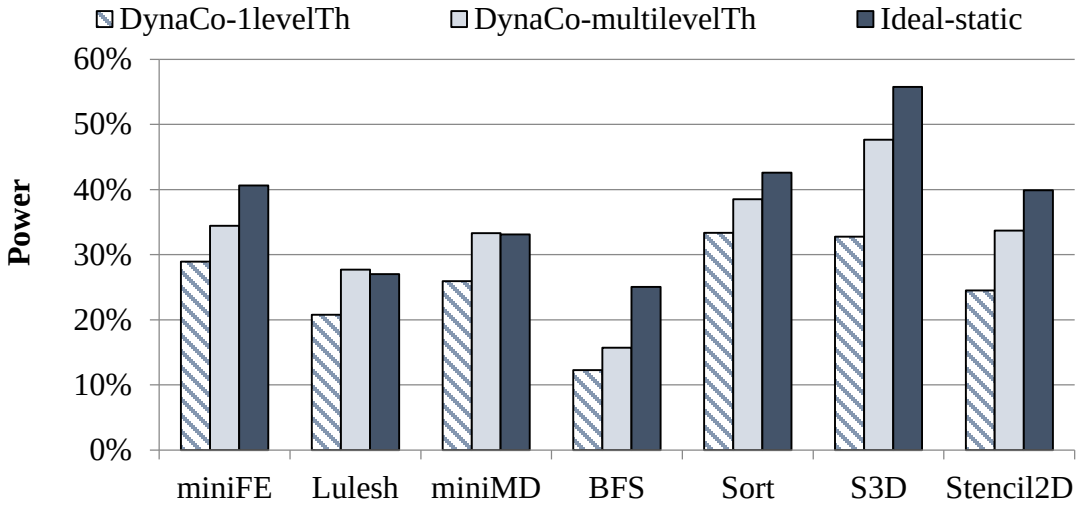


Figure 49: Power savings with DynaCo.

The amount of energy-efficiency gain in *S3D* is slightly higher than the rest of the benchmarks. *S3D* is a compute-intensive application. However, when we run multiple iterations of this benchmark from the SHOC suite, the compute-intensive active phases appear to last for a small fraction of the total time it takes to compile and launch the application kernels. This causes only small periods of activity on the GPU followed by long idle periods. During this idle period, the GPU is power-gated for all three schemes as well as the baseline. However, the CPU is busy compiling and preparing the work to launch the kernels. Portions of this phase do not contribute to the overall performance of the application. Boost algorithms, such as the BAPM algorithm used for the baseline, allocate the highest CPU frequencies during this phase when power and thermal headroom is available. However, in our run-time and ideal-static schemes we are able to utilize the low frequency P-states during the frequency-insensitive phase. We also notice that DynaCo-multilevelTh provides better energy efficiency than ideal-static for *miniMD* due to the higher slowdowns observed with the profile-based scheme.

The power savings achieved with DynaCo are illustrated in Figure 49. The average power savings are 24% with DynaCo-1levelTh, 31% with DynaCo-multilevelTh, and 36% with ideal-static. We see that DynaCo-multilevelTh provides greater power savings compared to DynaCo-1levelTh due to utilization of the very-low-frequency CPU P-states. While ideal static provides greater power savings by picking the best DVFS state for each kernel, it does not provide the same tight performance bounds as the other two schemes, as shown in Figure 46. In addition, it requires user intervention and prior offline profiling of all the kernels in an application across multiple CPU and GPU frequencies to determine the best state.

6.5.2 Performance Analysis and Power Shifting

We now analyze the case of power-shifting and power-reduction scenarios with the two DynaCo schemes for every application. We present a sub-set of those results here. Figure 50 shows the percentage GPU DVFS residency for each of the three main kernels

of *Lulesh* as well as the overall application. Since the GPU DVFS decision between the two DynaCo schemes is handled similarly from an algorithmic perspective, we show GPU DVFS residency results for only DynaCo-1levelTh.

The *CalcHourGlass* kernel spends 21% less time in GPU-high, 14% more time in GPU-med, and 8% more time in GPU-low than the baseline. On further examination, this kernel is memory-bounded 30% of its run-time; during those times, power is shifted away from the GPU. Similarly, the entire *Lulesh* application spends 9% less time in GPU-high with DynaCo. For the *CalcFBHourGlass* kernel, DynaCo performs similarly to the baseline. This kernel is heavily compute-bound on GPU with high WeightedALUBusy; hence, DynaCo boosts performance by selecting the highest-frequency state.

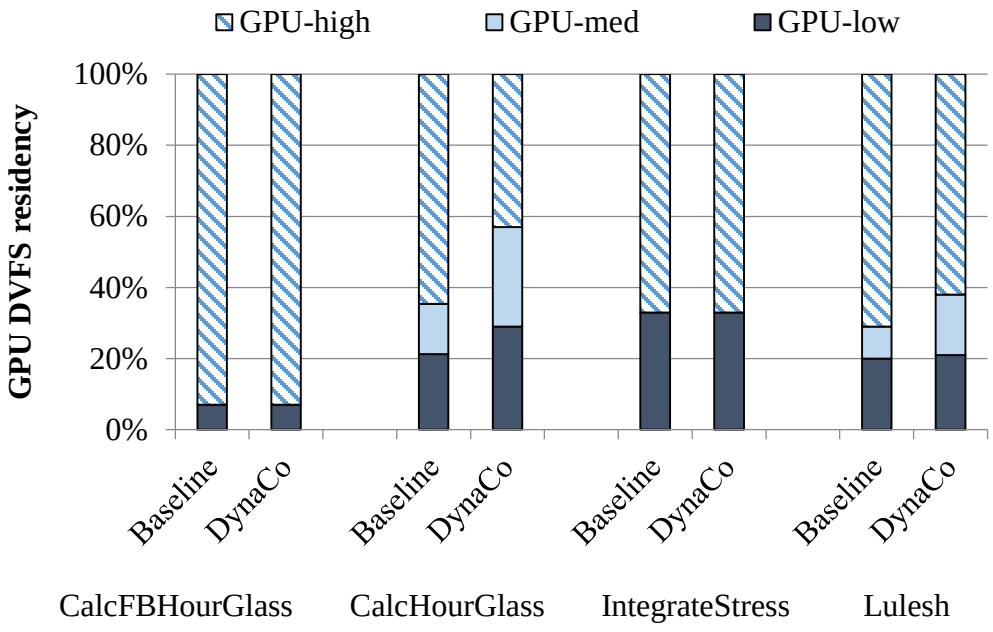


Figure 50: GPU DVFS residency for DynaCo and baseline.

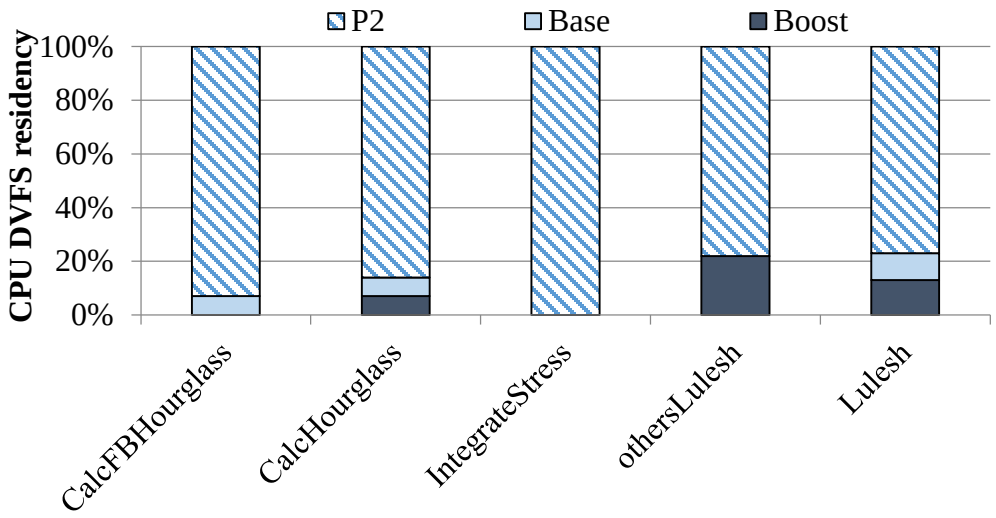


Figure 51: CPU DVFS residency with DynaCo-1levelTh.

Further, in Figure 51 and Figure 52, we see that for all three kernels we are able to shift power away from CPU significantly due to poor CPU frequency sensitivity, while the baseline runs the CPU at the high-frequency boost states due to availability of power and thermal headroom. Specifically, during the more than 20 other kernel phases, DynaCo correctly boosts CPU to the high-frequency P-states as needed due to the high CPU dependency observed for these miscellaneous fine-grained kernels, as depicted in the phase behavior shown in Figure 14. Further, DynaCo-multilevelTh is able to utilize the lower-frequency CPU P-states P3 and P4 59% of the time.

We also observe that the fine-grain, relatively small kernels such as *IntegrateStress* become performance-coupled to the CPU more quickly than the main *Hourglass* computation kernels. Hence, *IntegrateStress* does not utilize the low-frequency P-states of P3 and P4 that can cause significant performance loss. *Lulesh* provides an example of a case when power can be saved from both CPU and GPU and boosting to higher frequencies is utilized when the application phase needs it.

Similarly, for *miniMD*, DynaCo correctly estimates the frequency sensitivity of the different kernels. The heavily compute-intensive nature of the *force* kernel causes it to boost to the highest GPU frequency 100% of its run-time, similar to the baseline. On the other hand, the *neighbor* kernel has aggregated CPU-GPU memory bandwidth that is close to the peak bandwidth that the DDR bus can sustain after accounting for bus efficiency. Hence, we are able to run the GPU at the medium DVFS frequency without noticeable performance degradation. Moreover, small kernels in *miniMD* such as *atom_comm*, which rely on the CPU for data transfer and launch frequently, spend almost 70% of their time in the medium- and low-frequency GPU DVFS states using DynaCo. During much of this time, CPU is closely coupled to the GPU and runs at high-frequency P-states. Contrary to this, the baseline algorithm runs at maximum CPU and GPU frequencies for all *miniMD* kernels due to temperature headroom.

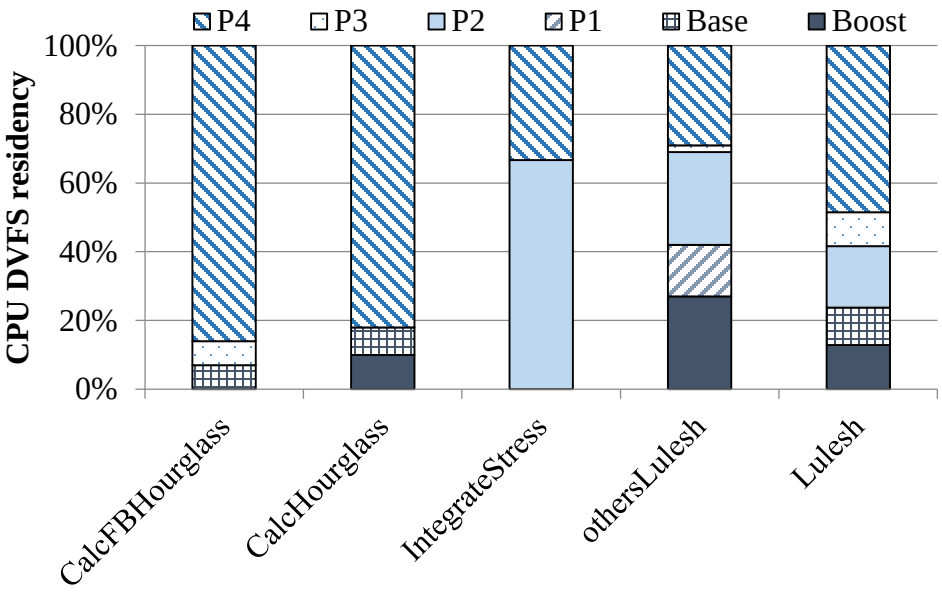


Figure 52: CPU DVFS residency with DynaCo-multilevelTh.

In the graph algorithm *BFS*, we see that due to control divergence the GPU has short bursts of computation followed by phases of low utilization on the GPU. About 25% of the time, threads are waiting for re-convergence. DynaCo correctly assigns high GPU frequency to avoid slowing the critical path, but it saves power from the CPU due to low UPC. The baseline always runs *BFS* at the maximum CPU and GPU frequencies due to the available thermal headroom, causing energy inefficiency.

Due to the lower power consumption, we also see a reduction in the peak die temperatures using DynaCo. This is due to a combination of leakage power savings from reduced voltage operations and dynamic power savings from reduced frequency. With DynaCo, peak die temperature is, on average, up to 2°C lower across all the applications. Lower temperatures result in lower cooling costs, better energy efficiency, and better heat management.

In summary, we have shown that DynaCo successfully leverages the metrics discussed in Section 6.3 to improve the energy efficiency of HPC application on a heterogeneous processor. DynaCo is able to produce significant power savings with a small reduction in performance, resulting in energy efficiencies comparable to an ideal static management scheme without the additional overhead of profiling required for the static scheme.

6.6 Conclusions

This component of the thesis proposed and implemented a set of techniques to improve the energy efficiency of integrated CPU-GPU processors. To the best of our knowledge, this is the first such implementation. We described the unique characteristics of HPC applications and the opportunities they present to save energy. We proposed a model to capture the application's frequency sensitivity in such architectures and used this model as the basis for a dynamic, coordinated energy-management scheme to improve energy efficiency at negligible performance loss. The proposed scheme achieves an

average ED^2 benefit of up to 30% compared to the baseline with less than 2% average performance loss across a variety of Exascale and other HPC applications. *This research work was published at Supercomputing (SC) 2013 [91] and Journal of Scientific Programming 2014 [92]*.

CHAPTER 7

GPU-MEMORY PERFORMANCE COUPLING MANAGEMENT

In CHAPTER 4 we described how compute and memory behavior are fundamentally performance coupled. If we ignore this coupling in managing the shared platform power envelope in future massively parallel systems with a large amount of available memory bandwidth, significant energy/power is wasted. Effective use of the power budget is of utmost importance to meet performance critical demands and improve energy efficiency. We show how the philosophy of coordinated power management continues to be a key ingredient in satisfying these demands between other pairs of coupled resources. This chapter addresses the problem of efficiently managing the relative power demands of a high performance GPU and its memory system by developing a run-time power management infrastructure called *Harmonia* that coordinates platform hardware configurations between compute and memory with time-varying application demands so that they are in balance, or in "harmony".

7.1 Overview

GPUs or graphics processing units are now commonly used for data parallel applications that do not fit into the traditional graphics space. They have been shown to provide significant improvements in power efficiency and performance efficiency for many classes of applications [6][62][128][129]. However, while compute has been a major consumer of power in such systems, moving forward we see that the power spent in the memory system and in data movement will begin to become a major, and sometimes a dominant component of platform power [64][103], as discussed in Chapter 4.2.2. This distribution of power consumption between compute and memory must operate under a fixed board level power and thermal envelope, while with the advent of on-package DRAM

e.g., die stacks and EDRAM [39][56][74][93], they must share an even tighter package power and thermal envelope. Therefore we argue that effective dynamic power redistribution between compute and memory will be key to energy and power efficiency for future high performance GPUs.

As discussed in Chapter 4.2.2, hardware tunables such as the number of parallel cores, core operating frequency, and the memory bandwidth collectively capture the relative time and power cost of performing operations vs. memory accesses in the hardware platform. Ideally, the relative ops/byte demand of the applications matches the relative time and power costs of compute and memory hardware of the platform and we have a perfectly balanced system [22][116], without wasted power and/or unexploited performance opportunities. In reality, application behavior is time-varying, and the ops/byte costs of the platform depend on the values of the hardware tunables. Hence to retain the most power efficient operation we need a runtime power management infrastructure that must identify the time-varying performance sensitivity of an application to the ops/byte cost of a platform and coordinate power states of the processor (GPU) and the off-chip memory system so that they are in balance, or in "harmony".

To this end, we propose *Harmonia*, a runtime scheme that adjusts the hardware tunables on a state-of-the-art, high performance discrete GPU so as to balance the power in the memory system and GPU cores to match the demanded ops/byte characteristics of an HPC application. We show how such a balance can reduce overall platform power with little compromise in performance. Our focus is on the high performance computing (HPC) domain where applications are characterized by relatively uncompromising demands for execution time performance thereby placing stringent demands on improvements in power and energy efficiency.

Specifically, this chapter makes the following contributions:

- Through measurements on a modern GPU, we provide an analysis of algorithmic, architectural and micro-architectural behaviors that have a significant impact on the

performance sensitivity of high performance and scientific computing applications with respect to three hardware tunables—the number of GPU compute units (CU), CU frequency, and memory bandwidth. In-depth characterization of the performance sensitivity was discussed in Chapter 4.2.2.

- Based on the analysis, we derive online models that predict performance sensitivity of application kernels to each of the three hardware tunables.
- We propose a coordinated two-level power management scheme, *Harmonia*, to tune platform balance between compute throughput and memory bandwidth by i) a coarse-grain adjustment of the GPU and the memory power states based on online sensitivity prediction, ii) followed by fine-grain tuning through closed-loop performance feedback.
- Using measurements from an implementation on commodity hardware, we compare Harmonia to a commercial, state-of-the-practice power management algorithm, demonstrating that up to 36% (average of 12%) improvements in energy-delay^2 product are feasible with minimal sacrifices in performance. In addition, we also show that Harmonia achieves to within 3% of an oracle scheme.

The rest of the chapter is organized as follows: Section 7.2 presents a detailed analysis of the behaviors that have a substantive impact on performance coupling between compute and memory. Section 7.3 presents run-time metrics and the sensitivity model to capture these behaviors. Section 7.4 details the power management algorithm, Harmonia, to balance the power allocation between performance-coupled GPU compute and memory. Section 7.5 describes the experimental setup. Finally, Section 7.6 presents a detailed implementation and evaluation of Harmonia on a modern high performance GPU.

7.2 Compute and Memory Bandwidth Sensitivity Analysis

Chapter 4.2.2.2 describes the scope of impact of hardware tunables on power and performance on a modern high performance GPU AMD HD7970. The characterization underscores the power saving opportunities associated with hardware balance points. The key now is identifying measurable quantities that reflect performance sensitivities to balance points and using these measurements to tune the hardware to the optimal balance point. For the rest of this chapter we use the experimental setup described in Chapter 7.5.

To develop an online technique to effectively set the value of these three tunables we must understand the sensitivity of performance metrics to changes in values of these tunables. The sensitivity of performance to a hardware tunable is computed as the ratio of the relative change in the performance metric to the relative change in the corresponding values of the hardware tunable. Here we present some of the application and architectural considerations that determine sensitivity of various performance metrics. We illustrate these considerations through a few representative applications and their kernels depicting a wide range of phase behavior.

7.2.1 Kernel Occupancy and Latency Hiding

In massively parallel GPU architectures, latency to access memory is largely hidden through overlapped concurrent execution of many wavefronts. Kernel occupancy is a measure of concurrent execution and the utilization of the hardware resources, e.g., LDS, SGPRs and VGPRs. The number of wavefronts that can be in flight depends on how these resources are allocated across wavefronts – any one resource can be the bottleneck. For example, the higher the scratchpad and register resource requirements per work-item, the less number of waves that can be simultaneously in flight. This results in lower memory parallelism and often less sensitivity to memory bandwidth.

Figure 53 shows memory bandwidth sensitivity of kernel occupancy measured on the HD7970 for *Sort.bottom_scan* from the SHOC benchmark suite, and

CoMD.AdvanceVelocity from the ExaScale proxy applications [15][30]. In this case *Sort.bottom_scan* has a kernel occupancy of only 30%. The limiting factor is the number of VGPRs used. The VGPRs needed per wavefront is more than 25% (66) of the total number of available VGPRs (256), hence only 3 simultaneous wavefronts per SIMD unit (instead of a maximum 10) or 12 per CU can be in-flight concurrently, leading to less sensitivity to memory bus frequency due to less degree of parallelism. On the other hand, *CoMD.AdvanceVelocity* kernel has 100% kernel occupancy because the VGPR is not a limiting resource, leading to increased memory level parallelism and sensitivity to memory bandwidth.

The lesson here is we need online measurements of an application's algorithmic properties (such as workgroup size and GPR usage) and kernel occupancy to make effective decisions regarding the impact and setting of compute and memory configurations.

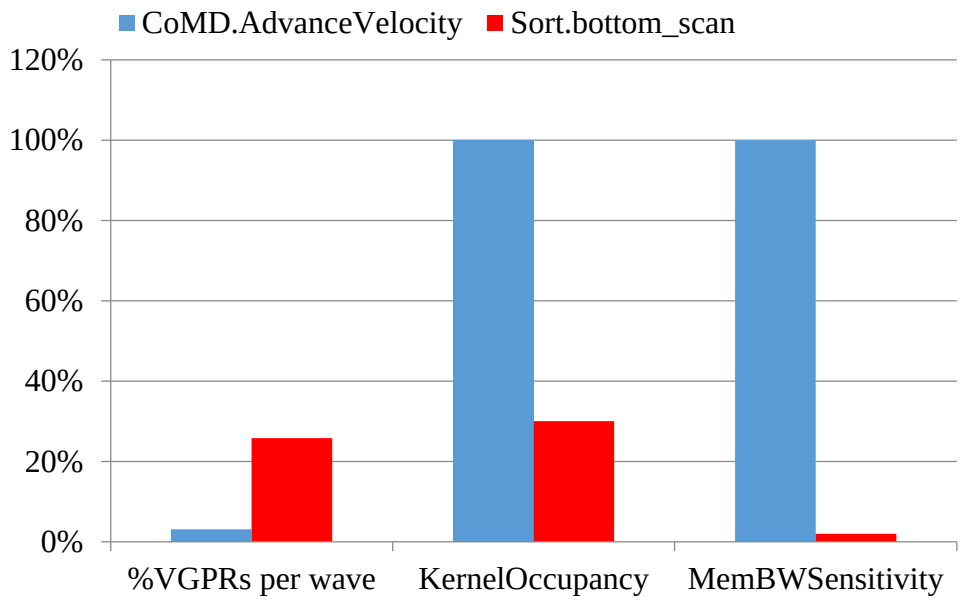


Figure 53: Effects of VGPR-caused kernel occupancy limitation.

7.2.2 Load Imbalance Due to Branch Divergence and Kernel Complexity

Control divergence causes thread serialization which can severely degrade performance. Prior works [77] have shown that performance is sensitive to compute frequency for such workloads since it speeds up serialized execution of parallel threads and thereby shortens the overall execution time. However, frequency sensitivity cannot be inferred by branch divergence measures alone. Low divergence in large kernels can have significant impact while large divergence in small kernels (i.e., executing a smaller number of dynamic instructions) may have little impact.

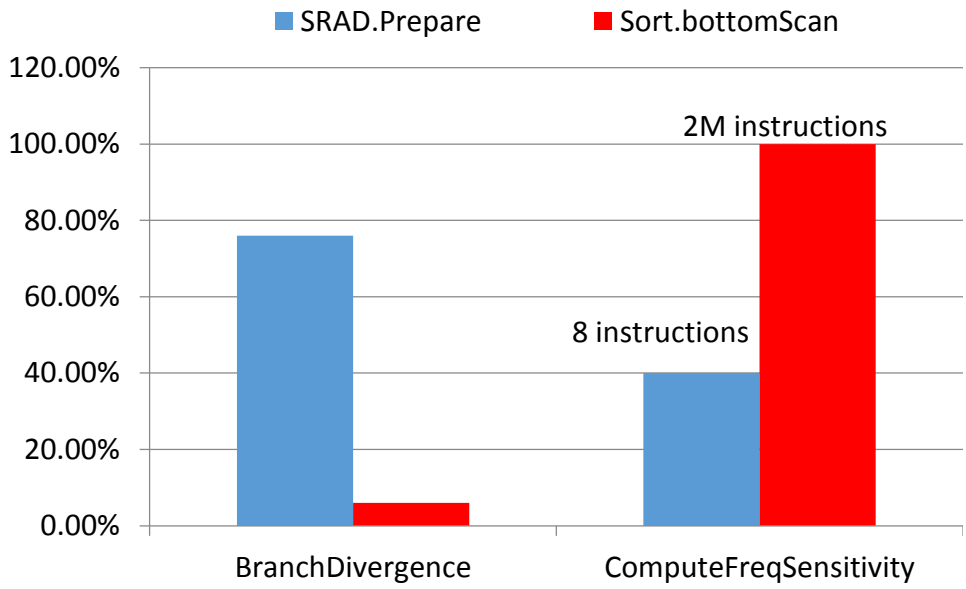


Figure 54: Impact on compute frequency sensitivity from load imbalance (branch divergence) and no. of instructions.

Figure 54 shows compute frequency sensitivity for *SRAD.Prepare* and *Sort.BottomScan*, from Rodinia and SHOC benchmark suites respectively. The first set of bars indicate branch divergence and the second set indicate measured compute frequency

sensitivity. While the *SRAD.Prepare* kernel has about 75% branch divergence it has only 8 ALU instructions, making this kernel's impact on application performance less sensitive to compute frequency and more dominated by other overheads. However, *Sort.BottomScan* has only 6% branch divergence across over 2 million instructions, leading to significant thread serialization effects and load imbalances, and thus high sensitivity to compute frequency.

The lesson here is branch divergence affects compute frequency sensitivity and its effects must be evaluated in the context of the amount of work a kernel performs, to determine its net influence on load imbalance, critical path and overall performance.

7.2.3 Architectural Clock Domains

Finally, we note that chip-scale global interactions between multiple clock domains can create non-obvious sensitivities. In our case, the GPU L2 cache (using the compute clock) and the on-chip memory controller (using the memory clock) are in different clock domains. Reducing compute frequency reduces the rate at which requests are delivered from the L2 cache to the memory controller clock domain. For extremely memory bound benchmarks with very poor L2 hit rates, slowing down compute frequency can hurt overall performance. The left column in Figure 55 shows off-chip interconnect activity (icActivity) for *DeviceMemory*. This application has an ops/byte demand of 2 with poor cache hit rate in the L2, which would otherwise make this kernel memory bound. However, the right column in Figure 55 indicates its high sensitivity to compute frequency, especially when compute frequency is low since the effective bandwidth to the DRAM is reduced.

The lesson here is that multiple clock domains introduce performance sensitivities that are non-obvious and must be amenable to measurement. In addition, off-chip data movement activities and operational intensity must be monitored.

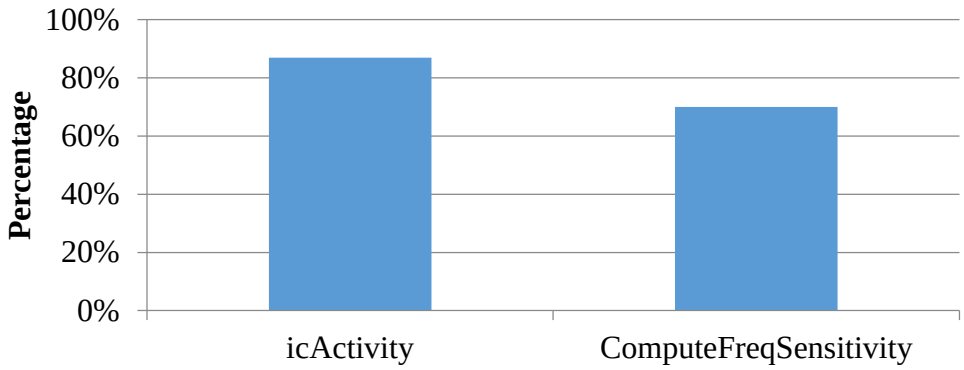


Figure 55: Impact of clock domains on compute frequency sensitivity for memory-intensive workloads.

In summary, hardware balance is sensitive to several expected as well as non-obvious behaviors in the hardware platform. Our analysis indicates that achieving hardware balance requires periodically assessing the sensitivity of performance to the hardware tunables accompanied by proportional changes to the values of the hardware tunables. The next section describes the development of sensitivity predictors for this purpose.

7.3 Compute and Memory Bandwidth Sensitivity Predictors

Our goal is the coordinated determination of the power consumption of the GPU cores and power consumption of the memory system. Power consumption of the GPU cores is controlled through the control of the number of CUs and the frequency (and hence voltage) of the CUs. The power consumption of the memory system is controlled through the memory bus frequency. We develop models to predict the sensitivity of the application to compute throughput (set by active CU count and CU frequency) and memory bandwidth (set by memory frequency) configurations. The predictors are developed based on measurement data from a wide range of simple and complex applications with one or many kernels for a total of 25 application kernels representing a variety of behaviors common in the domain of HPC and scientific computing.

7.3.1 Performance Sensitivity Measurements

We executed the kernels and applications multiple times for multiple iterations across an entire design space of 450 distinct compute and memory configuration states over 8X range of CUs, 3.33X range of CU frequency and 2.89X range of memory bandwidth. More details of the experimental methodology are explained in Chapter 4.2.2.1.1. For each hardware configuration, we measured average execution time for each kernel across all the iterations. Sensitivity is computed for each hardware configuration. CU sensitivity is computed as the ratio of i) relative change in execution times, to ii) relative change in number of active CUs. CU frequency and memory bandwidth are set to their maximum possible values in the hardware so that they are not the limiting factors. Sensitivities to CU frequency and memory bandwidth are similarly computed. Finally, the sensitivity to the number of CUs and CU frequency are aggregated into a single compute throughput sensitivity metric. The sensitivity models are then derived from these measurements as follows.

Table 7: Performance counters and metrics for Harmonia.

Metric	Description
VALUUtilization	The percentage of active vector ALU threads in a wave, indicates branch divergence
MemUnitBusy	The percentage of GPUTime the memory fetch/read unit is active, including stalls and cache effects
MemUnitStalled	The percentage of GPUTime the memory fetch/read unit is stalled
WriteUnitStalled	The % of GPUTime memory write/store unit is stalled.
NormVGPR	The number of general purpose vector registers used by the kernel, normalized by max 256
NormSGPR	The number of general purpose scalar registers used by the kernel, normalized by max 102
icActivity	Off-chip interconnect bus utilization between GPU L2 and DRAM
ComptoMemIntensity	Ratio of the time the vector ALU unit is busy processing active threads (VALUBusy*VALUUtilization) to the time the memory unit is busy (MemUnitBusy), normalized to 100

7.3.2 Performance Counter Correlation

Together with performance, we recorded an average of 50+ performance counters over all iterations of each kernel and application, resulting in one data point for every counter for each kernel at every hardware configuration. We normalized all counter values to a percentage of its maximum possible value in order to ensure proper weighted representation of all events in the training data. For a total of 25 kernels, this resulted in a total of 11250 vectors of performance counter values (25x450). We found that among multiple application kernels the performance counters vary quite a bit as expected. However, for the same kernel across multiple hardware configurations there are generally only small variations around the nominal values. Therefore, each performance counter value for a kernel is replaced by its average value across all hardware configurations. This enabled us to reduce the total training data set to 2000 points across all kernels. Each such vector is associated with its corresponding compute throughput sensitivity and memory bandwidth sensitivity.

7.3.3 Sensitivity Predictor Creation

Across the 2000 points we performed a correlation analysis between measured sensitivities and performance counters across all kernels using linear regression. Coefficient values greater than 0.5 or less than -0.5 are considered a strong positive or negative correlation, respectively [11]. From correlation analysis we selected a few counters that capture behaviors identified in Section 7.2 as shown in Table 7. These are used to construct a linear regression model for compute throughput and memory bandwidth sensitivity. The correlation coefficient using this combination of metrics was 0.91 for compute throughput sensitivity and 0.96 for bandwidth sensitivity respectively. Accuracies of these predictors are discussed in Section 7.6. **Table 8** represents the coefficients of the linear regression models. Two metrics are not directly available in hardware performance counters, and are calculated as follows.

$$icActivitiy = \frac{ReadWriteMemBW}{PeakMemBW}$$

Where,

$$PeakMemBW = MemFrequency * BusWidth * numMemChannels * GDDR5TransferRate$$

To determine compute to memory intensity of an application online, we use the following metric:

$$ComptoMemIntensity = \frac{\%time\ GPU\ is\ busy\ processing\ active\ ALU\ operations}{\%time\ GPU\ is\ busy\ processing\ memory\ operations} = \frac{(VALUBusy * VALUUtilization)/100}{MemUnitBusy}$$

Table 8: Harmonia sensitivity model and parameters.

Bandwidth Sensitivity		**Compute Sensitivity**	
Counter or Metric	**Coefficient**	**Counter or Metric**	**Coefficient**
Intercept	-0.42	Intercept	0.06
VALUUtilization	0.003	ComptoMemIntensity	0.007
WriteUnitStalled	0.011	NormVGPR	0.452
MemUnitBusy	0.01	NormSGPR	0.024
MemUnitStalled	-0.004		
icActivity	1.003		
NormVGPR	1.158		
NormSGPR	-0.731		

7.4 Harmonia: Coordinated Two-Level Power Management

Based on the preceding analysis, we find that an effective approach to achieving hardware balance involves two steps: i) employing sensitivities to the hardware tunables

Harmonia Algorithm

```
1: while TRUE do
2:   /*Monitoring Loop*/
3:   /* Online Sensitivity Computation Loop */
4:       ComputeThroughputSensitivity = model1
5:       BandwidthSensitivity = model2
6:       Bin sensitivities to HIGH, MED, LOW
7:   /*Coarse-Grain tuning (CG Block)*/
8:   if(sensitivity) changed
9:        if(CU or comp_freq or mem_freq) changed in
previous iteration
10:            Revert_prev_decision() /*sensitivities artificially
changed due to configuration change*/
11:         else /*Application phase change
12:            SetCU_Freq _MemBW(sensitivity_bin);
13:   else /*case of same sensitivities*/
14:    /*Fine-Grain tuning (FG Block)*/
15:       if(VALUBusy gradient)>=0
16:           Decrement state; //CU, CU_freq,or BW
17:       elseif(VALUBusy gradient)<0
18:           Increment state
19:           CountDithering()
20:         if(dithering>max) converge to last state with zero
gradient
21:       end if
22:    endif
23:  Run at config identified
23:     Sleep.time(SAMPLING_INTERVAL);
24: end while
```

Figure 56: Harmonia algorithm overview.

to make larger adjustments to the hardware configurations, and ii) fine tuning the configurations based on performance feedback to further improve hardware balance. We refer to the former as coarse-grain (CG) tuning and the latter as fine-grain (FG) tuning. As the number of hardware power configurations grows in future processors we expect such

coarse-fine schemes will be increasingly effective. Figure 56 specifies the Harmonia algorithm.

7.4.1 Harmonia: Structure

Harmonia operates as a system software policy overlaid on top of the baseline HD7970 power management system. As described in Chapter 3.4.2, the baseline policy manages power to just the power states mentioned in Section 7.5. Our implementation is organized into i) a monitoring block that samples the performance counters at application kernel boundaries, ii) a coarse-grain decision block CG that calculates memory bandwidth and compute throughput sensitivities based on **Table 8** and brings the hardware configuration to the "vicinty" of the balance point, and iii) a fine-grain tuning block FG that fine-tunes configurations to further improve balance, based on real time performance feedback. Although the monitoring and decision blocks of Harmonia can operate at periodic smaller intervals, due to performance counter limitations in the current device, we monitor and calculate sensitivities at kernel boundaries and use each kernel's historical data from previous iterations to predict hardware configurations for the same kernel in the next iteration. For applications that use iterative convergence algorithms and invoke multiple kernels multiple times, Harmonia records the last best hardware configuration for all kernels within that application. This state is the initial state for the subsequent iteration. Such iterative behaviors are quite common in HPC and scientific applications.

7.4.2 Harmonia: Algorithm

In this section, we describe the details of the Harmonia algorithm and its different components.

7.4.2.1 Sensitivity based Coarse-Grain (CG) Adjustments

Within the CG block, Harmonia implements a sensitivity binning policy to bin measured compute and bandwidth sensitivities into three bins of high, medium and low. In our case the thresholds for the three bins are set to 70% and 30% of the maximum, driven

by empirical measurements of ops/byte across all benchmarks. The change in actual values of the hardware tunables is proportional to the sensitivity value. Periodic enforcement of hardware configurations can artificially change sensitivities and dampen natural workload behavior. To prevent this and isolate sensitivity changes due to workload from those due to changes in the hardware tunables, we only execute CG when there has been no changes in the hardware tunables prior to the sensitivity change.

7.4.2.2 Performance Feedback Driven Fine-Grain (FG) Tuning

Predictive techniques using sensitivity models provide guidance for making "near-optimal" adjustments to the power states of the GPU. However, during run-time power management, closed-loop performance feedback is essential for making fine grained adjustments for each application to achieve the most effective hardware balance. This is especially important when there are mispredictions in the sensitivities or there are longer term changes in learned behaviors that are used to train the sensitivity predictors. Harmonia's FG block fine-tunes each of the hardware tunables based on performance feedback using the gradient of core utilization. The idea is to reduce power when the gradient is positive or zero and increase power when the gradient is negative so as to eventually settle at the balance point (minimum configuration with 0 gradient). To prevent oscillation, the configuration is set to the last best state after a certain number of oscillations to enable convergence prior to the next workload phase. In our implementation, utilization is measured with the Vector ALU Busy (VALUBusy) performance counter—this counter represents the percentage time the GPU is processing ALU instructions and is strongly correlated to execution time performance as shown in Figure 57. Here X-axis indicates %VALUBusy and Y-axis indicates performance normalized to the minimum hardware configuration.

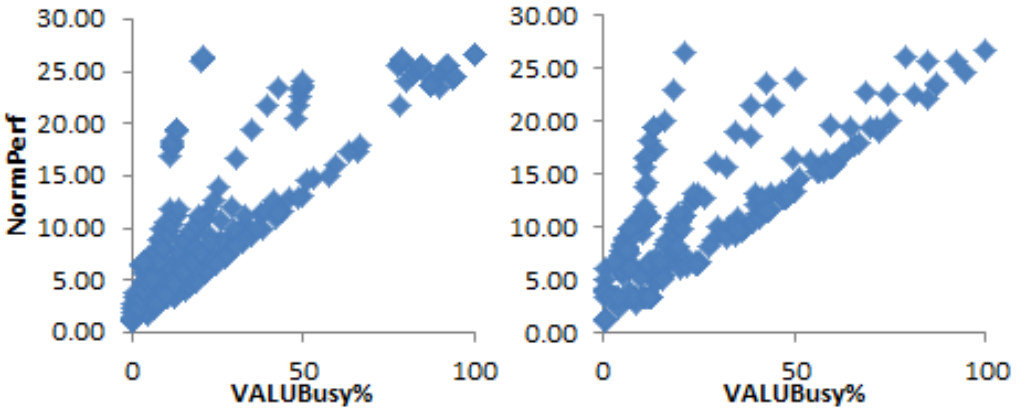


Figure 57: Performance sensitivity of Vector ALU Busy (VALUBusy) to number of active CUs (left) and memory bandwidth (right).

7.5 Experimental Setup

We use an AMD Radeon HD7970 discrete graphics card with 32 compute units as the baseline for all our experiments and analysis. The possible hardware configurations in the default HD7970 are provided in Table 9. In our analysis there are 450 possible combinations of the number of active CUs, compute frequency, and memory bus frequency as described in Chapter 4.2.2.1.1. When varying compute frequency, voltage is also scaled as noted in Table 9. When scaling memory bus frequency, voltage was fixed at the hardware default value due to platform constraints. All inactive CUs are power gated. Hardware performance counters were monitored using the GPU performance counter library CodeXL running in Red Hat Linux OS [9]. We implemented Harmonia as a run-time system software policy by layering it on top of the baseline AMD HD7970 power management system.

We selected 14 applications with many kernels, covering a wide range of typical applications to reflect the needs of the HPC and scientific computing community. They include Exascale HPC proxy apps (*CoMD, XSBench, miniFE*) [15][46], *Graph500* [81], B+Tree (*BPT*) [28], *CFD, LUD, SRAD* and *Streamcluster* from Rodinia [18][19], and *Stencil, Sort, SPMV, MaxFlops* and *DeviceMemory* from SHOC [30]. We ran each

application multiple times and recorded the average to eliminate run-to-run variance in our hardware measurements.

Table 9: GPU DVFS states for AMD HD7970 dGPU.

GPU DVFS-state	Freq (MHz)	Voltage (V)
DPM0	300	0.85
DPM1	500	0.95
DPM2	925	1.17

We measured performance as the total execution time of the application running on the GPU. Power was profiled using a National Instruments data acquisition (DAQ) card (NI PCIe-6353), with a sampling frequency of 1KHz. Total GPU card power (GPUCardPwr) was measured at the PCI-e connector interface between the motherboard and the GPU card and it includes power of the GPU chip, its on-chip memory controller, DDR bus transceivers (PHYs), off-chip GDDR5 memory, fan, voltage regulators, and other miscellaneous components on the card. We also separately measured the GPU chip power (GPUPwr) which includes power of the GPU compute, integrated memory controller, but not the PHYs. Through detailed measurements and evaluation under idle conditions, we characterized the "rest of the card power" (OtherPwr) as power due to the fan, voltage regulators, board trace losses and other minor discrete components. To ensure a constant OtherPwr we fixed the fan speed to the highest RPM at all times, independent of the workload. Based on these measurements, we derived memory power (MemPwr) as the power consumed by off-chip memory and DDR PHYs that are integrated within the GPU chip. Due to platform measurement constraints, memory controller power is not included

in measured memory power, instead it is part of GPUPwr, but it accounts for only about 3% of the overall memory power in our case.

$$\text{MemPwr} = \text{GPUCardPwr} - \text{GPUPwr} - \text{RestOfCardPwr}$$

7.5.1 Metrics

We note that HPC applications demand minimal degradations in execution time. Consequently, our goal is to minimize energy expenditure while keeping execution time constant (at best). This can be achieved by improving energy efficiency (ops/joule). Under a fixed execution time constraint it is equivalent to improving power efficiency. To capture this relative importance of *both* time and energy we can utilize metrics of energy-delay

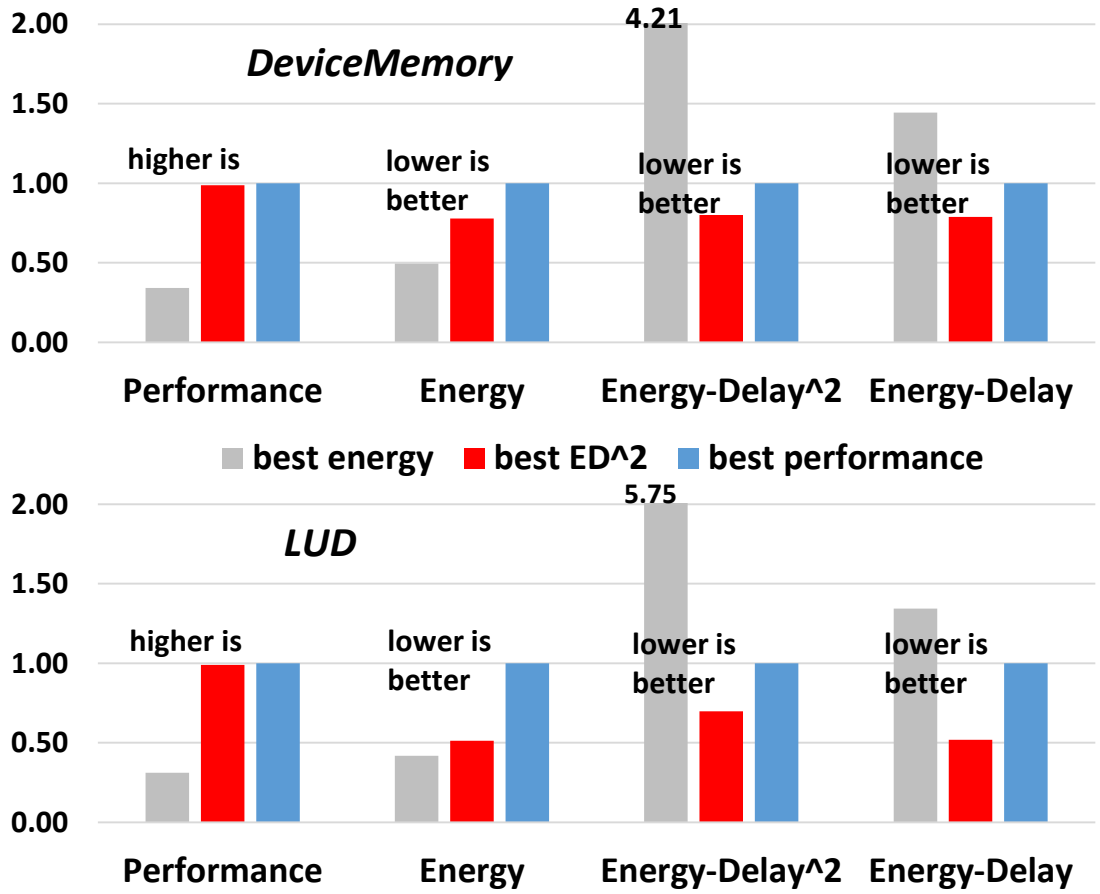


Figure 58: Performance, energy, energy-delay^2 and energy-delay comparisons for LUD and DeviceMemory. Energy efficiency leads to significant performance.

(ED) and energy-delay square (ED2). The latter in particular is commonly used in HPC application analysis [67][111].

Figure 58 shows the following analysis of the behavior of these metrics. We performed an exhaustive design space exploration across all 450 hardware configurations for *LUD* and *DeviceMemory* searching for the configurations that i) minimize energy, ii) minimize ED^2, or iii) maximize performance, as indicated by the three bars in each group of columns. For each of these three configurations we noted the corresponding measured performance, energy, ED^2 and ED. All results are normalized relative to the best performing configuration. We found that the configuration optimizing for energy (1st bar) would result in 69% and 66% performance loss for *LUD* and *DeviceMemory*, respectively, compared to the best performing configuration (3rd bar). On the other hand, the configuration optimizing for ED^2 (2nd bar) has only 1% performance penalty but still realizes 60% and 38% reduction in energy compared to the energy optimized case. Thus, we use ED^2 as the main metric for evaluation motivated by its wide usage in HPC application analysis [67][111] and note that using ED here yielded similar conclusions.

7.6 Results

All results were obtained from commodity hardware and are normalized to the baseline HD7970 power management system discussed in Section 3.4.2. All averages represent the geometric mean across the applications. Finally, we also compare Harmonia with an oracle scheme optimized for ED^2 based on exhaustive online profiling of every iteration of each kernel across all of the 450 possible hardware configurations (See Chapter 4.2.2.1.1). While the oracle technique provides a useful basis for evaluation, it is impractical as a power management strategy.

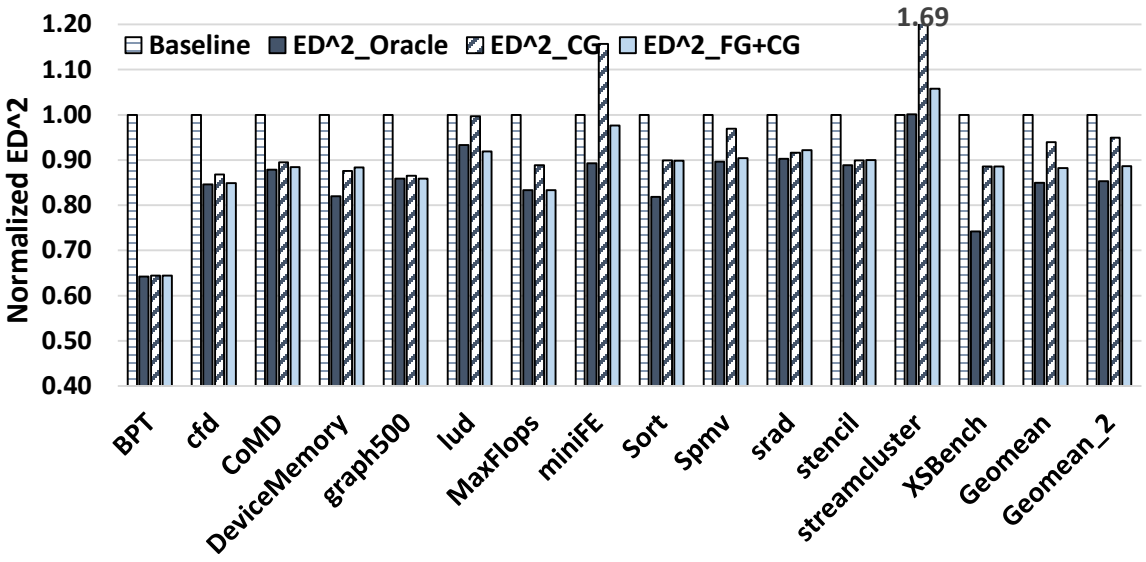


Figure 59: Overall combined performance and energy gain from Harmonia, using the ED^2 metric.

7.6.1 Performance, Power, and Energy Efficiency

Figure 59 and Figure 60 illustrate improvements in ED^2 and energy respectively relative to the baseline and the oracle. In addition, we also demonstrate the performance of just CG tuning. Harmonia is represented by the "FG+CG" bars. Due to the consistent availability of thermal headroom, the baseline power management always ran at the boost frequency of 1GHz for all applications. We show two geometric means to ensure results are not skewed by the stress benchmarks *MaxFlops* and *DeviceMemory*, which represent extreme cases of compute and memory limiting behavior respectively. Geomean_2 which is the last set of bars excludes those two stress benchmarks. Harmonia realizes an average ED^2 improvement of 12% compared to the baseline, with up to 36% savings for *BPT*. Of this 12% ED^2 savings, about 6% is due to CG tuning, with the remaining from the fine-grain tuning. In addition, Harmonia is typically within 3% of the oracle. Interestingly, we observe that the energy savings is almost identical between the CG and FG+CG schemes,

with a contribution of only 2% coming from the FG loop. However, FG tuning is important for preserving performance as described next.

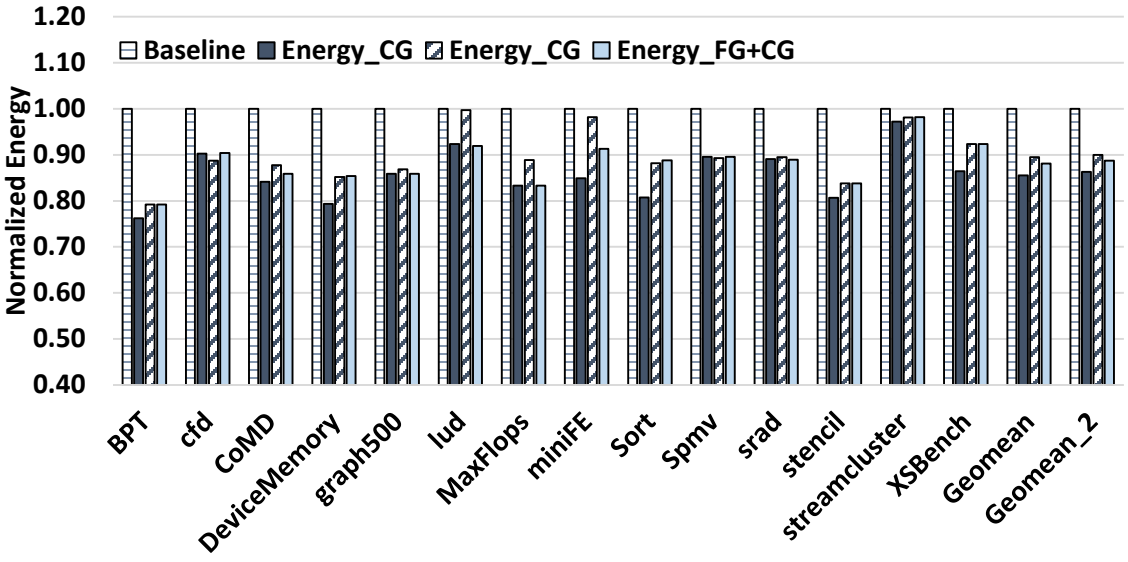


Figure 60: Overall energy gain from Harmonia.

In Figure 61 we see an average loss in performance of 0.36% across all the applications using Harmonia (FG+CG) excluding *MaxFlops* and *DeviceMemory*, with up to 3.6% maximum slow-down in Streamcluster. This illustrates the efficacy of Harmonia in optimizing energy efficiency under performance constraints by pushing the hardware to operate at its balance point for each application's kernel. We also note that employing CG tuning alone results in an average performance loss of 2.2% compared to the baseline, with up to 27% maximum slow-down for Streamcluster. This is due to the lack of any performance feedback in CG tuning. Thus, while the use of CG tuning alone achieves energy savings comparable to Harmonia, the performance-driven FG tuning loop ensures much better performance across all applications and avoids outliers resulting in better overall ED^2 gains.

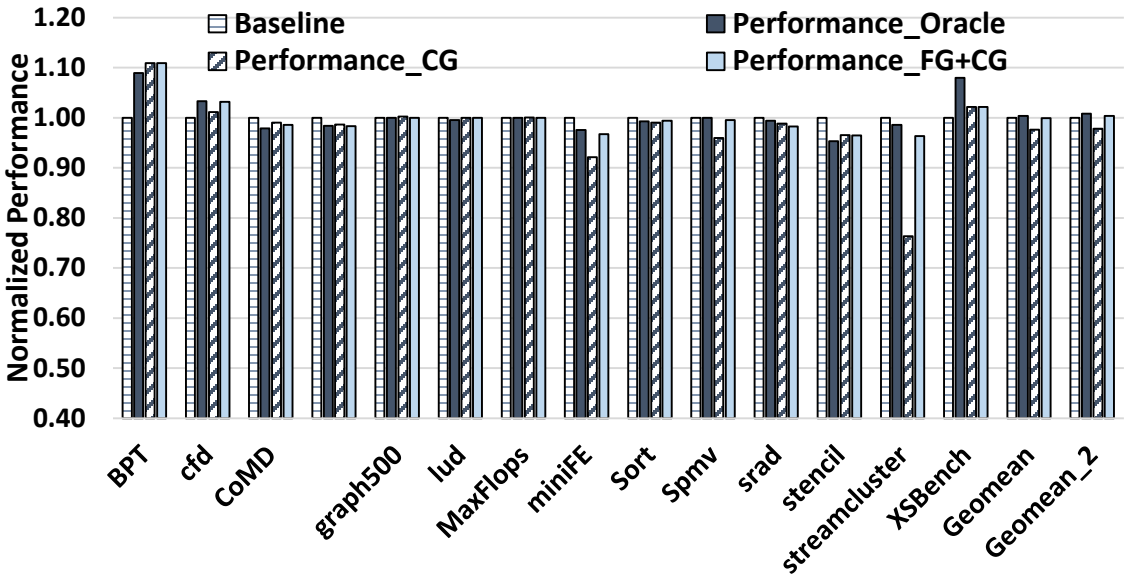


Figure 61: Overall performance from Harmonia.

There are three applications that are worth noting here. They are *BPT*, *CFD* and *XSBench*. These applications see an *improvement* in performance with Harmonia. *BPT* sees an 11% performance gain, while *CFD* and *XSBench* each realize 3% performance improvement. In the baseline hardware configuration, we observed heavy cache thrashing and pollution accompanied by significant memory divergence. Thus lowering the number of active CUs via power gating also improves performance by reducing interference in the L2 cache. Harmonia captures the optimal compute to memory balance point via the sensitivity to CU count for these applications.

We also observe an average power savings of 12% across the entire GPU card, with the maximum savings of 19% for *Stencil*. Figure 62 shows the average total power savings for each application. During application phases less sensitive to memory bandwidth such as *EAM_Force_1* in *CoMD*, reducing memory bus frequency just enough without increasing memory related stalling and exposing memory access latency results in reduction of memory bus power and thereby savings of the overall board power -

GPUCardPwr. Notice that more memory power saving would be possible if HD7970's memory interface supports multiple voltages. On the other hand, *AdvanceVelocity* in *CoMD* is memory intensive with moderate compute demands and Harmonia finds the balance points by reducing compute power without performance loss. Similarly, due to poor thread level parallelism (kernel occupancy of 30%) in Bottomscan, the main kernel in *Sort*, the memory bus frequency could be reduced down to 475MHz without hurting performance with a 12% overall GPU card power savings.

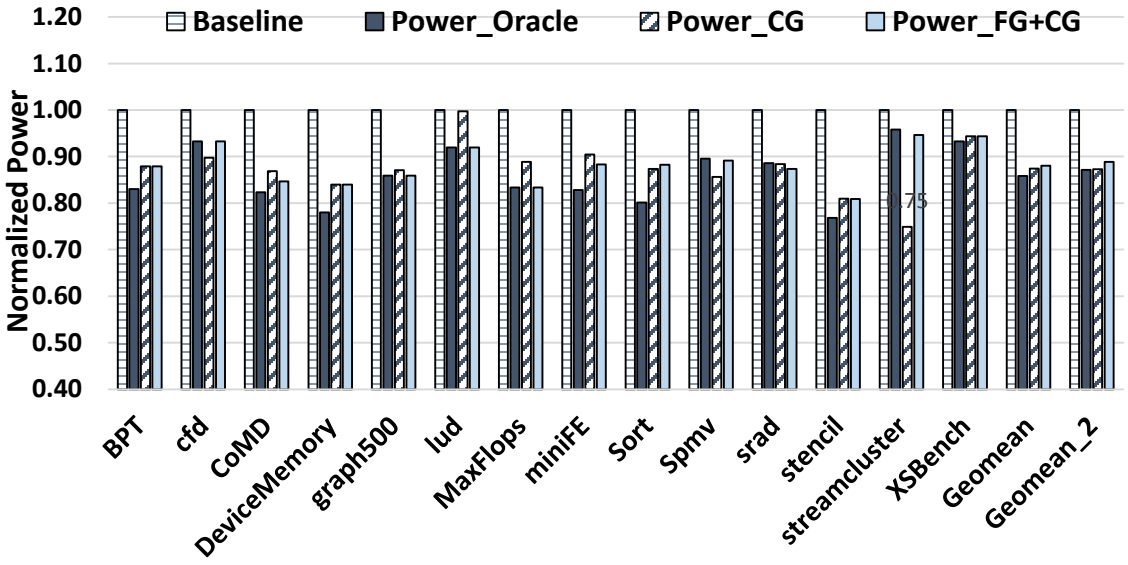


Figure 62: Overall power savings from Harmonia.

7.6.2 Adaptation Behavior

In this section, we explain the adaptation behavior of Harmonia in response to workload changes.

Intra-kernel Phase Changes: Figure 63 illustrates the time-varying workload behavior of the main kernel *BottomStepup* in *Graph500*. The Y-axis indicates the total number of compute instructions (VALUInsts), memory reads (VFetchInsts) and memory

writes (VWriteInsts) executed in eight successive iterations, each iteration lasting anywhere from 0.9 to 5.6 seconds. This kernel is performing a Breadth First Search. Note that the raw total number of instructions across iterations can vary significantly. The memory fetch unit was active anywhere from 40% to 80% of the total kernel execution time. The compute sensitivity was high 95% of the time and branch divergence was significant. As a result Harmonia mostly utilized all 32 CUs and 1GHz compute frequency to speed up execution of threads serialized by branch divergence. However, bandwidth sensitivity changed frequently between medium and low as the predictor adapted to input argument changes and consequent changes in demand for memory bandwidth. Thus, through CG and FG tuning, memory frequency dithered between 925MHz and 775MHz. Figure 64 shows the distribution of time spent at the different memory bus frequencies in Harmonia over the kernel's entire execution.

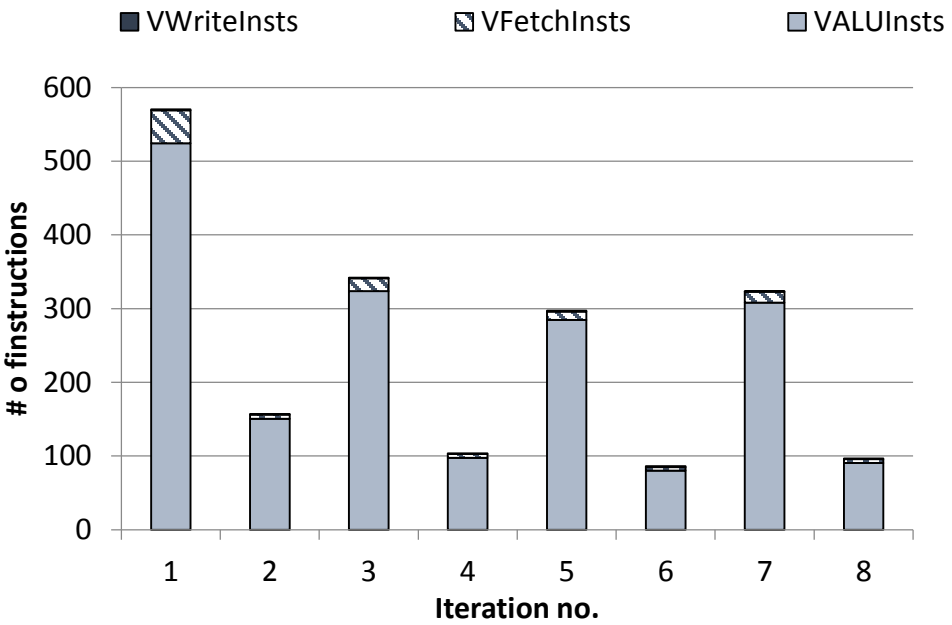


Figure 63: Behavior of Graph500.BottomStepUp over time.

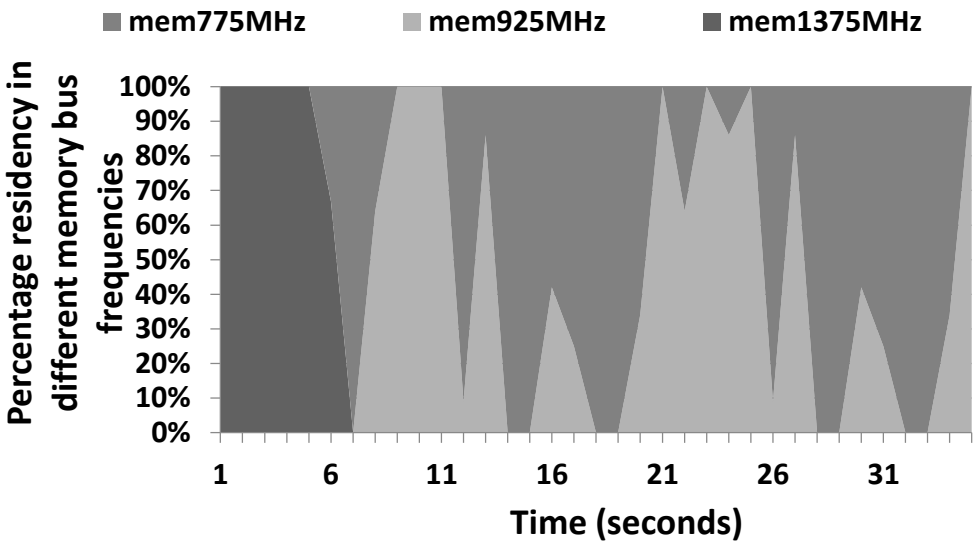


Figure 64: Memory bus frequency residency change as time progresses in Graph500.BottomStepUp.

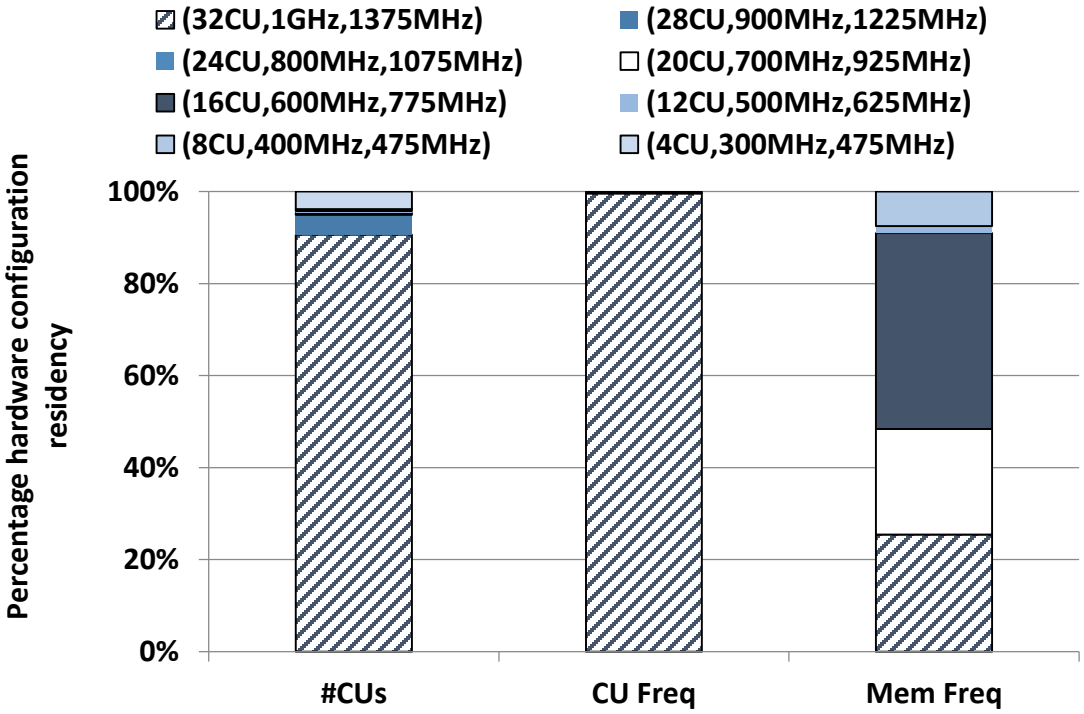


Figure 65: Residency of the hardware tunables in Graph500.

Inter-kernel Phase Changes: We observe that the ops/byte value of *Graph500* varies from 0.54 to 264. Figure 65 shows the fraction of time each hardware tunable spent in each power state as Harmonia moved the hardware towards the right balance point. For this application due to high branch divergence, Harmonia tunes to the maximum compute frequency (single state in CUFreq column). This was accompanied by tuning of the CU count and memory bandwidth that reduced power. The #CUs column shows that about 90% of the time 32 CUs were used; the remaining time was spent in dithering between 4, 8, 12 and 16 CUs based on time-varying ops/byte. The memory bus frequency varied between 1375MHz (25% of the time), 925MHz (23%), 775MHz (42%) and 475MHz (8%) as the operational intensity of the three kernels in *Graph500* varied from lows of 0.64 ops/byte to bursts of 264 ops/byte.

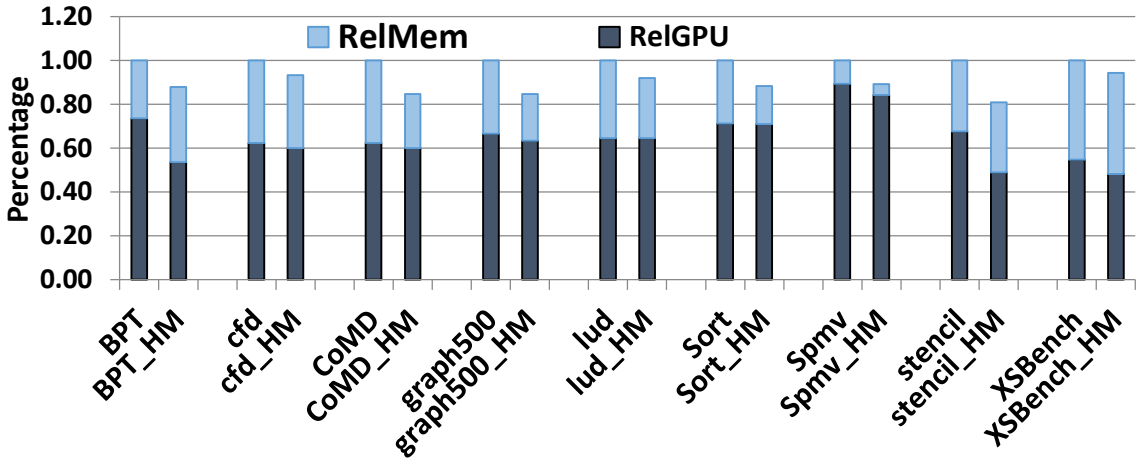


Figure 66: Relative GPU and memory power consumption.

Coordinated Power Sharing: Figure 67 shows the GPU and memory power consumption across a subset of the applications with both baseline and Harmonia (HM), relative to the measured total power for GPU and memory. Here the total power is

normalized with respect to the baseline. Power due to remaining elements on the board are not shown since they are roughly constant. We observe that out of the average 12% power savings, 64% of the savings come from varying the GPU compute configuration. The remaining 36% comes from changing the memory bus frequencies. We believe that it is feasible to achieve far more power savings from memory configuration changes if voltage scaling is applied while lowering bus speeds. In our current setup we were not able to scale voltage of the memory bus interface.

Another interesting observation is that most often, Harmonia adjusted CU counts and memory bus frequencies rather than the full range of compute frequencies. This behavior was consistent across all applications. In fact compute frequency and voltage scaling alone achieved only an average ED^2 gain of 3% with a 1% performance loss compared to the baseline. The reason is two-fold: i) parallel execution and data movement demands are inherent to the application and govern demanded ops/byte values which vary widely across applications, thus making available hardware resources in excess of these demands is not helpful, and ii) as explained in Section 7.2.3, architectural clock domain crossings reduce opportunities for compute frequency to improve energy efficiency for memory intensive applications.

Algorithm Convergence and Relative Impact of CG vs. FG Tuning: Figure 67 shows the relative contributions from CG and FG tuning for energy efficiency improvement across a subset of applications. In most applications CG tuning requires only one iteration. Even in applications with a small number of iterations (insufficient for feedback driven FG tuning) CG is very effective in rapidly reaching a lower power operation point often in a single iteration. An example is *XS-Bench* which executes only 2 iterations for each of its kernels. Even here, Harmonia is able to save 4% overall GPU card power while improving overall application performance by 2%, resulting in 9% energy efficiency gain. However, in certain cases such as *LUD, SPMV*, due to prediction outliers or lack of performance feedback, CG can omit additional power savings opportunity or

degrade performance. In such cases FG tuning plays a crucial role. The FG step typically takes an additional 3 to 4 iterations to converge. In HPC applications, many kernels represent iterative computations that typically execute several times to converge with minimal algorithmic error. For such kernels, the overhead of FG tuning is amortized over successive kernel invocations. Therefore, both steps are necessary in order to have Harmonia cover a broad range of workloads.

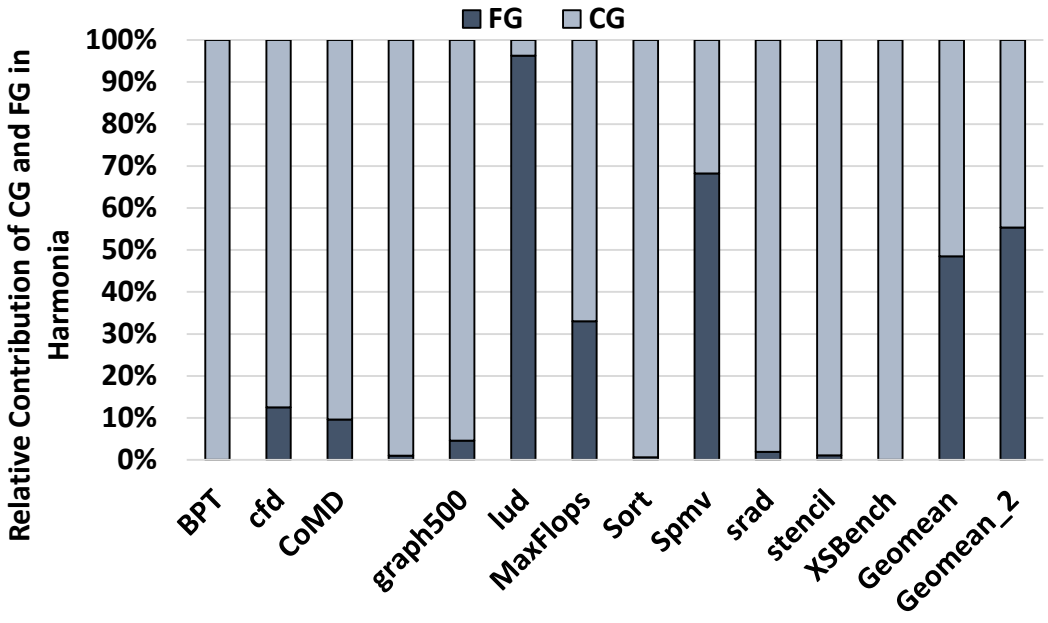


Figure 67: Relative contributions of CG vs. FG in Harmonia.

Sensitivity Predictors: The prediction error between measured and estimated bandwidth and compute sensitivities is 3.03% and 5.71% respectively across all the applications used in this study. Since our goal is to develop simple, effective and practical sensitivity predictors that can be easily implemented in hardware, we found that simple linear regression based sensitivity models such as the ones proposed in this thesis combined

with an effective binning methodology can significantly help improve the accuracy of the predictors.

7.6.3 Summary of Key Insights

In this section we summarize our main results and insights:

1. Compute and memory behavior are fundamentally performance coupled. Optimizing only compute or memory behavior has limited benefits. It is necessary to balance the time and energy costs of compute and memory to improve energy efficiency with minimal performance loss.
2. Scaling parallelism (number of active CUs) and memory bandwidth is more effective than scaling CU frequency since it has a greater impact on ops/byte behaviors. Note that modern systems rely primarily on scaling compute frequency for energy efficiency gains.
3. Clock domain crossings and interconnect sizing have non-trivial impact on energy efficiency.
4. Feedback driven fine-grained adjustments are effective in correcting coarse grain tuning mispredictions or longer-term changes in learned behaviors.
5. Improving energy efficiency can lead to improvements in execution time due to reduction of interference in shared resources, e.g., cache or interconnect.
6. With advanced packaging technologies, compute and memory will share tighter package power envelopes, e.g., compute with stacked memory [74]. Coordinated power management and the concept of hardware balance will become more important in such systems.

7.7 Conclusions

This component of the thesis applies the notion of hardware balance to the development of a practical scheme for the coordinated management of compute and memory power in a high performance discrete GPU platform. By tracking the time-varying

relative compute and memory demands of applications, the corresponding hardware power configurations of the core and memory system can be set to reduce overall platform power and thereby improve energy efficiency with minimal compromises in performance. This research work was published at *International Symposium of Computer Architecture (ISCA) 2015 [89]*.

CHAPTER 8

CONCLUSIONS AND FUTURE WORK

Following the end of Dennard scaling, the major system challenge facing the industry is to sustain performance scaling with Moore's Law while preparing for the transition to post-CMOS technologies. As W. Dally noted [29], performance scaling in the post-Dennard dark-silicon era will involve improving efficiency of power and energy usage instead of clock speed scaling. The modern industry shift towards heterogeneous computing is largely motivated by energy and power efficiencies. While such tightly coupled systems benefit from reduced latency and improved performance, they also give rise to new management challenges due to phenomena such as physical asymmetry in thermal and power signatures between the diverse elements and functional asymmetry in performance.

The objective of this dissertation is to understand the relationships between the physical phenomena (power and thermal coupling), functional behaviors (direct and indirect performance coupling) and their impact on system performance, and utilize that understanding towards proposing abstractions and guiding principles towards managing such coupling effects. Towards this end, the final chapter of this dissertation summarizes the aspects of various coupling effects in modern heterogeneous processors, along with key insights on coordinated management to tackle such effects while improving system level performance, power efficiency and energy. This chapter also identifies related open problems that merit additional work in future.

8.1 Summary of Key Contributions

This thesis demonstrates the concepts of thermal coupling and performance coupling and the needs for coordinated management of functional and physical resources of a heterogeneous system. It shows that this interaction between processor physics and

performance is not an artifact of an architecture instance, but is fundamental to the operation of many core and heterogeneous architectures.

Specially, following subsections describe the key contributions of this dissertation research.

8.1.1 Analysis and Abstractions of New Management Challenges

One of the main contributions of this thesis is analyzing the interactions between physics, such as heat transfer and power delivery, and system level performance in a heterogeneous processor. Our studies in this research point out that various forms of coupling effects between major subsystems in a heterogeneous processor (CPU, GPU, memory) are beginning to dominate energy, power and performance efficiencies. This phenomenon will become more pronounced at future technology nodes as die sizes shrink and the trend towards Systems-on-Chip style processors grows with more components getting integrated. We illustrate the distinct and asymmetric power and thermal signatures of the various integrated elements that exist in modern processors, which is turn causes unequal thermal coupling effects among multiple elements of the processor. We also demonstrate the functional dependencies and performance coupling between diverse compute and memory subsystems of the chip. This conceptual understanding forms the foundation of the thesis as well as opens the door to many open problems as described in the future work Section 8.2.

8.1.2 Characterization of Emerging Classes of Applications

This dissertation performs an in-depth characterization of the emerging class of compute applications on integrated CPU-GPU processors as well as high performance GPUs. We show that, unlike traditional applications that are suitable for a particular device, emerging compute applications require cooperative execution of both CPU and GPU cores with time-varying redistribution of the compute intensities. In addition, we also illustrate that at any given time, compute throughput and bandwidth demands of an application must match the

hardware compute and memory costs in order to operate most efficiently. Any excess available bandwidth than the bytes transferred is a waste of power or hardware cost without any benefit in performance. This highlights the need for coordinated management of the power states and hardware configurations of modern heterogeneous processors. The remaining contributions focus on building guiding principles, metrics and power management algorithms to manage the interactions between various types of coupling, workload execution profiles and optimize overall system level performance, power and energy under various constraints.

8.1.3 Models and Run-time Metrics for Coordinated Management

One of the key contributions of this thesis is the demonstration of simple, practical and effective ways of dealing with the different types of coupling effects in modern heterogeneous processors through actual hardware measurements and implementation. To this end, the first step is to identify run-time parameters and metrics that can accurately model the effects of thermal coupling and performance coupling between multiple compute or compute and memory elements of the heterogeneous architecture. This dissertation establishes criteria for when thermal coupling effects are detrimental in an APU and must be balanced with the needs of performance coupling. It identifies performance metrics and their statistical behavior that captures the point of performance dependency of the GPU on the CPU, termed as critical CPU P-state, in order to facilitate computation both using an offload as well a concurrent computation programming model.

The dissertation also identifies and categorizes behaviors that have a substantive impact on frequency sensitivity of the CPU and GPU in an APU and uses regression techniques to construct an analytic model of frequency sensitivity. Finally, it illustrates that hardware balance is sensitive to several expected as well as non-obvious behaviors in the hardware platform. It derives online predictors for assessing the performance sensitivity of application kernels to each of the hardware tunables (#CUs, compute frequency, memory bandwidth) in a tightly coupled high performance GPU and memory system.

8.1.4 Coordinated Management under Global Constraints

Power management solutions which do not understand and account for the interactions between power delivery, thermal coupling and performance coupling can have undesired consequences on system level metrics such as performance and energy efficiency. Through real-hardware measurements, this thesis demonstrates that effective power management in heterogeneous processors requires coordinated management across physically and functionally diverse compute and memory components. We demonstrate this through the development, implementation, and evaluation of chip-scale coordinated power management algorithms to optimize system level metrics such as performance and energy efficiency under global constraints such as thermal limits and power budgets.

First, we propose a dynamic power-management approach called *cooperative boosting (CB)* to allocate power dynamically between the CPU and GPU in a manner that balances thermal coupling against the needs of performance coupling by modifying the CPU P-state at run-time. Here goal is to optimize performance under the given power and thermal constraints of the die. Next, we propose coordinated management of the power states in both the CPU and GPU that encapsulates CPU-GPU frequency sensitivity models along with power management algorithms, *DynaCo*, to make the coupled operation as energy efficient as possible with minimal performance impact. Finally, we propose a coordinated two-level power management scheme, *Harmonia*, to tune platform balance between compute throughput and memory bandwidth by i) a coarse-grain adjustment of the GPU and the memory power states based on online sensitivity prediction, ii) followed by fine-grain tuning through close-loop performance feedback.

8.2 Future Work

This thesis establishes the concepts of thermal coupling and performance coupling and apply these concepts to develop novel power management algorithms for tackling the

various coupling effects in heterogeneous CPU-GPU and GPU-Memory systems. Here we explore processor physics on-die and its interactions with performance and power management. However, the concepts developed in this thesis can be applied to many other potential areas in different forms, factors and packages, likely leading to new and valuable discoveries. A few important future research directions are discussed in the following subsections.

8.2.1 Thermal and Performance Coupling Management in CPU-GPU-Memory Systems

One of the natural extensions of the work done in this thesis is applicability of the thermal and performance coupling analysis to a heterogeneous system with integrated CPU, GPU and memory (DRAM) subsystems. In future, with the advent of High Bandwidth Memory (HBM), Hybrid Memory Cube (HMC) and other die stacking memory technologies [93][39][56], heterogeneous architectures with integrated CPU, GPU and memory will emerge [131] sharing the package or chip-level thermal design power envelope (TDP). The integration of processors or other computing logic within the 3D die-stacked memories enables processing-in-memory (PIM) capabilities within each memory module. The advantage of this organization is the resultant high bandwidth, low latency memory accesses from the in-stack processing layer. However, 3D stacking of the processor layer and the memory layers incurs high power density with less efficient heat dissipation. This can potentially lead to adverse thermal coupling affects impacting system level power efficiency and performance. Power states in such heterogeneous die-stacked systems must be coordinated to mitigate the detrimental effects of thermal coupling. On the other hand, data placement and compute dispatch management can lead to various degrees of performance coupling in such systems. Thus, effective sharing of the power and thermal envelope between the memory system, the CPU and the GPU needs to be investigated and is an important research direction for future.

8.2.2 Thermal and Performance Coupling Management in Datacenters

Optimizing the performance of a machine within a power budget is an important problem. The "machine" could be a single GPU consisting of multiple CUs or a single node consisting of multiple sockets each consisting of a multi-core CPU system / or multiple CPU cores, GPU cores and main memory. A machine could also consist of many such nodes together in form of a data center/HPC cluster or a combination of many such clusters. This thesis demonstrates the effects of performance coupling and functional dependencies within the compute and memory elements of a single node system consisting of a processor (CPU and/or GPU) along with its memory. The work in this thesis can be expanded further to include multiple nodes in future systems. In HPC and Cloud Datacenters, often a job is split among multiple nodes and there are data dependencies between the tasks in each node. A node consuming high power due to the nature of tasks its executing may heat up adjacent nodes leading to thermal coupling. If the effects of thermal coupling are not managed carefully, it can lead to poor performance, higher cooling needs, increased costs, or even fatal errors and reliability issues. Performance metrics and models may be required to capture the effects of heat transfer among multiple nodes as well as the data dependencies among them. One can optimize energy efficiency and performance in this heterogeneous multi-node cluster by managing the complex interactions among heat dissipation, heat transfer, power delivery, data dependencies and power management.

8.2.3 Workload Scheduling and Computation Balancing in Heterogeneous Systems

Thermal capacity is a shared physical resource in the processor die where thermal coupling from one component to another, both spatially and temporally, can have an adverse performance impact on applications running on adjacent components. It becomes quite difficult to provide resource fairness to the applications that are co-located. Modern

power management algorithms do not account for resource sharing, co-location of threads/applications, QoS demands and distinct time-varying thread/application level priorities. Lack of this knowledge often results in reaching peak temperatures very quickly resulting in undesired thermal throttling of the co-located workloads, thereby penalizing applications leading to unfairness. This has a severe performance impact. In the future one can explore the effects of workload placement and concurrent usage of CPU and GPU to mitigate the effects of thermal coupling and manage chip-scale thermal capacity. For example, if the thermal signature of a particular phase of a workload executing on the CPU indicates much faster and higher heat dissipation as compared to it executing on the GPU, run-time system software may decide to run that portion of the workload on the GPU instead. Coordinated power management combined with system software driven workload balancing and data placement is the key to performance and power efficiencies in future heterogeneous processors.

8.3 Conclusions

This thesis demonstrates via detailed measurement and analysis the relationship among and time-varying workload execution profiles, aggressive DVFS-based multicore power management, thermal capacity, thermal interactions, and functional dependencies between heterogeneous compute and memory systems. It proposes abstractions with which to articulate and reason about how physical behaviors affect system-level performance (thermal signatures, and thermal pollution). This leads to the notions of performance and thermal coupling and the understanding that power management must be aware of physical behaviors to avoid detrimental impact on performance. This interaction between thermal coupling and performance coupling is not an artifact of an architecture instance, but is fundamental to the operation of many core architectures. We argue that this awareness has the potential to exert significant influence over the design of future power and performance management algorithms.

REFERENCES

[1] Advanced Configuration and Power Interface (ACPI), Specification, http://www.acpi.info/spec.htm

[2] AMD APP SDK, http://developer.amd.com/tools/heterogeneous-computing/amd-accelerated-parallel-processing-app-sdk/.

[3] AMD PowerTune Technology whitepaper, 2010

[4] M. Arora, S. Nath, S. Mazumdar, S. Baden, D. Tullsen, "Redefining the Role of the CPU in the Era of CPU-GPU Integration," IEEE Micro2012.

[5] M. Arora, S. Manne, I. Paul, N. Jayasena, D. Tullsen, "Understanding idle behavior and power gating mechanisms in the context of modern benchmarks on CPU-GPU integrated systems", *HPCA 2015*.

[6] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, K. A. Yelick, "The landscape of parallel computing research: A view from Berkeley," Technical Report UCB/EECS-183, 2006.

[7] R. Ayoub, R. Nath, T. Rosing, "JETC: Joint Energy Thermal and Cooling Management for Memory and CPU Subsystems in Servers," HPCA 2012.

[8] Peter Bailis, V. J. Reddi, S. Gandhi, D. Brooks, M. Seltzer, "Dimetrodon: Processor-level Preventive Thermal Management via Idle Cycle Injection," DAC 2011.

[9] P. Balaprakash, D. Buntinas, A. Chan, A. Guha, R. Gupta, S. Narayanan, A. Chieny, P. Hovland, B. Norris, "An exascale workload study," SCC 2012.

[10] W.L. Bircher, L.K. John, "Complete System Power Estimation: A Trickle-Down Approach Based on Performance Events," ISPASS 2007.

[11] W.L. Bircher, M. Valluri, J. Law, L.K. John, "Runtime Identification of Microprocessor Energy Saving Opportunities," ISLPED 2005.

[12] BIOS and Kernel Developer's Guide: http://support.amd.com/us/Processor_TechDocs/42300_15h_Mod_10h-1Fh_BKDG.pdf

[13] P. Blinzer, "The HSA System Architecture Requirements: An overview", APU 2013.

[14] D. Brooks, M. Martonosi, "Dynamic thermal management for high-performance microprocessors," HPCA 2001.

[15] W.M. Brown, P. Wang, S.J. Plimpton, A.N. Tharrington, "Implementing molecular dynamics on hybrid high performance computers- short range forces," Compute Physics Communications 2011.

[16] M. Butler, L. Barnes, D.D. Sarma, B. Gelinas, Bulldozer, "An approach to multithreaded compute performance", IEEE Micro 2011.

[17] S. Che, B. Beckmann, S. Reinhardt, K. Skadron, "Pannotia: Understanding Irregular GPGPU Graph Applications", IISWC 2013.

[18] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, Lee S-H, K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," IISWC 2009.

[19] S. Che, J. W. Sheaffer, M. Boyer, L. Szafaryn, K. Skadron, "A characterization of the Rodinia benchmark suite with comparison to contemporary CMP workloads," IISWC 2010.

[20] L. Chen, X. Huo, G. Agrawal, "Accelerating map-reduce on a coupled CPU-GPU architecture," SC 2012.

[21] M. Cho, C. Kersey, M. P. Gupta, N. Sathe, S. Kumar, S. Yalamanchili, S. Mukhopadhyay, "Power Multiplexing for Thermal Field Management in Many Core Processors," IEEE Transactions on Components, Packaging, and Manufacturing Technologies (TCPMT), 2013.

[22] J.W.Choi, D. Bedard, R. Fowler, R. Vuduc, "A Roofline Model of Energy", IPDPS 2013.

[23] J. Choi, C. Cher, H. Franke, H. Haman, A. Weger, P. Bose, "Thermal-aware task scheduling at the system software level," ISLPED 2007.

[24] A. K. Coskun, T. S. Rosing, K. C. Gross, "Temperature management in multiprocessor SoCs using online learning," DAC 2008.

[25] A. K. Coskun, T. S. Rosing, D. A. Alonso, J. Leblebici, J. Ayala, "Dynamic thermal management in 3D multicore architectures," DATE 2009.

[26] J. Cong, S. W. Chung, K. Skadron, "Recent Thermal Management Techniques for Microprocessors," ACM Computing Surveys 2012.

[27] M. Curtis-Maury, A. Shah, F. Blagojevic, D. Nikolopoulos, B.R. de Supinski, M. Schulz, "Prediction Models for Multi-dimensional Power-Performance Optimization on Many Cores," PACT 2008.

[28] M. Daga, M. Nutter, "*Exploiting Coarse-grained Parallelism in B+ Tree Searches on APUs", 2nd Workshop on Irregular Applications: Architectures & Algorithms (IA3)*, November 2012.

[29] W. J. Dally, “It’s About the Power: An Architect’s View of Interconnect,” Keynote IEEE IITC, 2012.

[30] A. Danalis, G. Marin, C. McCurdy, J. Meredith, P. Roth, K. Spafford, V. Tipparaju, J. S. Vetter, “The scalable heterogeneous computing (SHOC) benchmarking suite”, GPGPU 2010

[31] H. David, C. Fallin, E. Gorbatov, U. R. Hanebutte, O. Mutlu, “Memory Power Management via Dynamic Voltage/Frequency Scaling”, ICAC 2011.

[32] H. David, E. Gorbatov, U.R. Hanebutte, R. Khanna, C. Le, “RAPL: Memory Power Estimation and Capping”, ISLPED 2010.

[33] Q. Deng, D. Meisner, L. Ramos, T. F. Wenisch, R. Bianchini, “MemScale: Active Low-Power Modes for Main Memory”, ASPLOS, 2011.

[34] Q. Deng, D. Meisner, A. Bhattacharjee, T. F. Weinisch, R. Bianchini, “CoScale: Coordinating CPU and Memory System DVFS in Server Systems”, MICRO, 2012.

[35] Q. Deng, D. Meisner, A. Bhattacharjee, T. F. Wenisch, R. Bianchini, “MultiScale: Memory System DVFS with Multiple Memory Controllers”, ISLPED 2012.

[36] R. Dennard, et al., “Design of ion-implanted MOSFETs with very small physical dimensions,” IEEE Journal of Solid State Circuits, vol. SC-9, no. 5, Oct. 1974.

[37] G. Diamos, A. Kerr, S. Yalamanchili, N. Clark, “Ocelot: A Dynamic Compiler for Bulk-Synchronous Applications in Heterogeneous Systems”, PACT 2010.

[38] J. Donald, M. Martonosi, "Techniques for multicore thermal management: classification and new exploration," ISCA 2006.

[39] “Elpida begins sample shipments of ddr3 sdram (x32) based on tsv stacking technology,” http://www.elpida.com/en/news/2011/06-27.html, June 2011, elpida2011.

[40] H. Esmaeilzadeh, E. Blem, R. Amant, K. Sankaralingam, D. Burger, “Dark Silicon and the End of Multicore Scaling”, ISCA 2011.

[41] W. Felter, K. Rajamani, T. Keller, C. Rusu, “A Performance-Conserving Approach for Reducing Peak Power Consumption in Server Systems”, ICS 2005.

[42] Folding At Home, http://folding.stanford.edu/English/Download

[43] W. Fung et al., “Dynamic warp formation and scheduling for efficient gpu control flow”, MICRO 2007.

[44] R. Hameed, W. Qadeer, M. Wachs, O. Azizi, A. Solomatnikov, B.C. Lee, S. Richardson, C. Kozyrakis, M. Horowitz, “Understanding sources of inefficiency in general-purpose chips”, ISCA 2010.

[45] V. Hanumaiah, S. Vrudhula, "Temperature- Aware DVFS for Hard Real-Time Applications on Multicore Processors," IEEE Transactions on Computers 2012.

[46] M. Heroux, D. Doerfler, P. Crozier, J. Willenbring, H.C. Edwards, A. Williams, M. Rajan, E. Keiter, H. Thornquist, R. Numrich, “Improving performance via mini-applications”, SAND2009-5574

[47] S. Hong, H. Kim, “An Analytical Model for a GPU Architecture with Memory-Level and Thread-Level Parallelism Awareness”, ISCA 2009.

[48] S. Hong, H. Kim, "An integrated GPU power and performance model," ISCA 2010.

[49] C. Hsu, W. Feng, "Effective dynamic voltage scaling through CPU-boundedness detection," Lecture Notes in Computer Science 2004.

[50] Z. Hu, D. Brooks, V. Zyuban, and P. Bose, "Microarchitecture-level power-performance simulators: modeling, validation and impact on design," MICRO 2003.

[51] W. Huang, E. Humenay, K. Skadron, M. R. Stan, “The need for a full-chip and package thermal model for thermally optimized IC designs”, ISLPED 2005.

[52] W. Huang, M. Stan, K. Sankaranarayanan, R. Ribando, and K. Skadron, "Many-core design from a thermal perspective," DAC 2008.

[53] W-L. Hung, Y. Xie, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, "Thermal-Aware Allocation and Scheduling for Systems-on-a-Chip Design," DATE 2005.

[54] X. Huo, V.T. Ravi, G. Agrawal, "Porting irregular reductions on heterogeneous CPU-GPU configurations," HiPC 2011.

[55] “ITRS. International technology roadmap for semiconductors”, 2012 update, 2013, http://www.itrs.net.

[56] “JEDEC publishes breakthrough standard for wide I/O mobile DRAM,” http://www.jedec.org/news/pressreleases/jedecpublishes- breakthrough-standard-wide-io-mobile-dram, Jan2012.

[57] M. K. Jeong et al, “A QoS-Aware Memory Controller for Dynamically Balancing GPU and CPU Bandwidth Use in an MPSoC”, DAC 2012.

[58] W. Jia, K. Shaw, and M. Martonosi, "Stargazer: Automated Regression-Based GPU Design Space Exploration," IEEE ISPASS 2012

[59] I. Karlin, “LULESH programming model and performance ports overview”, LLNL-TR-608824

[60] S. Kaxiras, M. Martonosi, "Computer Architecture Techniques for Power Efficiency," Synthesis Lectures on Computer Architecture.

[61] O. Kayiran, A. Jog, M. T. Kandemir, and C. R. Das, “Neither More Nor Less: Optimizing Thread-level Parallelism for GPGPUs,” PACT 2013.

[62] S. Keckler,W. Dally, B. Khailany, M. Garland, and D. Glasco, “GPUs and the Future of Parallel Computing,” *IEEE Micro,* 2011.

[63] A. Kerr, E. Anger, G. Hendry, and S. Yalamanchili. “Eiger: A framework for the automated synthesis of statistical performance models.” *1st* Workshop on Performance Engineering and Applications (WPEA), held with HiPC 2012.

[64] G. Kestor, R. Gioiosa, D. Kerbyson, and A. Hoisie, “Quantifying the Energy Cost of Data Movement in Scientific Applications,” in *International Symposium on Workload Characterization (IISWC),* 2013.

[65] R. Kumar, D. Tullsen, N. Joupppi, P. Ranganathan, "Heterogeneous Chip Multiprocessors," IEEE Computer 2005.

[66] E. Kursun, C. Y. Cher, "Temperature Variation Characterization and Thermal Management in Multicore Architectures," IEEE Micro 2009.

[67] J. H. Laros III, K. T. Pedretti, S. M. Kelly, W. Shu, and C. T. Vaughan, “Energy based performance tuning for large scale high performance computing systems,” HPC 2012

[68] J. Lee, H. Kim, "TAP: A TLP-aware cache management policy for a CPU-GPU heterogeneous architecture," HPCA 2012.

[69] J. Lee, N. Kim, “Optimizing throughput of power- and thermal-constrained multicore processors using DVFS and per-core power-gating”, DAC 2009.

[70] J. Lee, V. Sathish, M. Schulte, K. Compton, and N. Kim, "Improving throughput of power-constrained GPUs using dynamic voltage/frequency and core scaling," PACT 2011.

[71] M. Lee, S. Song, J. Moon, J. Kim, W. Seo, Y. Cho, and S. Ryu, “Improving GPGPU Resource Utilization Through Alternative Thread Block Scheduling,” HPCA 2014.

[72] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, V. J. Reddi, “GPUWattch: Enabling energy optimization in GPGPUs”, ISCA 2013.

[73] J. Li and J. Martinez, "Dynamic power-performance adaptation of parallel computation on chip multiprocessors," HPCA 2006.

[74] G. Loh, “3D-stacked memory architectures for multi-core processors,” ISCA 2008.

[75] C. Luk, S. Hong, and H. Kim, "Qilin: Exploiting parallelism on heterogeneous multiprocessors with adaptive mapping," MICRO 2009.

[76] M. Mantor, M. Houston “AMD Graphic Core Next”, AMD Fusion Developer Summit 2011.

[77] A. Mclaughlin, I. Paul, J. Greathouse, S. Manne, S. Yalamanchili, “A Power Characterization and Management of GPU Graph Traversal”. Architectures and Systems for Big Data, 2014.

[78] G. Moore, “Cramming More Components onto Integrate Circuits”, Electronics, magazine, 1965.

[79] R. Mukherjee and S. O. Memik, "Physical aware frequency selection for dynamic thermal management in multi-core systems," ICCAD 2006.

[80] S. Murali, A. Mutapcic, D. Atienza, R. Gupta, S. P. Boyd, L. Benini, and D. Micheli, "Temperature control of high-performance multi-core platforms using convex optimization," DATE 2008.

[81] R.C.Murphy, K.B.Wheeler, B.W.Barett, J.A.Ang, “Introducing the Graph500”, Cray User’s Group (CUG), May 2010.

[82] V. Narasiman et al. Improving GPU performance via large warps and two-level warp scheduling, MICRO 2011

[83] S. Nussabaum, "AMD Trinity APU," HotChips 2012.

[84] “NVidia’s next generation compute architecture: Fermi”, Whitepaper.

[85] NVIDIA. Cuda reference manual. http://developer.download.nvidia.com/compute/cuda/2_3/toolkit/docs/CUDA_Reference_Manual_2.3.pdf, 2009.

[86] “The OpenCL Specification version 2.0”, Khronos OpenCL Workng Group.

[87] S. Pakin, C. Storlie, M. Lang, R. Fields III, E. Romero, C. Idler, S. Michalak, H. Greenberg, J. Loncaric, R. Rheinheimer, G. Grider, J. Wendelberger, “ Power usage of production supercomputers and production workloads”, SC 2012.

[88] Y. Y. Pan, T. Zhang, “Improving VLIW processor performance using three-dimensional (3d) DRAM stacking,” ASAP 2009.

[89] I. Paul, W. Huang, M. Arora, S. Yalamanchili, “Harmonia: Balancing compute and memory power in high performance GPUs”, ISCA 2015.

[90] I. Paul, S. Manne, M. Arora, W.L. Bircher, S. Yalamanchili, “Cooperative boosting: needy versus greedy power management”, ISCA 2013.

[91] I. Paul, V. Ravi, S. Manne, M. Arora, S. Yalamanchili, “Coordinated Energy Management in Heterogeneous Processors”, SC 2013 [best paper award finalist].

[92] I. Paul, V. Ravi, S. Manne, M. Arora, S. Yalamanchili, “Coordinated Energy Management in Heterogeneous Processors”, Journal of Scientific Programming, 2014.

[93] J. T. Pawlowski, “Hybrid memory cube (HMC),” in Proceedings of Hot Chips 23, 2011.

[94] M. D. Powell. M. Gomaa, and T. N. Vijaykumar, "Heat-and-run: leveraging SMT and CMP to manage power density through the operating system," ASPLOS 2004.

[95] A. Raghavan, Y. Luo, A. Chandawalla, M. Papaefthymiou, K. P. Pipe, T. F. Wenisch, and M. M. K. Martin, "Computational Sprinting," HPCA 2012.

[96] P. Ranganathan, “Recipe for Efficiency: Principles of Power-Aware Computing”, ACM Communications 2010.

[97] V.T. Ravi, W. Ma, D. Chiu, G, Agrawal, “Compiler and runtime support for enabling generalized reduction computations on heterogeneous parallel configurations”, ICS 2010.

[98] V.T. Ravi, G. Agrawal, "A dynamic scheduling framework for emerging heterogeneous systems," HiPC 2011.

[99] E. Rotem, A. Naveh, D. Rajwan, A. Ananthakrishnan, E. Weisman, "Power Management Architectures of the Intel Microarchitecture Code-Named Sandy Bridge," IEEE Micro 2012.

[100] B. Rountree, D.K. Lowenthal, S. Funk, V. Freeh, B.R. de Supinski, M. Schulz, “Bounding energy consumption in large-scale MPI programs”, SC 2007

[101] B. Rountree, D.K. Lowenthal, B.R. de Supinski, M. Schulz, V. Freeh, T. Bletsch, “Adagio: Making DVS Practical for Complex HPC Applications”, ICS 2009.

[102] B. Rountree, D.K. Lowenthal, M. Schulz, B.R. de Supinski, “Practical performance prediction under dynamic voltage frequency scaling”, IGCC 2011

[103] J. Shalf, S. Dosanjh, and J. Morrison, “Exascale computing technology challenges,” in *International Conference on High Performance Computing for Computational Science*, 2010.

[104] J. Sheaffer, K. Skadron, D. Luebke, “Studying Thermal Management for Graphics-Processor Architectures”, ISPASS 2005.

[105] K. Skadron, T. Abdelzaher, M. R. Stan, "Control-theoretic techniques and thermal-RC modeling for accurate and localized dynamic thermal management," HPCA 2002.

[106] K. Skadron, M. R. Stan, W. Huang, S. Veluswamy, K. Sankaranrayan, and D. Tarjan, "Temperature-aware microarchitecture," ISCA 2003.

[107] K. Skadron, "Hybrid architectural dynamic thermal management," DATE 2004.

[108] E. Sorgard, I. Rickards, "Integrating CPU and GPU, the ARM methodology", ARM developer whitepaper.

[109] The Standard Performance Evaluation Corporation (SPEC). Web resource, http://www.spec.org.

[110] C. Sun, L. Shang, R. P. Dick, "Three-dimensional multiprocessor system-on-chip thermal optimization," Proceedings of International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS) 2007.

[111] A. Tiwari, M. Laurenzano, L. Carrington, and A. Snavely, "Auto-tuning for Energy Usage in Scientific Applications," in *International Conference on Parallel Processing (Euro-Par)*, 2011.

[112] Leslie G. Valiant. A bridging model for parallel computation. Communications. ACM, 1990.

[113] A. Varma, B. Ganesh, M. Sen, S. R. Choudhury, L. Srinivasan, B. L. Jacob. A Control-Theoretic Approach to Dynamic Voltage", International Conference on Compilers, Architectures and Synthesis for Embedded Systems 2003.

[114] Viewdle, http://viewdle.com/products/desktop/index.html

[115] H. Wang, V. Sathish, R. Singh, M. Schulte, N. Kim, "Workload and power budget partitioning for single chip heterogeneous processors," PACT 2012.

[116] S. Williams, A. Waterman, D. Patterson, "Roofline: An Insightful Visual Performance Model for Multicore Architectures", Communications of the ACM, April 2009.

[117] Windows Power Management Overview, http://www.microsoft.com/en-us/download/details.aspx?id=23878.

[118] J. A. Winter, D. Albonesi, "Addressing thermal non-uniformity in SMT workloads," ACM TACO 2008.

[119] D. H. Woo, N. H. Seong, D. Lewis, H.-H. Lee, "An optimized 3D-stacked memory architecture by exploiting excessive, high-density TSV bandwidth," HPCA 2012.

[120] D. H. Woo et al., “COMPASS: a programmable data prefetcher using idle GPU shaders”, ASPLOS 2010.

[121] Q. Wu, P. Juang, M. Martonosi, D. W. Clark, “Formal Online Methods for Voltage/Frequency Control in Multiple Clock Domain Microprocessors”, ASPLOS 2004.

[122] Q. Wu, M. Martonosi, D. Clark, V. Reddi, D. Connors, Y. Wu, J. Lee, D. Brooks, “Dynamic Compiler-Driven Control for Microprocessor Energy and Performance”, IEEE Micro 2006.

[123] Y. Yang et al, “CPU-assisted GPGPU on fused CPU-GPU architectures”, HPCA 2012.

[124] F. Zanini, D. Atienza, G. D. Micheli, "A control theory approach for thermal balancing of MPSoC," ASP-DAC 2009.

[125] http://www.amd.com/us/products/desktop/processors/a-series/Pages/a-series-model-number-comparison.aspx

[126] http://www.xbitlabs.com/news/other/display/20111102214137_AMD_and_Penguin_Build_World_s_First_HPC_Cluster_Based_on_Fusion_APUs.html

[127] http://developer.amd.com/tools-and-sdks/heterogeneous-computing/codexl/

[128] http://www.green500.org

[129] http://www.top500.org

[130] http://www.amd.com/us/products/notebook/apu/ultrathin/pages/ultrathin.aspx#3, AMD A8 4555M.

[131] http://www.techspot.com/news/52003-future-nvidia-volta-gpu-has-stacked-dram-offers-1tb-s-bandwidth.html, March 2013

VITA

Indrani Paul

Indrani Paul was born in Calcultta, India on December 7, 1977 to Mr. Hridesh Ranjan Paul and Mrs. Ratna Paul. In 1996 she began her undergraduate studies in the department of Electronics and Electrical Communication Engineering at Indian Institute of Technology, Kharagpur and she received her Bachelor of Technology (BTech. with Honors.) degree in 2000. After graduation she came to the United States to pursue graduate studies at Georgia Institute of Technology. She earned her Master of Science (MS) degree in Electrical and Computer Engineering in 2002. During MS, she was advised by Prof. Sudhakar Yalamanchili with her thesis topic being "Switch scheduling algorithms for multimedia router". After MS, Indrani joined Dell Inc. in 2002 and moved to Round Rock, Texas. She was a server design engineer at the Enterprise division in Dell architecting power management, power capping and power budgeting solutions for PowerEdge rack and blade servers and general server manageability. While being at Dell, in 2010 Fall, Indrani re-enrolled at Georgia Institute of Technology to start her PhD in ECE under the advice of Prof. Sudhakar Yalamanchili. Almost at the same time, Indrani joined Advanced Micro Devices Inc. in 2011. She has been at AMD since 2011, and has conducted research on heterogeneous processor and system architectures with focus on application analysis, power, performance & thermal monitoring, modeling and management. She is currently a Senior Member of Technical Staff leading the energy utilization efforts at AMD Research for the Department of Energy's Exascale high performance computing project. She has also played various key leadership roles in post-silicon power- performance analysis and optimization of AMD's Opteron line of server and high end client processors. Indrani's PhD education was partly supported by AMD. Indrani has 12 publications, 10 US patents granted and numerous US patents pending in the areas of system architecture, power, performance & thermal management.