Modeling and Runtime Systems for Coordinated Power-Performance Management

Bo Li

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Science & Application

Kirk W. Cameron, Chair
Ali R. Butt
Godmar Back
Dongyoon Lee
Edgar A. León

December 10, 2018
Blacksburg, Virginia

Keywords: Parallel Performance Modeling, Dynamic Voltage and Frequency Scaling, Dynamic Memory Throttling, Dynamic Concurrency Throttling, Shared-Memory Systems

Modeling and Runtime Systems for Coordinated Power-Performance Management

Bo Li

(ABSTRACT)

Emergent systems in high-performance computing (HPC) expect maximal efficiency to achieve the goal of power budget under 20–40 megawatts for 1 exaflop set by the Department of Energy. To optimize efficiency, emergent systems provide multiple power-performance control techniques to throttle different system components and scale of concurrency. In this dissertation, we focus on three throttling techniques: CPU dynamic voltage and frequency scaling (DVFS), dynamic memory throttling (DMT), and dynamic concurrency throttling (DCT). We first conduct an empirical analysis of the performance and energy trade-offs of different architectures under the throttling techniques. We show the impact on performance and energy consumption on Intel x86 systems with accelerators of Intel Xeon Phi and a Nvidia general-purpose graphics processing unit (GPGPU). We show the trade-offs and potentials for improving efficiency. Furthermore, we propose a parallel performance model for coordinating DVFS, DMT, and DCT simultaneously. We present a multivariate linear regression-based approach to approximate the impact of DVFS, DMT, and DCT on performance for performance prediction. Validation using 19 HPC applications/kernels on two architectures (i.e., Intel x86 and IBM BG/Q) shows up to 7% and 17% prediction error correspondingly. Thereafter, we develop the metrics for capturing the performance impact of DVFS, DMT, and DCT. We apply the artificial neural network model to approximate the nonlinear effects on performance impact and present a runtime control strategy accordingly for power capping. Our validation using 37 HPC applications/kernels shows up to a 20% performance improvement under a given power budget compared with the Intel RAPL-based method.

Modeling and Runtime Systems for Coordinated Power-Performance Management

Bo Li

(GENERAL AUDIENCE ABSTRACT)

System efficiency on high-performance computing (HPC) systems is the key to achieving the goal of power budget for exascale supercomputers. Techniques for adjusting the performance of different system components can help accomplish this goal by dynamically controlling system performance according to application behaviors. In this dissertation, we focus on three techniques: adjusting CPU performance, memory performance, and the number of threads for running parallel applications. First, we profile the performance and energy consumption of different HPC applications on both Intel systems with accelerators and IBM BG/Q systems. We explore the trade-offs of performance and energy under these techniques and provide optimization insights. Furthermore, we propose a parallel performance model that can accurately capture the impact of these techniques on performance in terms of job completion time. We present an approximation approach for performance prediction. The approximation has up to 7% and 17% prediction error on Intel x86 and IBM BG/Q systems respectively under 19 HPC applications. Thereafter, we apply the performance model in a runtime system design for improving performance under a given power budget. Our runtime strategy achieves up to 20% performance improvement to the baseline method.

Dedication

Dedicated to my wife Yao, my mother and parents-in-law, and my son Chengyu for their endless support that makes every journey a joy.

Acknowledgments

The completion of this dissertation required great support from the people around me. My family has always been a source of energy throughout my life. I want to thank my wife, Yao, for being supportive and spending tremondous effort taking care of our family; and for the greatest gift, our son Chengyu. Thanks to my mother and parents-in-law since I owe them so much.

I feel so lucky to have had Dr. Kirk W. Cameron as my Ph.D. advisor. He is the person I can go to no matter what question I have. He is not only my Ph.D. advisor, but also a life mentor. I can share my happiness, excitement, frustration, and sadness with him, and every single time he feels it. His encouragement is the most important thing that helped me survive my Ph.D. And after years of encouragement, he even turned me into an optimistic person. Thank you!

Dr. Edgar A León is the person that taught me appreciate meaningful collaborations. He told me he saw me grow over the years of my Ph.D.. But I want to say that this growth would never have come to pass without you. Thank you!

I also want to thank Dr. Ali R. Butt for his guidance during my Ph.D. in my academic progress and career choices. He always encouraged me. Also thanks to my other Ph.D. committee members, Dr. Godmar Back and Dr. Dongyoon Lee for their discussions and helpful suggestions. The whole committee led me all the way to the finish line of my Ph.D. Thank you!

I also want to express my gratitude to other faculty who guided me along the way: Dr. Eli Tilevich, Dr. Danfeng Yao, and Dr. Changhee Jung.

I have been fortunate to work with people from different projects: PowerPack, SeeMore,

Varsys, etc. I want to thank those labmates who helped me develop from a junior student to a senior Ph.D. student and mentor. I also enjoyed every moment working with everyone in these teams. Thank you, my friends!

At last, I want to thank the department of computer science at Virginia Tech: Dr. Calvin J. Ribbens, Sharon, Melanie, Teresa, etc. for being there for all students and for providing help and resources all the time.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Exascale systems demand maximal performance under strict power consumption budgets. The Department of Energy (DOE) sets the goal of power consumption to be 20–40 $megawatts$ for 1 exaflop, which is equal to 50 $Gflops/watt$ in terms of energy efficiency. However, the top systems in the world today are suffering from substantially lower efficiency. Meeting the goal of performance is relatively easy by simply scaling up from today's techniques and putting more building blocks into a giant supercomputer. However, the total power demand would easily reach 200 $megawatts$, which is ten times over the DOE's power goal. Today, the fastest system in the Top500 List [7], Summit, can only achieve approximately 14 $Gflops/watt$, though it can perform at best 143,500 $Tflop/second$. On the other hand, the top system in the Green500 List [2], which is the most energy-efficient supercomputer in the world, can reach at best 17 $Gflops/watt$. However, it trades the performance off by operating only at 1063 $Tflop/second$ as its peak. This is far from the performance requirement of exascale supercomputers.

Several reasons explain why the efficiency of systems is lower than expectations. First of all, the performance of large-scale high-performance systems is limited by the hardware. For example, the slow memory subsystem lowers the utilization of compute resources by making the processors idle to wait for slow data accesses from the main memory. A network of

high-performance systems is another example. While the performance of high-performance systems can be increased in terms of $flop/second$ by using more building blocks, the system-level communication overhead becomes another source of inefficiency due to the slow network connections. Though cutting-edge networking technologies such as InfiniBand [3] can achieve up to hundreds of $GB/second$, this is still too slow compared with processor speed and memory access latency. The major components of system nodes are thus idle and less utilized while data are being transferred across these nodes. Nevertheless, as heterogeneous architectures become more popular and important in high-performance computing (HPC), heterogeneous components such as GPGPUs make the system-level efficiency even lower by introducing more data movement between the host system and accelerators.

On the other hand, HPC applications also bring big challenges to system efficiency from the software perspective. Realistic HPC applications have dynamic computation and memory characteristics while emergent high-performance systems are not smart enough to adopt hardware resources accordingly to maximize system efficiency. For example, LULESH [8], which is an Unstructured Lagrangian Explicit Shock Hydrodynamics application, consists of several iterative code regions defined by scientific application developers. These code regions consume over 90% of LULESH's total execution time [131]. Among the five code regions, two regions are relatively compute intensive while the rest are memory-bound. While the bottleneck dynamically changes with the execution of LULESH, system resources are overprovisioned at different time points. For a memory-intensive code region, the system does not have to provide maximal compute power.

To resolve the efficiency crisis caused by both hardware and software, people have devised different power and performance management techniques, some of which have become commonly available on modern systems. For example, CPU dynamic voltage and frequency scaling (DVFS) [118] is one of the most popular power management techniques and has been

investigated by researchers for years. DVFS is supported on processors of almost all major chip manufacturers for HPC (i.e., Intel, IBM, and AMD). In this thesis, we refer to CPU frequency and power scaling when "DVFS" is mentioned. By adjusting the clock frequency of processors from either the hardware level or the operating system level, users control the performance and power consumption of high-performance systems.

Shared-memory, multicore processors are the basic building block of high-performance systems today. Dynamic concurrency throttling (DCT) is an important power and performance management technique. By dynamically assigning parallel applications to multiple cores on a system, both performance and power may vary accordingly. More cores means more power used by the system. Ideally, this condition leads to better performance.

Similar to DVFS, more and more systems are supporting hardware- and software-based memory subsystem throttling. We can adjust either memory frequency [56] through hardware interfaces or memory bandwidth [96] by inserting idle cycles between memory accesses through the operating system. In this thesis, we generally refer to these actions as dynamic memory throttling (DMT).

Some other effective techniques are also designed and implemented for managing the power and performance of high-performance systems. CPU modulation forces processors to step into lower performance and power modes by sending the idle command. Power gating is another chip-related technique that allows users to put nonutilized logic into deep sleep mode for power-saving purposes without hurting performance. Similarly, the sleep states of the main memory can be changed to control performance and power consumption in real time. Since hard disk access is not always intensive for many HPC applications, spinning speed control is the knob for traditional block I/O devices which are still the major type of storage today in supercomputers. For heterogeneous architectures, people are investigating control techniques on accelerators, such as GPGPU and Intel Xeon Phi, among others. [87,

132, 147, 176, 202]. Nvidia GPGPUs support different CPU and memory frequencies from the hardware perspective. Concurrent kernel execution on GPGPUs can also be managed to improve resource utilization of onboard cores and memory bandwidth. Combining as many techniques as possible can help the systems achieve the best efficiency by making the best use of compute units and power consumption. However, making full use of these techniques simultaneously requires understanding the impact of both hardware interfaces and HPC applications on power and performance. A coordination mechanism for these techniques is necessary to adopt real systems dynamically. Creation of this mechanism is challenging for several reasons:

- Multiple power and performance management techniques produce a large number of configurable operating modes for high-performance systems and applications. It is infeasible to explore the entire configuration space to identify the best combinations of operating modes.

- The combined effects of multiple management techniques have nonlinear performance impact. Some power modes are highly interactive, which makes the performance impact extremely hard to predict using linear modeling approaches (e.g., linear regression).

- The best available performance and power models do not work well for new architectures and new applications and for coordinating emergent systems. New runtime arrangement systems are needed to provide fast insights and accurate predictions.

1.2 Objectives and Research Approach

This thesis explores multiple coordinated power and performance management techniques to achieve the best performance under a given power budget. We focus on three important throttling techniques: DVFS, DCT, and DMT. DVFS and DMT are power management techniques provided by hardware while DCT is provided by software. We have three research objectives and a corresponding approach:

1. to conduct empirical analysis of system measurements of high-performance parallel applications and correlate these measurements with performance and power consumption;

2. to investigate the impact of DVFS, DCT, and DMT on performance (i.e., total execution time) of HPC applications and propose an accurate, insightful analytical performance model for dynamic variations in operating modes;

3. to design and implement a runtime system for parallel applications that leverage novel insights to achieve the best performance under power capping.

First of all, we profile the performance and power consumption of different parallel applications under different system and application states. We consider three throttling techniques that together determine the power-performance state: DVFS, DCT, and DMT. We explore different systems and architectures: Intel x86 system with accelerators such as Intel Xeon Phi and Nvidia GPGPUs, and IBM BG/Q systems from supercomputers at Lawrence Livermore National Laboratory. We extend PowerPack [132] to support fine-grain function-level power profiling. We analyze the performance and power consumption of parallel applications using the software instrumentation provided by PowerPack and PAPI [185].

Second, we explore three throttling techniques: DVFS, DCT, and DMT. By capturing and

quantifying their impact on the execution time of HPC applications, we develop a performance model and approximation approaches to predict the performance under different operating modes. The approximation is application-dependent due to the complicated combined effects of DVFS, DCT, and DMT and variations in application characteristics. Dynamic runtime characteristics make such approximation challenging. For example, some memory-intensive applications can change from memory to compute intensive by reducing CPU clock frequency or increasing memory frequency to some extent. By carefully characterizing the impact of system, architecture, and application, we propose a new parallel performance model for DVFS, DCT, and DMT.

Third, we observed that HPC applications act quite differently when we change operating modes because of their different and dynamic runtime characteristics. None of the existing runtime metrics can completely capture the impact and classify all applications. We develop new metrics that can fully characterize different parallel applications. We further develop a runtime strategy for power capping that coordinates DVFS, DCT, and DMT to identify the configuration with the best performance under a given power budget.

Completion of our research objectives will enable efficient runtime management of DVFS, DMT, and DCT. These efficiency improvements are critical for exascale systems to fulfill the promise of performance and power consumption. As both the scale and the number of the power modes of high-performance systems increase, our methodology can inform the exploration of the coordination of three or more power management techniques.

1.3 Research Contributions

In this subsection, we summarize the major contributions made upon completion of this dissertation. We present and discuss each contribution in detail in subsequent chapters

correspondingly.

1.3.1 Empirical Analysis on Performance and Energy in HPC

Trade-off of Performance and Energy on Intel Xeon Phi

Though numerous studies of GPU power performance have been conducted [50, 99, 177, 195], detailed studies of the Xeon Phi are few [24, 176]. The most closely related work to ours describes a methodology for instruction-level modeling of power on the Xeon Phi [176]. While useful for architectural trade-off analysis and runtime system steering, the accuracy and portability of such approaches is less suitable for system-to-system comparisons. In particular, the data provided focus on the Xeon Phi exclusively and derives power information from a total system measurement. This information is inherently less accurate than direct measurement, and the focus on the Xeon Phi at the instruction level limits portability and comparative studies.

In contrast to past approaches of power-performance studies of GPGPUs [50, 99, 177, 195] and Intel Xeon Phi [24, 176], we focus on collecting and analyzing power and performance information at the system level. We collect direct power measurement data and separate the data by component: CPU, memory, board, NIC, etc. We then align these data with application functions for a wide range of HPC applications. For this work, we extended our capabilities to separate Xeon Phi accelerator power from total system power. This was particularly challenging since accelerator cards often share power rails with the host CPU and other devices. We solved this problem by meticulously splicing the wiring of riser cards and measuring their power in-line between the accelerator and the main board. Our methodology is portable, and though designed for the Xeon Phi experiments, we additionally tested and compared results on an NVIDIA Tesla GPU for a wide range of HPC applications. Our

systemic view also allows us to compare and contrast the energy efficiency of off-loading from our Intel Sandy Bridge host processor to the Xeon Phi.

This work makes the following contributions:

- We develop a software and hardware infrastructure to obtain direct power measurement data from accelerators isolated from CPU, disk, memory, NIC, etc. We extend our software API to support tracing and aligning accelerator performance information with raw power data to enable fine-grain profiling.

- The infrastructure we developed can help application developers analyze their codes from the aspects of performance, power, and energy consumption. Correlation of devices to code enables pinpointing of performance and energy bottlenecks.

- We demonstrate the usefulness of our extended framework to profile and analyze the power and performance of HPC applications on Xeon Phi–based systems. We use this enhanced toolkit to show exactly where power goes when accelerators are used to execute common HPC applications. Additionally, we analyze host Sandy Bridge performance versus the Xeon Phi.

- We perform a detailed comparison of the Xeon Phi against an NVIDIA GPU using our extended framework. We compare and contrast the Xeon Phi and Tesla GPU in detail and explain the key differences in their power and performance profiles. Our results indicate efficiency depends substantially on the characteristics of the applications running on the system.

Memory Throttling on IBM BG/Q

Power and energy efficiency are major concerns in future supercomputing systems. We expect that HPC applications will be constrained to operate under a power budget, and thus, achieving the expected levels of application performance will be challenging. Furthermore, because of the complexity of these applications, the computational requirements within a given code may vary on a per-physics and per-phase basis dynamically and across different problem sizes and input sets. Given this environment, we need a better understanding of the computational requirements of applications so that we can distribute and reallocate power selectively to those system components that an application really needs.

In this work, we identify and analyze opportunities for distributing power among system components on a supercomputer system to improve the performance of a representative kernel of explicit hydrodynamics codes at the U.S. Department of Energy (LULESH). Using an energy-efficient supercomputer architecture, IBM Blue Gene/Q (BG/Q), we throttle the memory system on a per-region basis according to a linear regression model based on performance counters. We envision that in future machines, advanced runtime systems will shift power from the memory system to the CPU, for example, in regions with high locality and low demand for memory bandwidth. Although BG/Q does not allow redirecting power from one subsystem to another, our objective is to identify opportunities where this may be possible. We demonstrate that even with a simple linear regression model, we can save up to 20% of dynamic power with a marginal effect on performance.

1.3.2 Parallel Performance Modeling

In this work, we present the compute-overlap-stall (COS) model of parallel performance for dynamic variations in processor speed, memory speed, and thread concurrency. To the best

of our knowledge, this is the first model to accurately capture the simultaneous combined effects of these three operating modes.

The COS model is based on a simple observation. Past models of operating mode performance tend to combine the overlap of compute and memory performance into either compute time or memory stall time. However, we have observed that the behavior of overlap when these operating modes change is so complex that it must be modeled independently of these other times. This observation leads to the formulation of a COS model where each term can be modeled independently of the others.

In addition to presenting the COS model, we demonstrate how to capture these important (and independent) parameters on both Intel servers and the IBM BG/Q system. We also show how the COS model can be used to classify the best available models. We validate our modeling efforts on 19 HPC kernels and perform extensive sensitivity analyses to identify weaknesses. Our COS model has more functionality than previously available, and the accuracy is as good as or better than best available operating mode models with prediction errors as low as 7% on Intel systems and 17% on the IBM BG/Q system.

1.3.3 Runtime System Design for Maximizing Performance Under Power Cap

The efficiency of future high-performance systems will be the key to surviving tight power consumption requirements and various application characteristics. Emergent systems will have a large number of configurable operating modes to adapt to different application behaviors and achieve the best performance under a given power budget. The behavior of different high-performance parallel applications significantly varies, as the optimal operating modes are highly application-dependent.

In this work, we first characterize the performance impact of DVFS, DMT, and DCT on various high-performance parallel applications from different domains (scientific simulation, data mining, linear algebra, artificial intelligence, graph algorithm, etc.). We identify and present metrics for quantitatively measuring the separate impacts of configurations for multiple parallel applications. Considering the nonlinear effects caused by numerous operating modes and various parallel applications, we use a nonlinear machine learning technique, artificial neural networks (ANNs), to approximate the relationship between the observable parameters and the performance impact. We further propose a runtime control strategy called "greedy-walk" to adapt our ANN-based performance prediction to runtime steering of configurations for maximal performance under a given power budget. We validate our ANN-based approximation using 38 high-performance applications/ kernels. By comparing our greedy-walk strategy with existing RAPL-based power capping solutions under different power budget levels, we see up to a 20% performance improvement.

1.4 Organization of the Dissertation

The rest of the dissertation is organized as follows:

In Chapter 3, we introduce our research of power measurement and analysis on both Intel x86 and IBM BG/Q architectures. We present the function-level power profiling tool, PowerPack, for heterogeneous systems with accelerators. We also explore the performance and energy trade-offs on both architectures considering scalability and system performance throttling.

In Chapter 4, we analyze the performance impact of three throttling techniques: CPU dynamic voltage and frequency scaling (DVFS), dynamic memory throttling (DMT), and dynamic concurrency throttling (DCT). We identify the root cause of failure for all the existing performance models and propose a new parallel performance model, the "*COS*"

model. We present a linear regression-based performance prediction approach using the COS model and validate the accuracy for 19 HPC applications/kernels on both Intel x86 and IBM BG/Q systems. The details can be found in this chapter.

In Chapter 5, we discuss the demands for power capping in HPC and the limitations of optimizing performance. Based on the COS model, we applied a nonlinear machine learning technique (i.e., artificial neural networks) for low-overhead performance prediction. We propose a greedy-walk-based runtime control strategy for maximizing performance under power budget by leveraging DVFS, DMT, and DCT.

In Chapter 6, we summarize the research completed in this dissertation and discuss future work.

Chapter 2

Review of Literature

This thesis focuses on improving the energy efficiency of parallel applications on multi-core systems to meet the goals of the exascale era. In this section, we first review power profiling techniques on various architectures. Next, we review power and performance management techniques and performance models on modern systems. Lastly, we review runtime system designs which adapt different power management techniques to improve system efficiency and performance under power capping.

2.1 Power Measurement and Analysis on Heterogeneous Systems

Power measurement on heterogeneous system informs our in-depth analysis of energy efficiency bottlenecks.

Hardware instrumentation monitors the power consumption of different system components out-of-band or using data gathering independent of the system. Ge et. al. [86] presented the out-of-band power measurement tool, PowerPack. It measures the power consumption of each power rail in an ATX power supply to profile the power of system components. The complicated infrastructure of PowerPack makes it hard to be deployed in all the nodes of a large cluster of systems. Laros et. al. [124] designed a commodity power measurement

board that instruments the ATX power supply but is easier to setup. Around the same time, Bedard et. al. [28] proposed a similar out-of-band power profiling tool called "Powermon". However, none of these tools supports the power measurement and analysis of accelerators on heterogeneous systems. Lack of such tools makes it challenging to analyze the tradeoffs between performance and energy consumption on heterogeneous platforms.

Intel released Xeon Phi as their version of a system accelerator for HPC environments and claimed that many applications require small or even no changes to run on Xeon. Research has focused on exploring alternative algorithms or various optimization techniques to speedup parallel applications on Xeon Phi [59, 80, 101, 157, 161, 193, 203]. Park et al. [157] proposed an algorithm for synthetic aperture radar (SAR) via backprojection to achieve low-cost data movement and better parallelism on MIC. Williams et al. [193] examined a variety of optimization techniques, including communication-aggregation, threaded wavefront-based DRAM communication-avoiding, dynamic threading decisions, SIMDization, and fusion of operators, to optimize geometric multigrid on MIC. Pennycook et al. [161] explored how SIMD benefits molecular dynamics (MD) benchmark in terms of performance. Deisher et al. [59] accelerated LU factorization via a proposed SGEMM implementation, which takes a full advantage of multiple levels of memory hierarchy on MIC. Garea and Hoefler [80] proposed a performance model for cache-coherent architecture, and used the model to develop optimized algorithms for parallel data exchanges on Xeon Phi.

Early studies have focused on the performance analysis of Xeon Phi and showed that it can deliver good performance for several parallel application kernels and algorithms, including the BFS graph algorithm, sparse matrix multiplication, CG, and LU Decomposition kernels [51, 149, 171, 172]. Saule et al. [171, 172] investigated the Xeon Phi performance and scalability of the BFS graph algorithm [171], as well as the performance of SpMV [172], showing that the Xeon Phi's sparse kernel performance is better than that on a dual core

Intel Xeon X5680 CPU or a NVIDIA Tesla c2050 GPU. Cramer et al. [51] evaluated the speedup, throughput, and scalability of the OpenMP version of the CG kernel on Xeon Phi and compared the results with a 128-core SMP machine based on the Bull Coherence Switch (BCS) technology. Misra et al. [149] conducted the performance comparison analysis between Xeon Phi and CPU with two Rodinia kernels "LU Decomposition" and "HotSpot", showing that Xeon Phi outperforms CPU on LU decomposition but loses to CPU on HotSpot.

Xeon Phi power and performance studies are limited in the literature. Many references that mention power consumption are Intel manuals [5] or press releases [74, 114]. Power information can be gleaned from The Green500 List [44] which ranked Beacon, a Xeon Phi based system. However, the list entry was based upon total system measurements and data sheets [48]. Shao et al. [176] proposed an instruction-level energy consumption model for the Intel Xeon Phi. This study focused on architectural level analysis and the Xeon Phi in isolation. In contrast, our work attempts to capture power and performance information for the Xeon Phi in the context of system building blocks that form large scale systems for HPC applications.

2.2 Power Management Techniques

Energy efficiency is a growing challenge for supercomputers pursuing high computation throughput, and embedded platforms with limited battery capacity. Power management techniques enable applications on these systems to achieve better energy efficiency. In this section, we will divide the previous work of investigating power management techniques into two groups and review each separately: power management of hardware and software.

2.2.1 Power Management of Hardware

Hardware designers expose interfaces to users to control the power consumption of different system components which are major power consumers. There are throttling techniques for CPU, main memory, hard disk, and accelerators such as GPGPU and Intel Xeon Phi, which trade the performance of different components for power benefits.

Multi-core Processor Throttling

Multi-core processors are widely used in high-end servers as well as embedded platforms such as mobile phones. To throttle the performance of processors, throttling techniques include, but are not limited to: Dynamic Voltage and Frequency Scaling (DVFS), Per-Core Power Gating (PCPG), and clock frequency modulation. Similarly, we can adjust the performance of the memory subsystem by varying memory bus frequency or memory bandwidth, or even forcing it into sleep state. Depending on the type of hard disks, changing the spinning speed of HDDs or adapting heterogeneous hard disks can affect both the power consumption and performance of I/O. Accelerators, such as GPGPU, also support throttling techniques of on-board CPU and memory DVFS.

Dynamic Voltage and Frequency Scaling (DVFS) is one of the most common throttling knobs available on modern processors. There are numerous research projects investigating the benefits of applying DVFS to processors. Kim et. al. [118] compared the impact of using on-chip and off-chip regulators on DVFS transition latency. Isci et. al. [108] explored per-core DVFS to meet core-level power limits, and showed power savings. These findings motivates further investigation of energy efficiency optimization by using multiple clock domain (MCD) processors [174, 198, 199]. Semeraro et. al. [174] designed a four on-chip clock domain processor and showed that its energy efficiency outperformed the scheme of

applying DVFS on a single clock domain. The four clock domains are frontend, integer units, floating point units, and load-store units. Wu et. al. [198] developed a feedback control system which applies DVFS to the multiple domain processor to achieve high efficiency. They [199] also characterized the impact of reaction time of DVFS to the efficiency.

Since DVFS is commonly available and effective for power management, it is applied in different areas, such as HPC, cloud computing, and embedded systems (e.g. mobile, IoT).

Previous research into adapting CPU DVFS for HPC covers systems scaling from single-core and multi-core up to a cluster of multiple multi-core nodes. Choi et. al. [43] divided the workload of applications into on-chip and off-chip portions. By throttling the off-chip portion of workload using DVFS, energy savings can reach 80% for single-threaded memory-bound workloads. Dhiman and Rosing analyze [65] the impact of DVFS on energy consumption on a system which hosts multiple single-threaded tasks and designed [64] a DVFS based runtime system to reduce energy consumption by deciding the best clock frequency at any point of time during execution.

On multi-core architecture, application developers have explored steering DVFS to achieve better system efficiency. Alonso et. al. [18] compared the impact of two different DVFS control schemes on dense linear algebra applications for energy consumption and execution time on a multi-core system. This showed promising results in energy savings by adapting DVFS for negligible performance loss. In a follow-up work, they presented a DVFS-aware runtime system and reduced the energy consumption of CPU-bounded LU factorizaton by 5% with a minor increase in execution time [17]. Lu et. al. [140] proposed a new DVFS mechanism leveraging compile-time information for various HPC applications on a multi-core system. This achieved higher energy-delay product (EDP) than existing history- or profile-based DVFS approaches.

Most HPC applications focus on large scale systems with thousands of cores spread across many servers connected by a high-speed network with bandwidth at the level of hundreds of Gigabytes per second. Previous work has shown that there are great opportunities to optimize energy efficiency using DVFS on large scale HPC systems. Ge et. al. proposed "CPU Miser", a DVFS scheduling system for Message Passing Interface (MPI) applications, which selects the best CPU clock frequency by predicting the workload on a timestep basis. This helps save up to 20% energy for NPB benchmarks, which performs better than the default DVFS governor in the linux kernel. Rountree et. al. [169] designed a runtime solution called "Adagio" to dynamically change the processor frequency for each MPI task according to the estimated slack time. Their solution outperforms previous solutions by yielding more energy savings and less performance loss. Recently, Peraza et. al. [162] proposed a DVFS-based framework to adjust CPU frequency based on detected characteristics. If the tasks of MPI applications are imbalanced, DVFS is applied to adjust the execution time of each individual task to reduce the slack time and improve system efficiency. For the MPI applications detected with balanced tasks, DVFS is adapted to reduce energy consumption guided by a power model and timing information.

Much existing research into runtime mechanisms assume that iterative MPI applications have repeatable behaviors for each iteration. Venkatesh et. al. [189] observed that MPI applications could have dynamic communication behavior per iteration. They proposed a runtime solution to dynamically detect and estimate the slack time, which guides DVFS decisions. Guillen et. al. [93] proposed a DVFS-based auto-tuning solution for HPC applications (OpenMP and MPI) to optimize different objective settings including energy, cost of ownership, energy delay product, and power capping. The auto-tuning relies on a derived model of power, time, and energy consumption under CPU DVFS. Bhalachandra et. al. [32] presented a per-core DVFS solution for MPI applications according to duty cycle based

policy. This shows a 20% improvement in energy efficiency with less than 1% performance degradation on average across six standard MPI benchmarks and a real world application. Bailey et. al. [22] formulated the DVFS based scheduling of MPI/OpenMP tasks as a linear programming problem with a varying number of OpenMP threads, and showed the theoretical upper bound of performance under different power budgets. This provides insights into how high the energy efficiency could be on supercomputers.

Data centers are paying more attention to power management techniques such as DVFS as their electricity cost becomes large. As one of the most common cloud computing frameworks, MapReduce applications can use DVFS to reduce energy consumption [105] as well as improve energy efficiency [194]. Wu et. al. [196] proposed a scheduling algorithm based on DVFS to increase the utilization of resources which can effectively reduce the total energy consumption by 5%-25%. There are other research projects exploiting DVFS to improve energy efficiency of data centers in the cloud [20, 25, 73, 167].

Mobile platforms and embedded systems are powered by batteries with limited capacity. Power consumption and energy efficiency running onboard applications can significantly affect the duration of the battery. As modern mobile application processors start to support power aware features like DVFS, numerous previous works [9, 12, 19, 29, 37, 119, 156, 163, 175] have focused on improving efficiency by adapting DVFS in a static or dynamic manner.

Per-Core Power Gating (PCPG) is another CPU throttling technique available on most modern high end processors. This switches the state of the processor between idle and active according to the computation workload. Many processors design power gating in the form of "sleep mode". Idle mode conserves a significant amount of power while losing performance. Hu et. al. [102] proposed the idea of reducing leakage power from the hardware design point of view through power gating. They evaluated three different approaches to control power gating and showed they evoke sleep cycles for power savings during execution with little

performance loss. Jacobson et. al. [109] investigated power gating on real processors, IBM Power 4/5, and explored a strategy for identifying the necessary scenarios for power gating in real workload executions. Lee and Kim [125] characterized the impact of incorporating PCPG with DVFS on the performance of multi-core processors. They further showed improved performance under power and thermal limits. Lungu et. al. [143] also investigated the impact of the misprediction of power gating from the perspective of security. In order to prevent repetitive patterns from being exposed, they presented a guard mechanism for preventing power overruns.

CPU Clock Modulation is another practical method for throttling CPU performance and power consumption. Zhang et. al. [205] explored the interface of clock modulation on Intel multi-core processors and its impact on system efficiency. They observed that clock modulation could cause performance loss while overall system efficiency could be maintained by resolving resource bottlenecks using cache partitioning. Wang et. al. [192] compared the characteristics of DVFS and clock modulation and observed much lower transition latency using clock modulation. They further evaluated the energy savings and energy delay product of using clock modulation combined with concurrency throttling on several OpenMP applications. Schöne et. al. [173] conducted similar comparison between software controlled clock modulation with DVFS but on multiple generations of Intel processors. They showed a case of optimizing inter-process synchronization on the Intel Haswell processor using model-based throttling decisions. Cicotti et. al. [45] investigated the impact of DVFS and clock modulation on performance, power, energy consumption, and energy delay product. They proposed an online solution for the dynamic adjustment of CPU speed by DVFS and clock modulation, which shows power reduction and improvement of energy efficiency and EDP at different performance degradation levels.

Memory Throttling

Memory is another major source of power consumption in HPC systems as many HPC applications require intensive interactions between the memory and the processing unit (e.g. CPU, GPGPU, etc.). The amount of power drawn by the main memory (e.g. DRAM) is affected by several factors: the total amount of capacity usage, the frequency of interactions with the memory, and its operational clock frequency. Correspondingly, there are several possible power management methods: memory frequency scaling and memory bandwidth throttling. In addition, as with the CPU, we can also change the sleep states of main memory to adjust performance and power consumption. In general, we call these techniques: dynamic memory throttling (DMT).

Memory Frequency Scaling relies on the fact that the memory bus frequency of modern systems is adjustable. David et. al. [56] confirmed that memory consumes a significant amount of power on a typical server for data centers running SPEC CPU2006. They explored the opportunity of saving power by dynamically changing memory frequency through timing registers. A 0.17% slowdown can achieve 10.4% memory power reduction. Deng et. al. [60] analyzed the breakdown of memory power consumption and the impact of memory frequency scaling on each of them. They proposed a performance model and power model under memory frequency scaling to guide the online selection of frequency modes. The validation using single threaded applications showed significant energy reduction over other memory management strategies. They [61] further extended their model to support multiple memory controllers and proposed a heuristic algorithm for frequency scaling decision making. To achieve higher energy reduction, Deng et. al. [62] presented a new performance model to incorporate CPU DVFS and memory frequency scaling. The validation shows the runtime system achieves similar energy savings as the offline solution for a given performance bound.

Memory Bandwidth Throttling is another technique for throttling the performance of memory subsystems. Hanson and Rajamani [96] introduced and discussed the feature of memory bandwidth throttling on IBM Power6 systems. Users can throttle the memory bandwidth by inserting a certain number of idle cycles between memory accesses. The impact on performance and power consumption depends on both throttling level and application characteristics. The sensitivity analysis of memory throttling on performance and power for applications from compute bound to memory intensive provide insights for runtime system design. Li and León [133] explored the opportunity for power reduction by applying memory throttling on BG/Q according to the runtime characteristics of applications. They were able to achieve great energy reduction with very small performance loss using a hydrodynamics application.

Memory Sleep States are controllable and indicate whether the memory is active or not. Memory subsystems can conserve power by shifting to the idle state. Fan et. al. [72] discussed changing DRAM power states using the memory controller to improve energy efficiency. According to their idle time model, immediate transition to low power state is a better option than using a predictive approach for idle time. Huang et. al. [103] suggested that it is not always accurate and efficient to track the idle time of DRAM. Instead, they proposed a page migration based approach to group memory into hot rank and cold rank. Different memory sleep states are applied to different ranks accordingly for power reduction. Wu et. al. [141, 197] built on previous approaches to predict idle time. They tracked the memory access patterns and grouped the pages with similar locality into the same rank. They also estimated the delay for changing power modes to improve the accuracy of control. The results showed significant improvement on energy-delay product and energy reduction. The idea of page migration for dynamic power model control led to a new design of memory hierarchy. Sudan et. al. [181] proposed the "two-tier" memory hierarchy: the hot tier and

cold tier. The power mode-based management strategy was applied accordingly to improve energy efficiency. Diniz et. al. [66] provided another reason for capping memory power consumption when peak power consumption is not welcome due to effects related to packaging, cooling, etc. They proposed a runtime solution to adjust the power states of memory dynamically to cap the memory power. Hur and Lin [104] proposed a comprehensive approach which incorporates the low power mode of DRAM with idle-cycle based bandwidth throttling. This leads to significant improvement in energy efficiency for several HPC applications on the IBM Power5+ system. Zheng and Zhu [206] conducted a comprehensive investigation into different DRAM configurations including power mode, power breakdown, and different generations of DRAM. This provides insights into the relationship between applications and power performance tradeoffs under different memory configurations.

HardDisk Throttling

The main throttling technique for traditional harddisk (HDD) is to spin disk speed up and down (i.e. spin speed throttling). Carrera et. al. [38] explored different solutions for conserving disk power for high performance clusters. The results show that multi-speed disks can save energy up to 23% without any performance loss for the chosen applications.. Gurumurthi et. al. [95] presented a new approach called "DRPM" to dynamically modulate disk speed. Simulation results showed great energy reduction with little performance loss. This technique avoids the latency issue of putting disks into sleep mode and waking up when needed.

Accelerator Throttling

Accelerators, such as GPGPU and Intel Xeon Phi, play a growing role in HPC. They improve the performance of scientific applications by offloading simple and repetitive calculations to massive light weight cores in parallel. However, due to differences in application characteristics, performance and energy efficiency vary greatly for applications running on accelerators. The most common throttling knobs are on-board core and memory frequency scaling.

On-board Frequency Scaling refers to the multiple frequencies of GPU cores and memory on board the accelerator. Jiao et. al. [113] characterized the impact of GPU frequency scaling (for both core and memory) on the power and performance for applications with different computation intensity. They provided insights into how to improve energy efficiency accordingly. Ge et. al. [87] carried out a similar investigation but on a relatively new GPU architecture: Kepler. They highlighted several differences between CPU frequency scaling and GPU frequency scaling. Mei et. al. [147] also validated the opportunity of power reduction by adapting GPU DVFS with 37 benchmark applications. Jiao et. al. [111] presented a solution of combining GPU DVFS with concurrent kernel execution for improving energy efficiency. Nath and Tullsen [151] proposed an analytical performance model for GPU DVFS which accounts for the overlap between computation and memory accesses as well as store-related stalls. The model steers runtime prediction to improve energy efficiency. Lee et. al. [126] combined on-board DVFS with GPU core scaling to achieve the best performance under power budget.

2.2.2 Power Management of Software

Given the known system architecture, power management of software leverages information of both hardware, operating systems, and applications characteristics. It includes, but is not

limited to: dynamic concurrency throttling (DCT), data allocation, and concurrent kernel execution on accelerators.

Dynamic Concurrency Throttling (DCT)

Dynamic concurrency throttling is a well established approach for improving the energy efficiency of multi-threaded applications in HPC. By dynamically changing the concurrency level in number of used cores, performance and power consumption change with resource demand.

Curtis-Maury et. al. [53] discussed the performance and power impact of concurrency throttling of multi-threaded applications on a multi-core system. They proposed a hardware counter based runtime solution to adapt thread configurations according to application execution. The runtime framework relies on an IPC (instructions per cycle) based performance model to select the optimal configuration for the best energy efficiency. The concept of CPU packing is very similar to DCT as it selects the appropriate number of cores to use instead of choosing all cores for parallel application. Freeh et. al. [77] investigated the combined effect of such currency throttling with CPU DVFS. On their testbed with 4 AMD cores on two sockets, efficiency is highly dependent on the applications. Suleman et. al. [182] discussed two major bottlenecks as the concurrency increases for different applications. Synchronization intensive applications can have their performance degraded by increasing concurrency. Bandwidth intensive applications are more power sensitive to the concurrency value. Once bandwidth becomes the bottleneck, additional concurrency cannot improve performance. They proposed the concept of feedback driven threading and implemented two different strategies: synchronization aware and bandwidth aware threading. The optimal concurrency level is determined at runtime based on sampling applications and monitoring resource utilization. Validation using 12 multi-threaded applications shows improved execu-

tion time and lower power consumption. Li et. al. [135] characterized the imbalance of MPI tasks and explored opportunities for applying DCT and DVFS to OpenMP threads within each MPI task. Their goal was to improve energy efficiency for hybrid MPI/OpenMP applications with load imbalance. They proposed an empirical performance model to capture the impact of DCT and DVFS on the total time of each OpenMP thread and implemented a runtime system for steering HPC applications. Validation using popular HPC applications shows up to 13.8% energy reduction. Similarly, Lively et. al. [139] identified the most critical metrics for analyzing the impact of DCT and DVFS. They proposed a power model based on these and yielded a prediction error of less than 3%. Bailey et. al. [22] also focused on the hybrid MPI/OpenMP applications and proposed a solution for capturing the upper bound of performance under limit of power budget. They formulated the problem as a constrained optimization problem and developed a linear programming algorithm to solve it. They showed that power constrained performance could be improved by up to 41.1% for HPC applications.

More previous work on incorporating DCT with other power management techniques will be discussed in a later section.

Concurrent Kernel Execution on Accelerators

The idea of coscheduling execution was first explored on GPGPU [92] for the purpose of increasing throughput. The theory is that not all applications can fully utilize the resources of the GPGPU, and by concurrently scheduling multiple applications throughput and utilization may increase.

Gregg et. al. [91] found it non-trivial to identify the best resource partitioning solution due to complicated characteristics of application kernels. They characterized the interaction

of applications and proposed scheduling two OpenCL codes simultaneously. They used the scheduler to investigate interactions between kernels and their impact on throughput.

GPGPUs consume a lot of power. Wang et. al [191] explored different kernel coscheduling scenarios using CUDA applications. They also proposed energy aware scheduling algorithms to reduce energy consumption.

Adriaens et. al. [10] also found concurrent kernel execution is effective for power constrained GPGPUs. They observed resource underutilization for many GPGPU applications and evaluated several heuristics algorithms for partitioning GPU stream multiprocessors. They were able to achieve better speedup when multiple applications require coscheduling. Similar optimization approaches have been explored by several other groups [136, 155, 207].

Jiao et. al. [112] took advantage of the newly implemented concurrent kernel scheduling interface on up-to-date GPGPUs. They proposed a power-performance model for concurrent kernel execution and the DVFS of core and memory on board. Their evaluation of configuration prediction can improve energy efficiency by up to 34.5% compared with the most energy efficient sequential execution.

2.3 Performance Modeling and Runtime Design for HPC

Performance and power modeling is a critical component for guiding applications and systems to achieve the best energy efficiency and performance. In this section, we will discuss existing models of performance and power consumption for single device power management. Then we will review those models which manage multiple devices. Corresponding runtime solutions enabled by performance models will also be reviewed.

2.3.1 Single Power Management Method

The ultimate goal of exploring power management techniques is to assist runtime steering to improve energy efficiency as well as performance. Performance and power modeling for different power management techniques is necessary before using in runtime system design. Previous research work has proposed different modeling approaches: analytical, empirical, statistical, or a hybrid of these. Performance models have been widely used for evaluating and optimizing systems. Some successful models include: PRAM [11, 49, 75], BSP [27, 57, 187], LogP [52] and its derivatives [16, 34, 35, 76, 98, 150]. However, in this review I will only focus on performance models for various management techniques.

Dynamic Voltage and Frequency Scaling (DVFS) has received much attention for power modeling and management by researchers in different areas (HPC, cloud computing, embedded systems, etc.). Rountree et. al. [168], Eyerman and Eeckhout [71], and Keramidas et. al. [117] proposed analogous analytical performance models for CPU DVFS by decomposing the total execution time of single-threaded applications into CPU time and memory time. Memory time starts from the very first main memory load request and ends with the completion of the last main memory load request. This is the so-called "leading loads model". The impact of DVFS is estimated as if affects CPU time while memory time is assumed unaffected no matter the clock frequency the CPU. They validated the leading loads model via simulation since some necessary information was not exposed or supported on real systems at that time. Validation using applications such as SPEC CPU2006 and NASA Parallel Benchmarks showed performance prediction accuracy.

Meanwhile, Keramidas et. al. [117] suggested another analytical performance model called the "stall-based model". This divides the total execution time into CPU time and stall time. Stall time is defined as when the CPU is completely idle and waiting for the data to be filled

by cache or memory. CPU time is modeled as proportional to the clock frequency while stall time is assumed constant throughout CPU DVFS. Both the leading loads model and stall-based model showed prediction accuracy depends on the application.

Choi et. al. [42] presented another performance modeling method under CPU DVFS which is similar to the idea of the stall-based model. They decomposed the computation workload into on-chip and off-chip. The change of CPU clock frequency mainly affects the on-chip portion of the workload in a proportional way. They validated performance prediction accuracy across several common single threaded benchmarks.

Miftakhutdinov et. al. [148] advanced the leading loads model by considering varying memory access latency on systems. Building on the leading loads model, they proposed a strategy to identify memory loads on the critical path of application execution. They also considered the impact of prefetching on memory bandwidth. They validated prediction accuracy using single threaded SPEC 2006 benchmarks and showed performance improvement over the leading loads model and stall based model.

March et. al. [146] proposed a Proc-Mem performance model derived from the leading loads model. This models the performance under DVFS on an in-order single core processor for embedded systems. The accuracy of Proc-Mem outperformed the model CMAT which assumes constant latency of all cache misses.

Nishtala et. al [152] proposed a linear regression based statistical performance model of DVFS for data centers. Across a large configuration space, a regression based model can be quickly built and applied for different applications. The prediction error of single core scenarios is up to 40%.

Akram et. al. [14] presented another leading loads derivative model, DEP+BURST, to predict the performance of managed multi-threaded applications under DVFS. They proposed

a strategy to identify the critical threads throughout the execution. By applying the theory of the leading loads model to each critical thread, they were able to achieve a prediction error of 6% on average.

Ge et. al. [82] derived a new performance model for HPC applications following the idea of workload decomposition. They further designed a runtime system to adjust CPU clock frequency according to system level runtime characteristics. Energy savings were up to 20% for the NASA Parallel Benchmarks.

Su et. al. [180] implemented the leading loads model on a real system for the first time. They leveraged the new hardware counters available on AMD processors to capture the leading loads for all the stall periods during application execution. The prediction error is 2.7% on average using a large set of parallel applications. The runtime solution requires no offline model training and reduced overhead for hardware counter monitoring.

Su et. al. [179] proposed an online algorithm to explore DVFS space for performance, power, and energy consumption. They also presented a user case of power capping by adapting this online framework.

Cameron et. al. [36] investigated the impact of DVFS and other system settings on performance variance. They developed machine learning models to predict performance variance based on the configurations.

Dynamic Concurrency Throttling (DCT) is an important technique for controlling performance and power consumption. Performance modeling of DCT is quite challenging due to the diverse characteristics of parallel applications. Most existing work on performance modeling for DCT uses empirical-based or statistical-based models.

Curtis-Maury et. al. [53] proposed a regression based model to predict IPC (instructions per cycle) under different settings of concurrency. Using critical component analysis, they

selected several hardware counters as the variables in their regression equation. They implemented a runtime system accordingly to identify configurations optimized for energy efficiency. Experimental validation showed this method outperforms all existing online solutions.

Sarood et. al. [170] explored power-capped performance optimization using DCT and power allocation on CPU and memory. They proposed an interpolation based strategy for the allocation of compute nodes and power budget. The performance speedup is 2.2x that of the global power budget strategy.

Ge et. al. [81] proposed a speed-up model similar to Amdahl's law under power constraints. The power-aware speed-up model quantified the performance under different concurrency levels by isolating parallel overhead. The prediction of performance and energy delay products under different frequency settings showed error of 7%.

Dynamic Memory Throttling (DMT) includes memory frequency scaling, memory bandwidth throttling, and memory power state throttling.

Deng et. al. [60] derived an empirical performance model from the stall-based model for memory frequency scaling on single core processors. They investigated and approximated the impact of memory frequency scaling on the memory queue. By incorporating the power model, they designed the mechanisms to dynamically select the optimal frequency at runtime.

Elangovan et. al. [67] studied the potential of memory frequency scaling at the granularity of the memory channel. They characterized memory channel access patterns of applications and determined the best frequency settings accordingly. Validation of the proposed runtime memory frequency scaling algorithm was conducted via hybrid simulation and real hardware.

Gulur et. al. [94] proposed an analytical performance model for the memory subsystem. It captures the impact of design choices (e.g. clock frequency) on latency and bandwidth using

queueing theory. Several metrics important to the queueing model and their impact were discussed.

Jang and Park [110] implemented memory frequency scaling at the OS level for the first time. They proposed an optimization method which improves energy efficiency by up to 18%.

Frequency Scaling on GPGPU is challenging due to the large overhead of computation and memory accesses. Nath and Tullsen [151] proposed an analytical performance model for GPGPU DVFS considering overhead and the impact of stall-related operations. The prediction accuracy outperforms previous models. They developed a runtime system based on the model and realized a 10.7% improvement in energy delay product.

Kim et. al. [79] presented an energy-centric DVFS solution that evaluates the level of resource contention and adjusts the clock frequency based accordingly. This, compared with the default "ondemand" DVFS scheduler, achieved high energy efficiency.

Power/energy models are typically proposed with performance models in order to design runtime systems targeting energy efficiency. Rossi et. al. [166] recently presented a regression based power model that captures the impact of DVFS policies (e.g. ondemand, performance, etc.), CPU utilization, and total time on the power consumption. David et. al. [55] proposed a memory power model for RAPL based on memory access characteristics. They applied the model to the design of a power capping runtime system.

2.3.2 Coordinating Multiple Power Management Methods

In this section, we will review performance modeling and runtime systems that consider coordinating multiple power management methods. Though a single power knob can improve the energy efficiency, multiple knobs are supposed to help achieve better results. In this

section, we categorize the discussion into groups by knobs involved.

DVFS and DMT: Deng et. al. [62] derived the performance model from the stall based model to capture the impact of DVFS and memory frequency scaling on the CPI (cycles per instruction). They used the linear regression based approach to correlate several hardware events to the performance and validated them by simulation. Lu et. al. [142] proposed a performance model under DVFS and memory power state throttling following the leading loads model. Sundriyal and Sosonkina [183] proposed a performance model for DVFS and memory frequency scaling on a real system for the first time. They used a stall-based model. They suggested several parameters in the performance equation to capture out-of-order execution and different memory access latencies.

Regarding runtime algorithm and system design, Deng et. al. [62] suggested a gradient-descent heuristic algorithm to iteratively test performance under different system configurations on a per-core and memory basis. The algorithm can suggest a configuration while considering a performance degradation threshold. Ercan et. al. [69] proposed an online algorithm to coordinate CPU and memory DVFS. This showed a higher improvement in energy efficiency compared with isolated control policies. With a goal of power capping, Liu et. al. [137] coordinated CPU DVFS with memory frequency scaling to achieve improved performance. They utilized a queueing model to predict job processing on a many-core system. The proposed algorithm can achieve better performance than CPU DVFS-only methods. Sundriyal et. al. [184] proposed an online solution for DVFS and memory frequency scaling for the NWChem application. Begum et. al. [30] presented algorithms to explore the DVFS and memory frequency scaling space and looked for the optimal configuration in terms of energy consumption. Rao et. al. [164] proposed a runtime system coordinating DVFS and memory bandwidth throttling for the android mobile platform. Other research has focused on embedded platforms and investigated coordinating DVFS with Device Power

Management (DPM) which is generally the same as memory power states. Some researchers [63, 89, 121] explored the combined effect on energy efficiency optimization and proposed a corresponding algorithm to figure out the timeline or schedule of power state switching.

DVFS and DCT: Curtis-Maury et. al. [54] presented a statistical model using multivariate linear regression to quantify the impact of DVFS and DCT on performance in terms of instructions per cycle (IPC). They validated the model using runtime design aiming to reduce energy consumption and EDP. They also validated that the coordinated approach is better than isolated control of DVFS and DCT. Li et. al. [134, 135] conducted a profiling based analysis of the MPI traces for hybrid MPI/OpenMP applications and proposed a slack time based MPI level execution time model. They also used the multivariate linear regression approach to quantify the impact of DVFS and DCT on execution time of OpenMP phases. Then they developed an algorithm for improving energy efficiency at runtime for MPI/OpenMP applications. Cochran et. al. [47] proposed another way of designing runtime solutions under DVFS and DCT. Instead of modeling the performance, they presented a multinomial logistic regression based approach to map runtime characteristics to configurations of clock frequency and concurrency level. This approach also yields great scalability as the configuration space expands. Imamura et. al. [107] discussed the factors that can have an impact on performance under DCT and DVFS and observed that NUMA architecture is critical for accurate performance modeling. They used Artificial Neural Networks for performance modeling under DVFS and DCT, which captures the nonlinear effect of thread allocation for NUMA systems. They further implemented this model in a runtime system design. Similarly to DCT, Srinivasan and Bellur [178] proposed an execution time model under DVFS and CPU allocation for virtual machines using a linear regression-based method.

DVFS and PCPG: Liu et. al. [138] combined DVFS with power state adjustment for

processors in data centers to reduce the power consumption. The performance model under DVFS and power states is discrete: a case for CPU bound applications and a case for memory bound applications. They used the queueing model to emulate the job queue for data centers and assumed service time for each job follows random distribution. The impact of DVFS on the time of CPU bound jobs is modeled proportionally, whereas the assumption of no impact on the time of memory bound jobs perseveres. Low power states are triggered when the system becomes idle and end when a new job arrives. They proposed an algorithm for the runtime system which identifies the best setting for power consumption. Validation using real data center traces shows power savings that outperform conventional power management strategies. Vega et. al. [188] showed the need to coordinate DVFS and PCPG instead of letting them work independently. They suggested an algorithm for making online decisions as to when to enable DVFS and when to enable PCPG. The validation showed comparable energy efficiency using their approach to baseline power management approaches; and avoided the performance loss from DVFS and PCPG.

GPGPU Related: Komoda et. al. [120] exploited load imbalance and task mapping for DVFS control of both CPU and GPU. by applying DVFS control to both CPU and GPU as well as task mapping for CPU-GPU applications. They empirically modeled the performance (i.e. time) as a function of CPU frequency, GPU frequency, percentage of workload on CPU, and percentage of workload on GPU. They carried out offline profiling and found a near optimal setting which can be applied to the applications. Their power capping case study showed comparable performance to the ideal case.

Paul et. al. [158] adapted three techniques for the energy efficiency problem in high performance GPUs: GPU core frequency, number of active cores on GPU, and GPU memory frequency. They proposed a performance sensitivity model as the first step in a two-step configuration prediction algorithm called Harmonia. The second step is to further balance the

hardware performance at finer granularity based on performance feedback. Their approach can improve energy efficiency with negligible performance loss.

Ma et. al. [144] also proposed a two-step solution for improving the energy efficiency of CPU-GPU systems. They first allocated different portions of the workload on CPU and GPU according to workload characteristics. The goal of this step was to balance the workload of the CPU and GPU to make them finish at the same or a similar time. Second, they adapted the GPU DVFS for both cores and memory to gain energy reduction with minimal performance loss. Validation shows average energy savings of 21.04%.

The performance effects of various operating modes have been studied mostly in isolation. Dynamic voltage and frequency scaling (*DVFS*), the automated adjustment of processor power and speed settings, has been explored extensively [39, 82, 106, 169]. Analogous research on the effects of dynamic memory voltage and frequency throttling (*DMT*), the automated adjustment of DRAM power and speed settings, has emerged [56, 60, 133]. Other memory power modes such as dynamic bandwidth throttling [1] (*DBT*), where one or more idle clock cycles are inserted between memory accesses to lower peak bandwidth, are emergent. Dynamic concurrency throttling (*DCT*), the automated adjustment of thread concurrency, has also received widespread attention for some time [53].

While some have attempted to study the combined effects of two types of operating modes (e.g., CPU and memory scaling [56, 60], CPU scaling and concurrency throttling [53]), to the best of our knowledge, no one has accurately modeled the combined effects of CPU throttling, memory throttling, and concurrency throttling.

[1] C.-H. R. Wu, "U.S. patent 7352641: Dynamic memory throttling for power and thermal limitations." Sun Microsystems, Inc., issued 2008.

2.4 Performance Optimization Under Power Cap

Power capping is one of the most important application scenarios for coordinating multiple power management techniques. A number of research projects focus on improving system efficiency by maximizing performance under a given power budget for exascale supercomputers.

Power capping generally relies on an accurate estimation of power consumption of parallel applications. Komoda et. al. [120] proposed an empirical power model to limit the peak power consumption. Their approach determines the optimal CPU frequency and the workload distribution between CPU and GPU to meet the pre-assigned power budget. Tsuzuku and Endo [186] applied a power model and performance model on a CPU-GPU system for power capping using DVFS. Wu et. al. [200] focused on coordinated DVFS and task allocation on a heterogeneous architecture: the ARM/FPGA SoCs. [201] developed a model and runtime support for dynamic power capping using DVFS, data partitioning, and core allocations. Fujimoto et. al. [78] proposed a runtime power capping system for FPGAs leveraging off-chip memory throttling. Bailey et. al. [21] proposed a performance model on a heterogeneous system with both CPU and GPU. They considered configurations including device type, CPU frequency, and thread concurrency. They compared performance and power constraint violations with frequency-only methods. Cochran et. al. [46] explored the opportunities for accurate power capping using DVFS and thread packing on shared-memory systems. Sensi et. al. [58] proposed a runtime algorithm for selecting the best configuration of CPU frequency, thread concurrency, and thread placement for power capping. They analyzed the application behavior and calibrated the performance model offline for a number of configurations.

Running Average Power Limit (RAPL [55]) is a popular tool from Intel for measuring power

consumption and limiting the peak value of either the CPU or the memory. RAPL manages power capping using a proprietary power model. By monitoring the system status through model-specific registers (MSRs), RAPL estimates the power consumption with a power model for each system state. To comply with the power budget, RAPL modifies system states using DVFS or DMT.

RAPL can efficiently control peak power consumption under a power budget with negligible overhead. After setting the power budget through the RAPL interface, the system can usually reach a stable operating state within a very short period. Zhang et. al. [204] investigated the stability of performance for different parallel applications when they cap the power consumption using RAPL on a shared-memory system. Pedretti et. al. [159] also demonstrated that node level power capping using RAPL complements p-state control (i.e. DVFS).

Gholkar et. al. [90] investigated the issue of workload imbalance for parallel applications running on large scale systems. By allocating a power budget between processors using RAPL, they improved the overall performance and energy efficiency.

Ellsworth et. al. [68] presented a heuristic runtime power allocation method to dynamically adjust the power budget at socket level for MPI applications. They set the power bound of each socket using RAPL. Their power-aware scheduling requires little prior knowledge of an application and can improve job runtime performance.

Borghesi et. al. [33] explored the opportunities for performance optimization under power capping by scheduling and re-ordering multiple jobs on large scale systems. The goal of optimization is to minimize quality-of-service (QoS) disruption given a power budget.

Bailey et. al. [22] formulated the power capped performance for hybrid MPI/OpenMP applications as a linear programming optimization problem. They suggested optimal con-

figurations of the DVFS state and thread concurrency number for a runtime system.

Marathe et. al. [145] considered critical path detection for improving power capped performance under DVFS and DCT. They proposed a runtime system which detects where the critical paths would most likely happen and allocates more power budget accordingly. DVFS and DCT are adapted to achieve the best performance. The validation shows up to 30% performance improvement for hybrid MPI/OpenMP applications.

Bari et. al. [26] applied RAPL for power capping to investigate the impact of configurations of threads, thread scheduling policies, and chunk size. They showed their configuration selections can achieve better performance under power capping than the default.

Ge et. al. [83] studied the impact of power allocation between CPU and memory on performance. They demonstrated that coordinating between DVFS and DMT can improve performance over isolated control. They further proposed an empiricalmodel [84] to predict the power allocation between CPU and memory, and the thread concurrency on shared-memory systems. Zou et. al. [208] proposed a power aware allocation framework to configure power allocation, thread concurrency and application placement on distributed clusters.

In comparison, our work can coordinate three throttling techniques while previous approaches coordinate no more than two techniques. In addition, our approach is general and applies to borad set of parallel applications.

Chapter 3

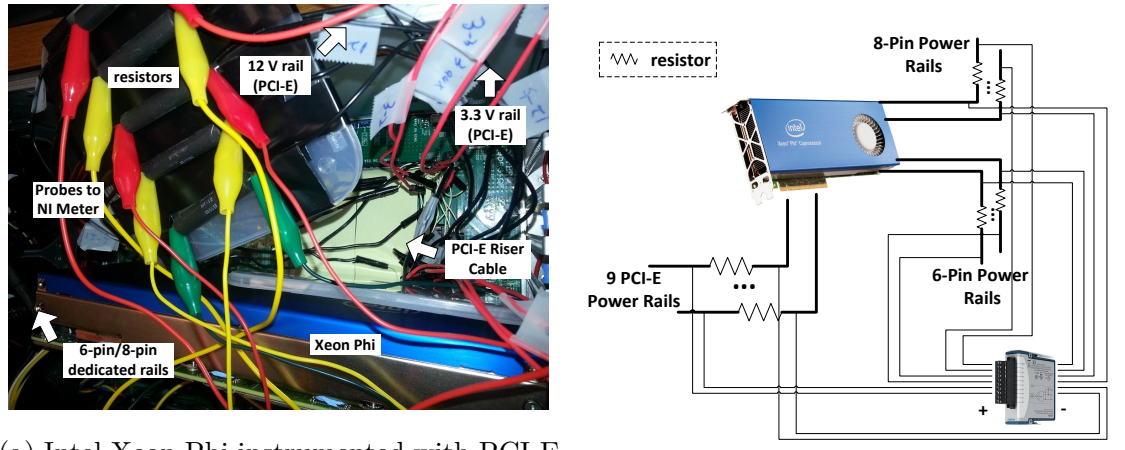
Power Measurement and Analysis

3.1 Tradeoff of Performance and Energy on Intel Xeon Phi

3.1.1 Methodology

The PowerPack framework [85] is a toolkit that is composed of both hardware and software components. The hardware components include AC power meter and DC power data acquisition devices. The software components contain drivers for measuring devices, and APIs for controlling profiling and data synchronization. Our PowerPack framework provides two unique features for power-performance analysis: 1) fine-grained component-level power measurement and 2) automatic power profiling for individual functions in an application. PowerPack uses direct measurement to isolate component level power consumption and requires hardware instrumentation of the power supply rails. Each system component (e.g. CPU, memory, hard drive, etc.) may be powered by several individual power rails. PowerPack measures power consumption through each dedicated power rail, and then uses the sum from all these rails as the component power.

In order to obtain accurate and fine-grained power readings from Xeon Phi, we required our extended profiling tool to: 1) directly measure power consumption of the Xeon Phi card



(a) Intel Xeon Phi instrumented with PCI-E riser cable.

(b) Power Measurement of Intel Xeon Phi.

Figure 3.1: Physical measurement of the Intel Xeon Phi using extensions to the PowerPack infrastructure for accelerator support. In this setup, we use a PCI-E riser cable to isolate the power rails for the system under test.

through the PCI-E bus and output fine-grained power measurement data along with power readings for other system components (e.g. CPU, Memory, etc.) in a readable format and 2) enable automated function level power profiling for Xeon Phi codes.

Fine-grained power measurement of accelerators: Current accelerators such as the Intel Xeon Phi and NVIDIA GPUs are powered through the PCI-E slot which is powered by the system power supply rails. Because these system power supply rails are shared by PCI-E and other system components such as CPU and Memory, it is difficult to separate the rails powering the PCI-E slot from other components. Thus, we purchased a PCI-E riser cable to isolate and measure all power supplied to the accelerator card – we identify and measure power over the PCI-E power pins. The riser cable implementation is shown in Figure 3.1. Figure 3.1a shows us physically how we instrument Intel Xeon Phi with the PCI-E riser cable. We also extended the software architecture of PowerPack to profile performance of functions native to the accelerator cards.

More precisely, we instrument five +12V DC pins and four +3.3V DC pins on the riser cable

according to the PCI-E implementation guide and later confirmed through testing. The instrumented PCI-E riser cable is used to connect Xeon Phi and motherboard through the PCI-E slot. PowerPack then uses the NI cDAQ9172 meter connected to the riser cable to measure the power coming through the PCI-E slot. Software collects the power data from the 9 PCI-E power pins and the 6-pin, and 8-pin dedicated power rails. The approach of measuring Intel Xeon Phi power, including from both PCI-E slot as well as dedicated power rails, is shown in Figure 3.1b.

Instrumentation using PCI-E riser cable enables us to measure the current passing through PCI-E interfaces. Because this power measurement approach is portable to any hardware (such as Ethernet card, GPGPU, etc.) that is completely or partially powered through the PCI-E slot, PowerPack framework can be easily deployed on a new system.

Automatic function-level profiling of accelerators: In order to fully automate data collecting/synchronizing process and provide function level finer power measurement, we extend the current PowerPack with new user friendly APIs. We also implement several service processes in order to cooperate with these APIs. PowerPack 4.0 system structure and data flow are shown in Figure 3.2.

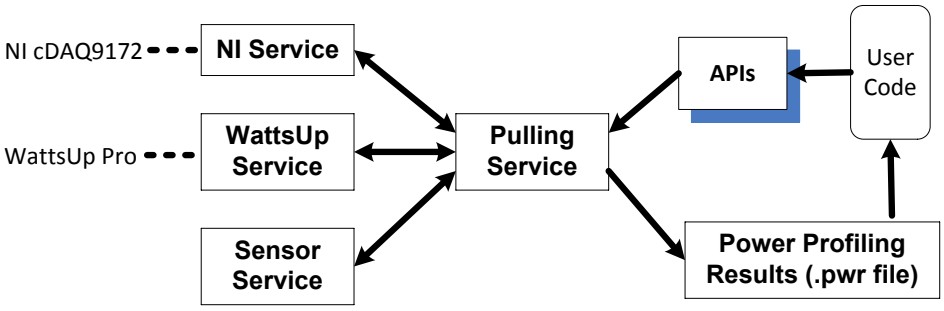


Figure 3.2: Software architecture extensions to the PowerPack infrastructure to support function-level profiling and automation of accelerator measurements.

Figure 3.2 shows the software architecture pulling service process and a set of meter/sensor

service processes running on our data collection system (a Linux server in our experimental setup). Each meter/sensor service process corresponds to one digital meter or sensor we collect readings from. The pulling service process is responsible for monitoring messages from user programs and also synchronizes the data streams from multiple meter/sensor service processes. To synchronize the pulling service process with the running application, profiled applications trigger message operations through a set of PowerPack function-level APIs and then inform the pulling service process to take corresponding actions to annotate the power profile. The pulling service process also owns the central clock and uses it to record data streams and messages from the meter/sensor services and the profiled application to eliminate time skew. Based on the experiments, the data transfer latency from the meter service process to the pulling process is within 0.09ms in our setup. Table 3.1 lists the function-level APIs (only showing the C APIs here but our library supports FORTRAN interface as well).

Table 3.1: PowerPack Function-Level APIs

powerpack_init()	Connect to the pulling service and start profiling.
powerpack_end()	Finish profiling and request results from server.
powerpack_func_start()	Start a new profile session and label it.
powerpack_func_end()	Stop the current profile session.

Currently, our extended PowerPack can capture power readings from the Intel Xeon Phi at a rate of 50 samples/sec. This sampling rate is flexible and was chosen empirically to provide a reasonable balance between data generation and profiling granularity. Additional details regarding the PowerPack infrastructure design and capabilities can be found on SCAPE lab software page.[1]

[1]http://scape.cs.vt.edu/software/powerpack-4-0/

3.1.2 Experimental Setup

Testbed

The Xeon Phi 5110P architecture studied uses 61 Cores (60 cores for computation, 1 core for OS) connected to a memory controller and PCI-E interface via a bidirectional ring bus. Table 3.2 lists key architectural details for all studied hardware including the Xeon Phi, Intel Sandy Bridge CPU and Tesla Fermi c2050 GPGPU.

Table 3.2: Xeon Phi, SandyBridge, and Fermi GPU Details

Xeon Phi 5110P		**SandyBridge Xeon E5-2620**		**Tesla c2050**	
# of Cores	60+1	# of Cores	6	# of Cores	448
Threads	240	Threads	12	Warp Size	32
Clock	1.05 GHz	Clock	2 GHz	Clock	1.15 GHz
L1 Cache	32 KB	L1 Cache	32 KB	L1 Cache	64 KB
L2 Cache	512 KB	L2 Cache	256 KB	L2 Cache	768 KB
Memory	8 GB	L3 Cache	15MB	Memory	3 GB
Bandwidth	320 GB/s	Memory	32 GB	Bandwidth	144 GB/s
		Bandwidth	100 GB/s		

Benchmarks

In this paper, we use two benchmark suites to evaluate power-performance of the Intel Xeon Phi, SandyBridge, and NVIDIA Tesla GPGPU. They include the SHOC benchmark suite from Oak Ridge National Laboratory [154] for Xeon Phi comparisons to the Tesla GPU and the Rodinia [41] benchmark suite from the University of Virginia for Xeon Phi comparisons to the Intel SandyBridge host CPU.

Applications from each suite were selected for two primary reasons: 1) they represent a spectrum of HPC application execution patterns with a variety of memory to computation

Table 3.3: SHOC and Rodinia Benchmarks Used in This Study

Benchmark	Suite	Application Type (primary)	Platform Tested	Workloads	Mode
FFT	SHOC	Memory Bound	Xeon Phi	s3<s4	Offload[1]
MD	SHOC	Computation+Memory Bound	Xeon Phi	s1<s2<s3<s4	Offload[1]
GEMM	SHOC	Computation Bound	Xeon Phi	s3<s4	Offload[1]
			Tesla	s3<s4	Offload[1]
Reduction	SHOC	Memory Bound	Xeon Phi	s3<s4	Offload[1]
			Tesla	s3<s4	Offload[1]
BFS	Rodinia	Memory Bound	Xeon Phi	default	Native[2]
			SandyBridge	default	CPU[3]
HotSpot	Rodinia	Memory Bound	Xeon Phi	default	Native[2]
			SandyBridge	default	CPU[3]
Kmeans	Rodinia	Computation Bound	Xeon Phi	default	Native[2]
			SandyBridge	default	CPU[3]
StreamCluster	Rodinia	Computation Bound	Xeon Phi	default	Native[2]
			SandyBridge	default	CPU[3]

[1] initiate the program on CPU and then launch the workload to run on accelerator (s).
[2] code is written in regular OpenMP. Using icc -mmic to compile the code so it can entirely run on Xeon Phi.
[3] code is written in regular OpenMP. It only runs on host multi-core CPUs.

ratios and 2) for pairwise comparisons, we favored codes that we could run natively or via cross-compilation for direct comparisons across architectures. Table 3.3 provides a summary of key attributes for these benchmarks.

We define the *kernel* of a benchmark as the computation and memory phases following initialization and beginning when data transfer to the accelerator completes. For direct comparisons across architectures, we primarily compare the kernel performance, power and energy use unless otherwise noted.

3.1.3 Results and Analysis

Xeon Phi 5110P

The SHOC suite was selected as representative of HPC applications targeting accelerators. We focus on FFT, GEMM, MD and parallel Reduction as representative of a span of memory

to computation ratios on the Xeon Phi architecture.

The SHOC applications have been optimized for load balancing, loop unrolling and vectorization. *Offload Mode* indicates applications are initiated on the host SandyBridge CPU and fully offloaded for computation on the accelerator. Though the Intel Xeon Phi architecture has 61 physical cores, for simplicity we focus on the 60 cores used for computation – the remaining core supports the OS. Whenever possible, we scale the applications to 240 threads to account for the additional parallelism provided via 4 hyper-threads per core on the Intel Xeon Phi. In some cases, smaller workloads lack the computations to scale effectively beyond a small number of cores. Thus, we focus on larger workloads (e.g. s3 and s4) in such cases. For additional performance data, we rely on native hardware performance counters accessed using the *perf* profiler tool supported by the native Linux kernel on Xeon Phi [6].

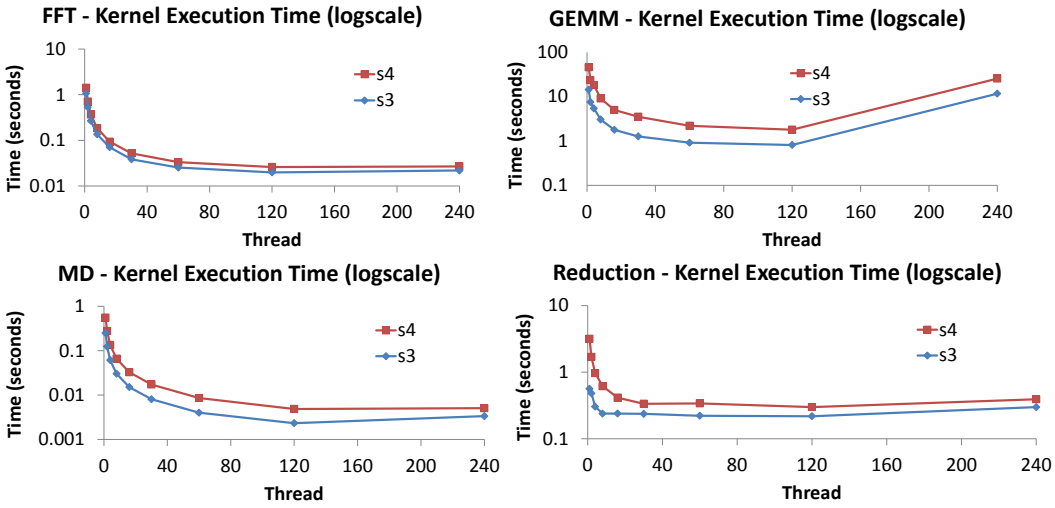


Figure 3.3: Performance of four SHOC Benchmarks varying in memory to computation ratio. Results for increasing numbers of threads for two of the largest available workloads (s3 and s4) are shown. Performance measurements focus on kernel execution time and exclude time spent for initialization and data transfer to accelerator.

Performance Analysis:

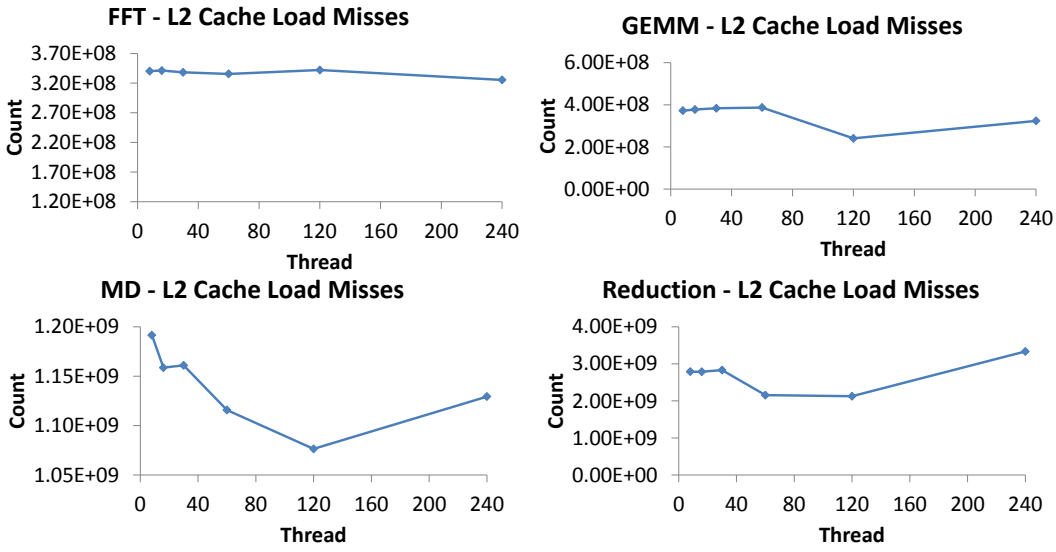


Figure 3.4: L2 data cache load misses for four SHOC Benchmarks varying in memory to computation ratio. Results for increasing numbers of threads for one of the larger available workloads (s3) are shown. Results for other workloads (e.g. s4) are similar.

Figure 3.3 shows the execution time of four SHOC applications for various numbers of threads on the Xeon Phi for two large workloads (i.e. datasets). Each *x-axis* tracks the number of threads used while the *y-axis* plots execution time logarithmically to emphasize differences.

FFT, GEMM and MD do not scale well beyond 120 threads for these workloads. Reduction performance stops scaling at about 30 threads. Beyond 120 threads, the resources of the Xeon Phi are fully saturated resulting in no performance gain. GEMM loses performance beyond 120 threads. Though others [80] have discussed the performance limitations of the Xeon Phi, we analyze the root cause further for completeness and to help explain power and energy characteristics later in this paper.

We identify several possible causes of performance loss in the Xeon Phi: A) oversubscribing cores; B) small L1 and L2 caches per core; C) lack of large shared L3 cache; and D) memory contention. The degree to which performance is affected by these causes is determined

by the memory access pattern and the memory to computation ratio of the benchmark. Oversubscribing cores can cause higher resource contention and pipeline latency. The small L1 (64 Kbytes) and L2 (512Kbytes) caches of the Xeon Phi combined with the lack of a large, shared L3 cache limits the size of working sets that can fit fully in the caches. For larger workloads, this can lead to increased average memory access latency and performance loss. The bidirectional ring interconnect of the Xeon Phi can lead to on-chip network congestion and slowdown for some applications. High memory to computation ratios combined with high thread counts can lead to increased memory contention as well.

The SHOC MD benchmark calculates an n-body simulation and results in cyclic memory- and computation-bound phases. Figure 3.3 shows the strong scaling results for MD. Figure 3.4 plots the L2 data cache misses (*y-axis*) as a function of thread count (*x-axis*) for MD and the other codes studied. There is a significant increase in L2 cache misses beyond 120 threads. MD performance is thus a victim of the combined effects of core oversubscription, smaller or nonexistent caches, and memory contention. The SHOC GEMM benchmark, which performs a general matrix multiply, suffers (albeit more dramatically) from similar effects for performance degradation beyond 120 threads.

The SHOC Reduction benchmark performs a parallel vector reduction operation common to scientific computation. Parallel Reduction performance scales to about 30 threads for the workloads studied (see Figure 3.3). Since all communications within the Xeon Phi are memory transactions, the Reduction benchmark is memory bound and sensitive to small, isolated L1 and L2 cache sizes. These effects lead to poor scalability beyond 120 threads as Figure 3.3 illustrates. However, the L2 cache (see Figure 3.4) misses decrease for $30 \leq$ num_threads ≤ 120 and then increase beyond 120 threads. We did some further analysis of the Reduction benchmark code and some additional experiments with larger workload sizes. Increasing the workload size had little to no effect on the scalability analysis. Though we

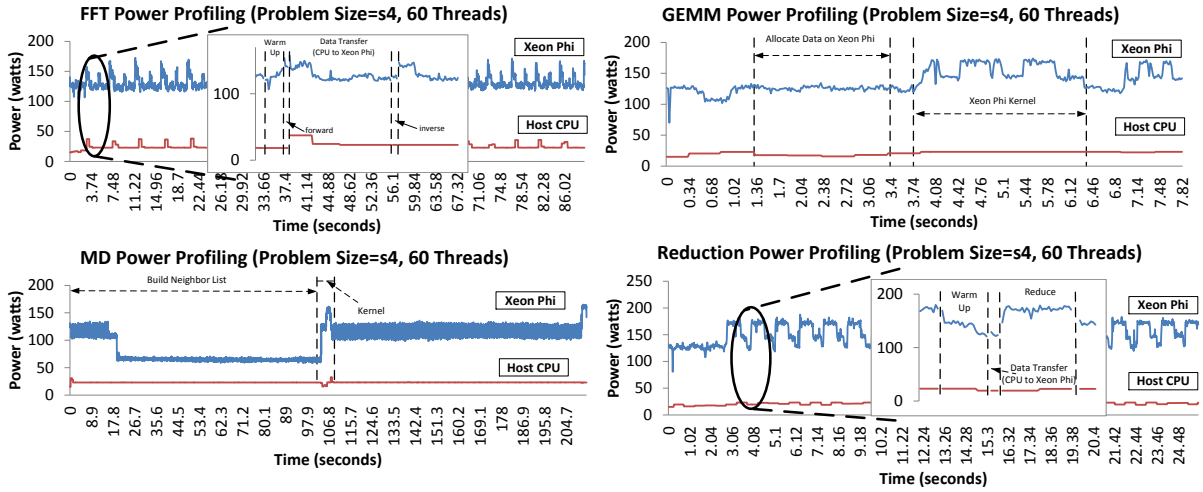


Figure 3.5: Power profiles of four SHOC Benchmarks varying in memory to computation ratio. Results for 60 threads (corresponding to 1 thread per physical core) on the largest available workload (s4). Insets show function-level profiling for FFT and Reduction Benchmarks provided by PowerPack extensions for accelerators. Power data includes entire card packaging (Xeon Phi) or Host CPU (SandyBridge) in isolation. Other system components power consumption contribute only nominally and are not shown to simplify presentation.

need to do further analyses, we concluded that the limitations from 30 to 120 threads are algorithmic since computation is dominated by memory transfers despite the working set fitting in the caches of the Xeon Phi. Beyond 120 threads it is likely the additional overhead for oversubscribing dominates the execution time as well.

Power and Energy Analysis:

Figure 3.5 shows the power profiles of the Intel Xeon Phi and SandyBridge host CPU on four SHOC applications with a sampling rate of 50Hz – determined empirically to balance the need for granular detail without generating an unwieldy amount of data. Our extended version of PowerPack provides detailed power measurements (*y-axis*) and automated labeling of key functions for each benchmark (see labels and insets).

The power profiles for each benchmark are distinct and meaningful. For example, the power

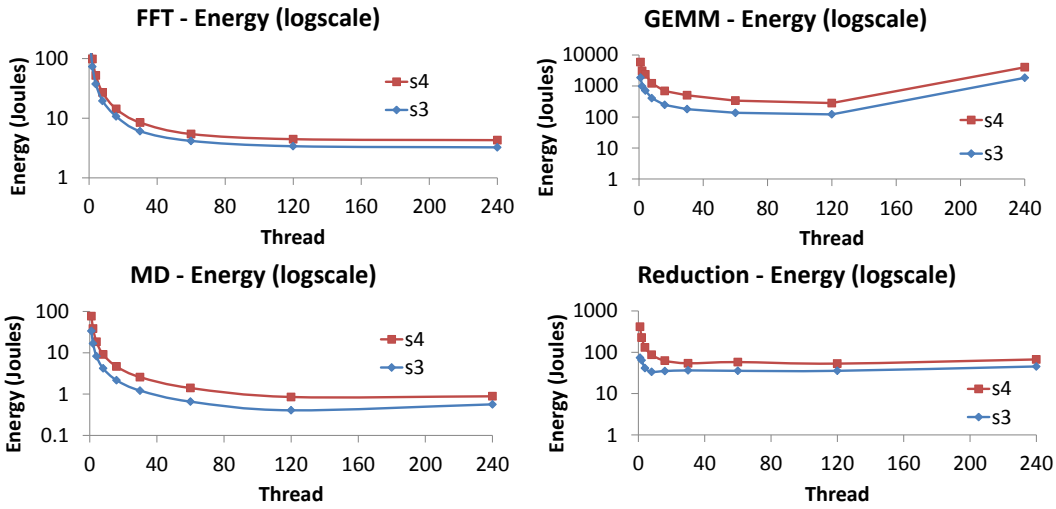


Figure 3.6: Energy consumption of Xeon Phi (isolated) on four SHOC Benchmarks varying in memory to computation ratio. Results for increasing numbers of threads for two of the largest available workloads (s3 and s4) are shown.

graph of FFT clearly reflects the power peaks (caused by *inverse* and *forward* functions) and valleys (where each iteration begins or ends). Another striking feature of these graphs is evidenced in the juxtaposition of the SandyBridge host CPU power consumption in isolation yet synchronized with the Xeon Phi power profile. For example, the power plateaus on the host CPU correspond to the data transfer phases on the CPU causing considerable power use on both devices for data transfers.

Figure 3.7 shows the power consumption (*y-axis*) of the Xeon Phi for increasing numbers of threads (*x-axis*) for two of the largest available workloads (s3 and s4) on the four SHOC benchmarks studied.

Energy is the product of average power and time. Figure 3.6 shows the energy consumption (*y-axis*) of the Xeon Phi for increasing numbers of threads (*x-axis*) for two of the largest available workloads (s3 and s4) on the four SHOC benchmarks studied. In all cases, the energy use follows the same scalability trends observed for the execution time analysis shown

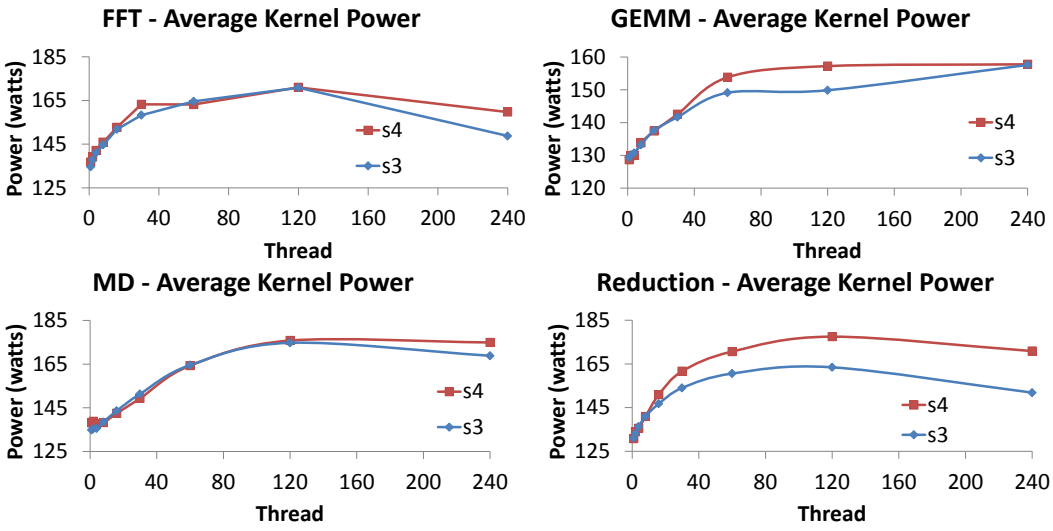


Figure 3.7: Average kernel power consumption after initialization and data transfer to accelerator complete for Xeon Phi (isolated) on four SHOC Benchmarks varying in memory to computation ratio. Results for increasing numbers of threads for two of the largest available workloads (s3 and s4) are shown.

in Figure 3.3. FFT, GEMM, and MD scale reasonably well through 120 threads while the Reduction code sees energy scalability drop off beyond 30 threads. For the largest workload (s4), MD spends the most time in its computational kernel followed by FFT, Reduction, and GEMM in that order.

We observe that energy scales well with number of threads beyond 30 (energy consumption stays relatively constant) for FFT, MD and Reduction. This indicates that for these applications we should choose the parallelism and problem size that give the best performance as the optimal configuration. GEMM's energy graph shows a very similar trend until it hits 120 threads and then its energy consumption increases substantially. This is caused by the dramatic performance decrease due to core oversubscription and main memory contention described in Figure 3.3.

Figure 3.8 shows the portion of the total energy use allocated to data transfer phases versus

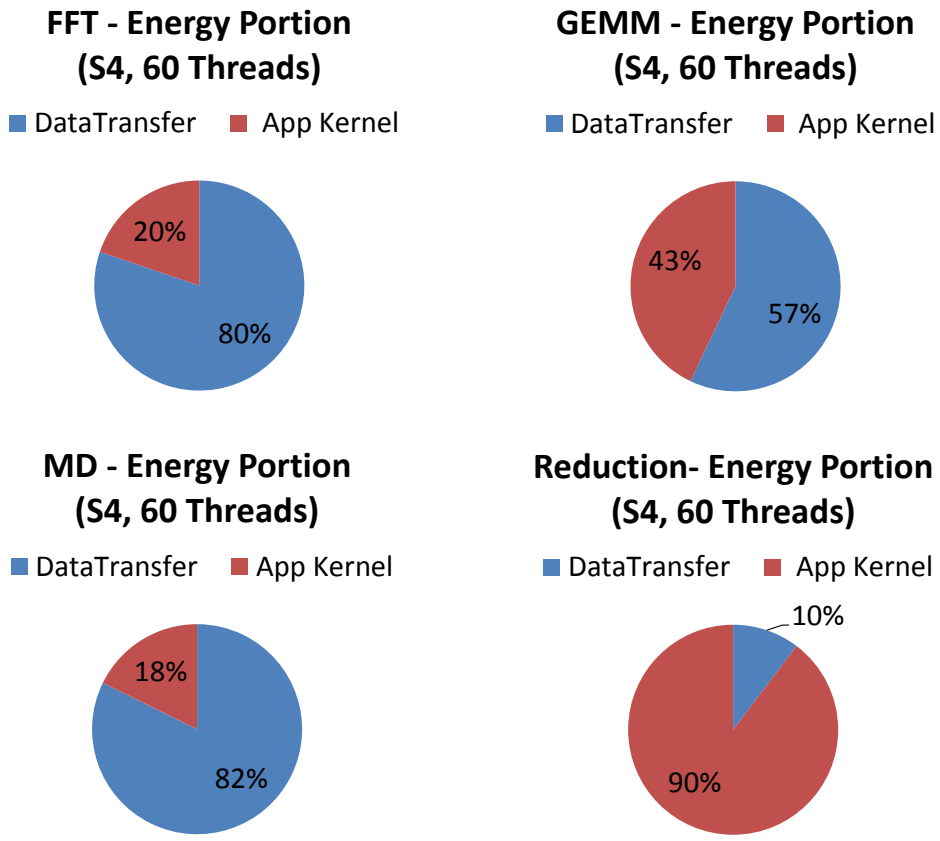


Figure 3.8: Energy budget allocated to data transfer versus application kernel execution on Xeon Phi. (s4, 60 threads)

application kernel phases for each benchmark. All cases use the largest s4 workload and 60 threads. The data transfer phase includes all energy used from the beginning to the end of allocating data from host to the Intel Xeon Phi. The kernel phase includes all energy used for one complete iteration of the kernel function on the Xeon Phi.

Data transfers consume a significant amount of energy for applications like FFT, GEMM, and MD on the Xeon Phi. Reduction uses the least energy for the data transfer though 10% is not insignificant. While other researchers [13, 153, 160] have indicated efficient data transfers are key to accelerator performance, quantifying the cost of data transfer energy

shows it varies significantly with application and can considerably reduce energy efficiency.

GFLOPS/watt: For better or for worse, throughput per watt has gained popularity in evaluating power efficiency of HPC applications. Figure 3.9 shows the GFLOPS/watt (y-axis) of four SHOC applications versus the number of threads (x-axis). The max power efficiency (in GFLOPS/watt) for GEMM, MD, and Reduction happens at 120 threads and the largest problem size studied (s4). For FFT, max power efficiency is achieved at 240 threads and the largest problem size (s4). This graph indicates that for these benchmarks increasing the problem size (i.e. working set) generally improves power efficiency on the Xeon. Since we were limited in the number and types of problem sizes to test, this only holds true for the benchmark parameters studied.

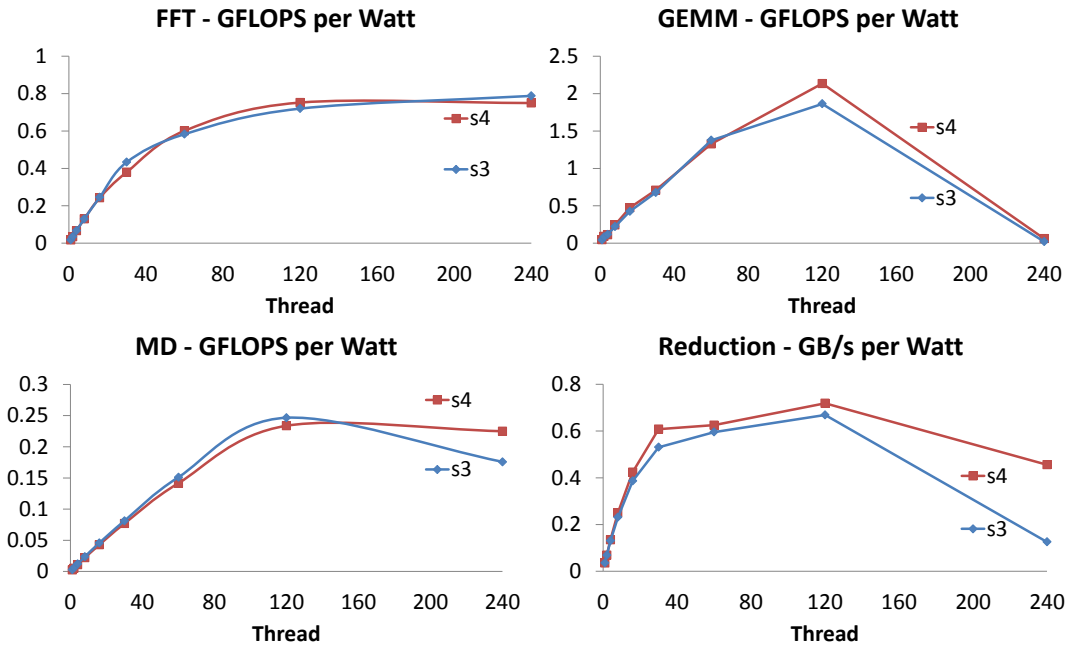


Figure 3.9: Power efficiency (GFLOPS/watt) of Xeon Phi (isolated) on four SHOC Benchmarks varying in memory to computation ratio. Results for increasing numbers of threads for two of the largest available workloads (s3 and s4) are shown.

Scalability Analysis: Figure 3.10 shows the energy efficiency (z-axis) of various combinations

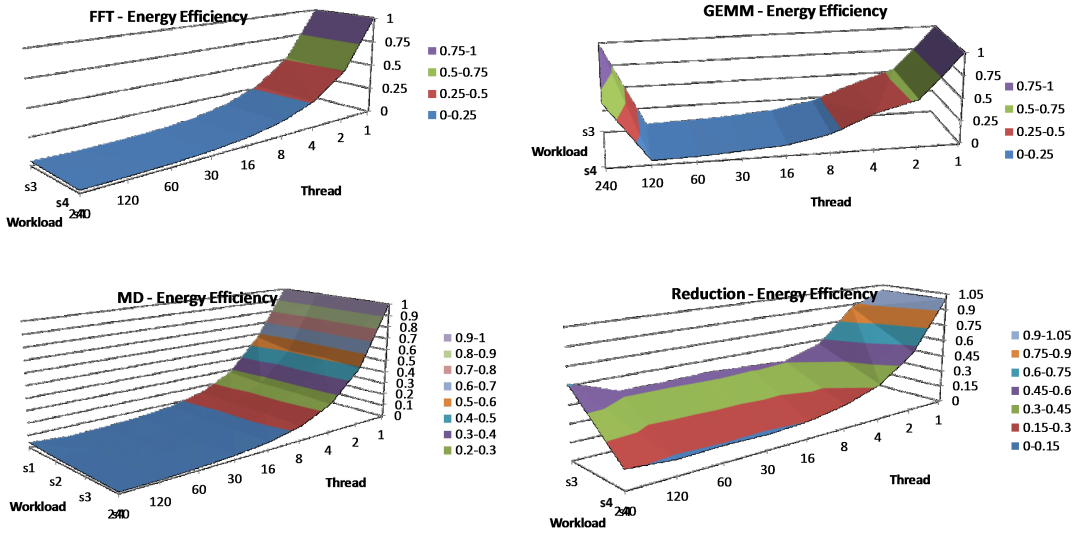


Figure 3.10: Xeon Phi (isolated) energy efficiency on four SHOC Benchmarks varying in memory to computation ratio. Results for increasing numbers of threads for all available workloads are shown. Energy efficiency is defined as: *Energy(n_Threads)/Energy(1_Thread) for each workload.*

of workloads (x-axis) and threads (y-axis) for the four SHOC benchmarks studied. Energy efficiency is calculated as the ratio of energy used for n threads over the energy used for 1 thread for a given problem size. These results are focused on the Xeon Phi efficiency in isolation (disregarding the costs of data transfer). The minimums of the resulting planes offer the thread count with the best energy efficiency for a given workload. For FFT and MD, 240 threads provides the best energy efficiency across the workloads. For GEMM, 120 threads provides the most efficient operation on the Xeon Phi for the workloads studied. The Reduction code was most sensitive to its workload and the best thread count varies considerably.

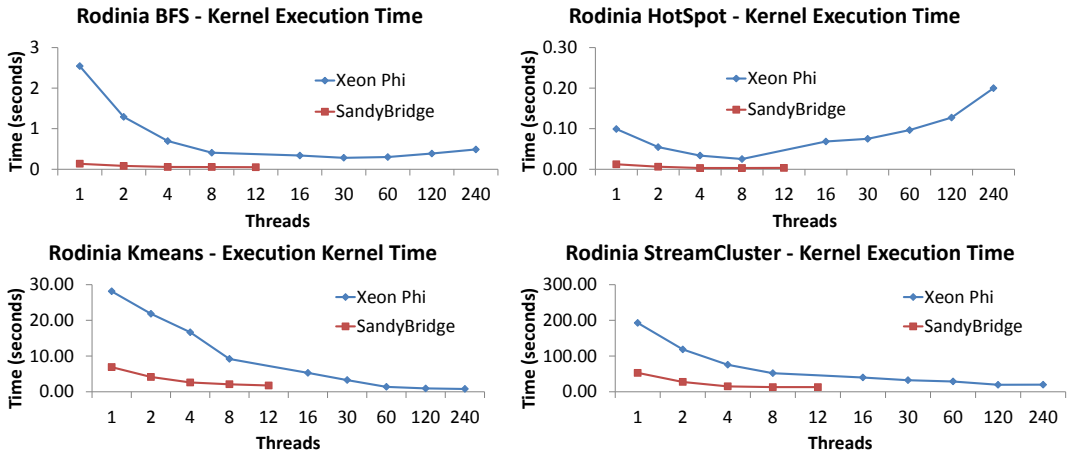


Figure 3.11: Performance comparison of Xeon Phi (isolated) and SandyBridge Host CPU (isolated) on four Rodinia Benchmarks. Results for increasing numbers of threads for the default workloads are shown. We scale the benchmarks to the maximum number of available virtual cores for Xeon Phi (240) and SandyBridge (12). Performance measurements focus on kernel execution time and exclude time spent for initialization and data transfer to accelerator.

Intel Xeon Phi versus Sandy Bridge Host

Programmers have the option of fully offloading code from the host CPU to an accelerator. In this series of experiments, we attempt to analyze the efficiency of executing exclusively on our Intel Sandy Bridge multicore processor host versus the Intel Xeon Phi accelerator. We are limited in the types of codes that can run directly on both the Sandy Bridge host and the Xeon Phi. Thus, we use four representative Rodinia OpenMP applications. Figure 3.11 shows results running the OpenMP codes compiled for the Sandy Bridge host and also run without modification on the Xeon Phi. In both cases, the execution time (y-axis) is shown for the number of threads (x-axis) scaled to the maximum available on the target architecture (240 for Xeon Phi, 12 for Sandy Bridge).

Figure 3.11 shows summary results for these experiments. We observe: 1) For memory bound applications like BFS and HotSpot which have large memory footprints and frequent random

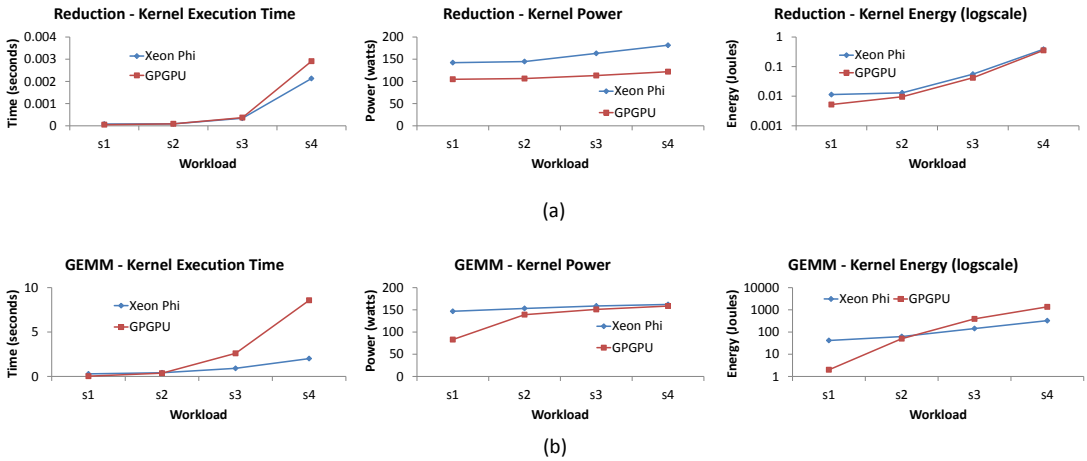


Figure 3.12: Performance, average power, and energy comparison of Xeon Phi (isolated) and NVIDIA c2050 GPU (isolated) on two SHOC Benchmarks (Reduction and GEMM). Results for increasing size of workloads (left to right on *x-axis*) for 120 threads (fixed). All measurements focus on kernel execution time and exclude time spent for initialization and data transfer to accelerator.

memory accesses Xeon Phi performs poorly compared to the Sandy Bridge host. Xeon Phi's simpler cores while greater in number lack the sophistication of a modern superscalar processors. The Xeon Phi cores operate at 1.05GHZ (versus 2GHz for Sandy Bridge) and lack a shared L3 cache (15MB L3 on Sandy Bridge). 2) For computation bound applications such as Kmeans and StreamCluster the Xeon Phi matches (StreamCluster) or exceeds the performance (KMeans) of the Sandy Bridge host. Clearly, the Xeon Phi performance increases substantially as the computation to communication ratio and potential thread-level parallelism increases. In both cases, the Xeon Phi would benefit from a native implementation not cross-compiled for the Sandy Bridge host.

Xeon Phi versus NVIDIA Tesla

Figures 3.12 (a) and (b) show the measured power, performance and energy of two SHOC kernels—GEMM and Reduction running on Xeon Phi 5110P and NVIDIA c2050 GPU. These

experiments were conducted under applicable optimization in each of these two architectures. Execution times are the best measured times for each workload and exclude initialization and data transfer to the accelerator. Power consumption is the average power over multiple runs for each workload. The number of threads was fixed at 120. Energy is calculated as average power times execution time and plotted logarithmically. The y-axis plots execution time (seconds), power (watts), and joules (log-joules) respectively for the aforementioned workloads (s1, s2, s3, s4).

For the memory bound Reduction shown in Figure 3.12 (a), the Xeon Phi outperforms the NVIDIA GPU for larger workloads (s3, s4). However, the Xeon Phi consumes more average power than the NVIDIA GPU for all the observed workloads. This effectively results in wasted energy for the smaller workloads running on the Xeon Phi architecture. For s1, s2, and s3 workloads, this energy does not result in significant performance gains. For s4, the Xeon Phi does slightly outperform the NVIDIA GPU potentially justifying the additional energy use.

For the computation bound GEMM application shown in Figure 3.12 (b), the Xeon Phi easily outperforms the older NVIDIA GPU for larger, more computationally demanding problem sizes (s3, s4). For smaller problems sizes, the Xeon Phi also consumes higher average power than the NVIDIA GPU, but for larger problem sizes (s3, s4) both accelerators consume comparable amounts of power. The logarithmic plot of energy use for GEMM highlights the differences across the problem sizes. For larger problem sizes (s3, s4), the NVIDIA GPU slower computation results in higher overall energy use despite the comparable average power use of the two accelerators.

As observed elsewhere in this study, the Xeon Phi potential for both performance and energy efficiency hinges heavily on the computational demand.

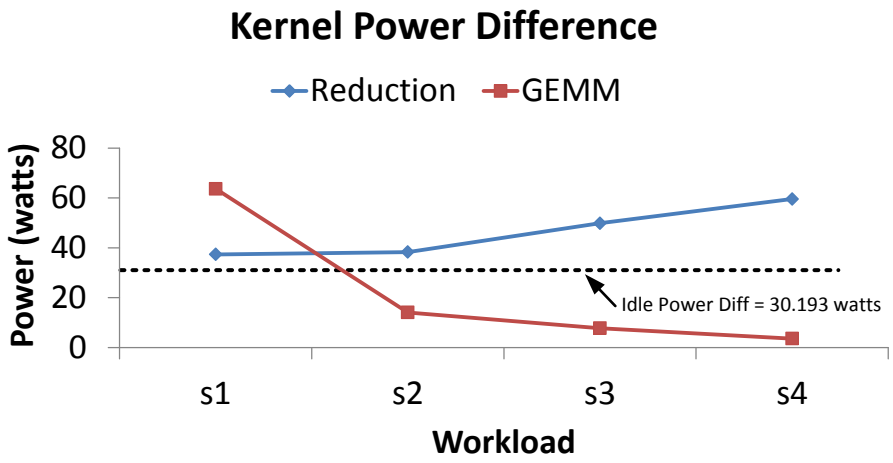


Figure 3.13: Average power difference of Xeon Phi (isolated) and NVIDIA c2050 GPU (isolated) on two SHOC Benchmarks (Reduction and GEMM). This power difference is not simply attributable to idle power differences as shown by the dotted line.

We have observed that the Xeon Phi can outperform the NVIDIA GPU for highly dense computational kernels and the average power of Xeon Phi always matches or exceeds the NVIDIA GPU in our experiments. We conducted a follow-on set of experiments to determine whether the extra power was simply do to a higher idle power in the Xeon Phi accelerator. This particular experiment (like many others herein) would have been exceptionally difficult without our PowerPack extensions that directly isolate the accelerator power.

Figure 3.13 shows the average power differences between Xeon Phi and the NVIDIA GPU. Power difference is calculated as: *(Xeon Phi Power) – (NVIDIA GPU Power)*. This figure shows results for increasing size of workloads (left to right on *x-axis*) for 120 threads (fixed). All of these measurements focus on kernel execution time and exclude time spent for initialization and data transfer to the accelerator. For example, Xeon Phi uses 65 more watts than than NVIDIA GPU running DGEMM on s1 workload. The dash line in the figure represents the idle power difference between Xeon Phi and Fermi GPU. Idle power is measured when

there is absolutely no device workload and all the cores are in deep sleep mode.

We observe the following from this set of experiments: 1) The idle Xeon Phi accelerator consumes 30.193 watts more power than NVIDIA Fermi c2050 GPU. 2) Differences are attributable to active power differences. 3) The power difference for Reduction, a memory intensive application, is divergent as problem size increases. Since performance is comparable at larger problem sizes, the gains from the Xeon Phi are arguably less for memory bound codes of significant scale. 4) The power difference for GEMM, a computationally intensive code, is divergent as problem size increases. The real gains in efficiency for the Xeon Phi are exhibited in computationally intensive kernels like GEMM.

3.1.4 Summary

Power Pack is a software-hardware infrastructure for profiling power consumption of system components (e.g. CPU) on high performance systems. We have extended the PowerPack toolkit to isolate and analyze the power and performance of Intel's Xeon Phi accelerators. The infrastructure can help developers collect useful data to improve performance and energy consumption of the codes. We demonstrated that the power and performance characteristics vary with the computation and memory characteristics of the applications under study. Studies of this type provide a vast amount of information above and beyond the raw total system numbers available from the Green500 and Top500 Lists. In contrast to architectural instruction-level performance models of the Xeon Phi, this work provides Xeon Phi efficiency data in the context of other system devices such as the host CPU. We used the infrastructure to quantify the energy cost of transferring data to an accelerator in a Xeon Phi system. To demonstrate the portability of our approach, we additionally evaluated and compared Xeon Phi power-performance efficiency to an NVIDIA Tesla GPU. Our results indicate that

although the Xeon Phi typically outperforms the older GPU, the results are mixed with regard to the energy efficiency. The application characteristics ultimately dictate which platform is superior. Our results are limited to the systems and applications tested. Thus in future work we plan to extend our infrastructure to measure other emerging accelerators and applications.

3.2 Memory Throttling on BG/Q: A Case Study with Explicit Hydrodynamics

3.2.1 LULESH

The Livermore Unstructured Lagrange Explicit Shock Hydrodynamics (LULESH) mini-app [116] provides a simplified source code that contains the data access patterns and computational characteristics of larger hydrodynamics codes. It uses an unstructured hexahedral mesh with two centerings and solves the Sedov problem that describes the evolution of a blast wave arising from a point explosion in a cold, uniform density, polytropic fluid. We chose it for this study because explicit hydrodynamics can consume up to one third of the compute cycles at the U.S. Department of Defense data centers.

From an application developer's perspective, LULESH can be divided into code phases or regions according to the physics performed [116]. This code breakdown allows us to understand changes in performance, power, and energy from one region to the next. On a multi-physics code, in addition to regions, we would be interested in changes between physics packages.

In this work, we study five regions that account for over 90% of the execution time of LULESH and present a diverse set of properties: *Region 1* performs the stress calculation routines;

Region 2 performs the hourglass calculation; *Region 3* consists of two memory-bound loops that update the velocity and position of nodes; *Region 4* includes `CalcKinematicsForElems` and `CalcMonotonicQForElems`, which gather values from nodes to element centers followed by compute-intense calculations; and *Region 5* includes `MonotonicQforRegions` and `MaterialApply`, which have a significant amount of control flow instructions and instructions with dependencies.

3.2.2 Memory throttling on BG/Q

In this subsection, we introduce BG/Q's memory throttling and its impact on system performance. A BG/Q compute node consists of a chip with 16 A2 user cores and 16 GB of SDRAM-DDR3 memory. Each core has four SMT threads and runs at 1.6 GHz. Each chip has a 32-byte, 1.33 GHz interface to memory for a peak read+write bandwidth of 42 GB/s.

BG/Q provides a built-in hardware feature on the DDR controller that introduces higher delays or *idle cycles* between each read and write to DDR. A user application can adjust this number, to slowdown the memory bus (and save power), as a parameter to the function `Kernel_SetPowerConsumptionParam`. This value ranges between 0 and 126 and can be specified at a node granularity. Each idle cycle is 1 DDR cycle at 1.33 GHz.

Impact on bandwidth and runtime

We ran the STREAM benchmark (copy, scale, add, and triad) with 16 threads on a BG/Q node to quantify the impact of memory throttling on bandwidth. The maximum bandwidth achieved under different memory speeds is shown in Figure 3.14. In the *baseline* case, we do not throttle the memory system (memory at full speed). As expected, as we add more idle cycles to the memory controller, memory bandwidth decreases.

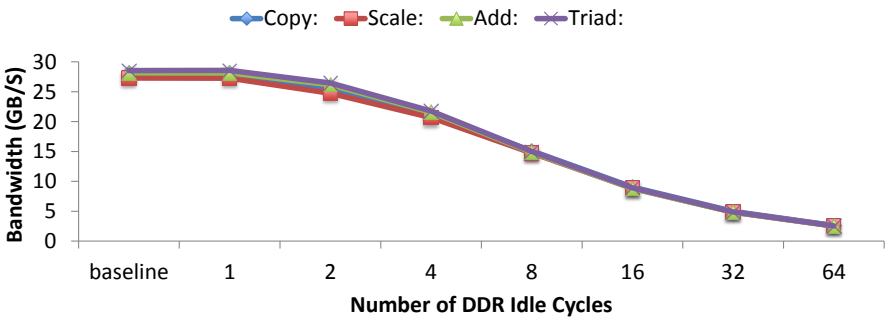


Figure 3.14: Effect of throttling on memory bandwidth.

We also measured the impact of throttling on the execution time of LULESH. Although we run experiments for all five regions, Figure 3.15 focuses on regions 3 and 4, which exemplifies the different computational characteristics across regions. As this figure shows, region 3 is more sensitive to memory bandwidth than region 4. As explained in Section 3.2.1, region 3 consists of two memory-bound loops that update the velocity and position of nodes. Region 3 shows degradation in performance between 8 and 16 idle cycles, while region 4 between 32 and 64. We will show later that we can use a simple model, based on performance counters, to predict the highest number of idle cycles to add, on a per-region basis, with a marginal effect on performance. The saved power could be shifted to other subsystems to improve performance.

Memory throttling latency

We now quantify how long it takes for an application to observe changes in memory speed as a result of executing `Kernel_SetPowerConsumptionParam`. This is important to understand when memory throttling can and cannot be used. If the latency is too long, throttling may not be applicable to short-lived regions.

We used STREAM again to exercise the memory system and allocated a separate tracer

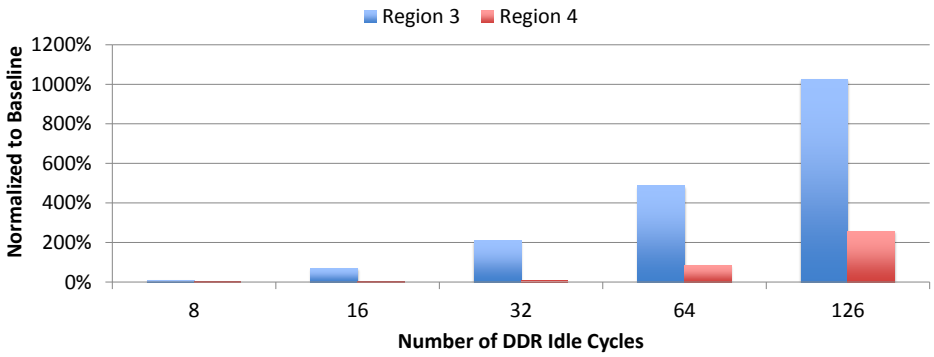


Figure 3.15: Effect of throttling on LULESH's execution time for input size 90 (90^3 elements) and 48 threads.

thread to measure memory bandwidth at regular intervals of 1, 10, 100, and 1000 μs. Using the tracer thread, we throttled the memory system and quantified the number of intervals it took to observe a difference in bandwidth. We found that an effective change in bandwidth occurs between 1 to 10 μs.

3.2.3 Measuring power

BG/Q systems are capable of measuring power and energy at a node-board granularity. Each board includes 32 compute nodes and 2 direct current amperage (DCA) modules. Each DCA has a microprocessor unit that measures current and voltage of at most seven domains. The compute nodes can read power/energy data via the EMON2 (Environmental Monitoring version 2) API. Through this API, a user application is able to retrieve cumulative energy consumption at a given point in time. With two snapshots, the EMON2 library computes the energy difference and the average power consumption for the given interval. In all of our experiments, we capture power and energy consumption every 10 ms of all seven domains. We focus on the processor and memory since the other domains consume mostly the same amount of power regardless of utilization (e.g., network links are always on). More

information on BG/Q's high-resolution power infrastructure and the proportions of static and dynamic power can be found elsewhere [31, 127].

3.2.4 Predicting optimal memory speed

On a given application, some code regions may be memory intensive while some others computation intensive. In those regions where computation is high and memory bandwidth utilization is low, we can throttle the bandwidth of memory system and save power by changing the length of idle time. If we can find the minimal memory bandwidth for a given region without significantly decreasing performance, we could reduce power and energy consumption.

The five code regions of LULESH have different computation and memory characteristics. So is the case when changing problem size[2] and level of concurrency (number of threads). In Figure 3.16, we show the lowest amount of throttling that can be applied to LULESH while maintaing at least 95% of its performance compared to the baseline (full memory speed). The top graph shows that the minimal memory throttling is region and concurrency dependent, while the lower graph shows that the minimal throttling for region 4 is also problem size dependent.

As shown in Figure 3.16, small problem sizes and low levels of concurrency do not require high memory speeds (low number of idle cycles). Regions 1, 2, and 3 are more sensitive to memory bandwidth, while region 5 is the least sensitive as it can absorb a high number of idle cycles without affecting performance.

Thus, choosing the minimal memory speed is a function of several parameters including code region, problem size, and concurrency. To predict this *optimal* memory speed dynamically,

[2]Problem size = X refers to an input of X^3 elements in LULESH.

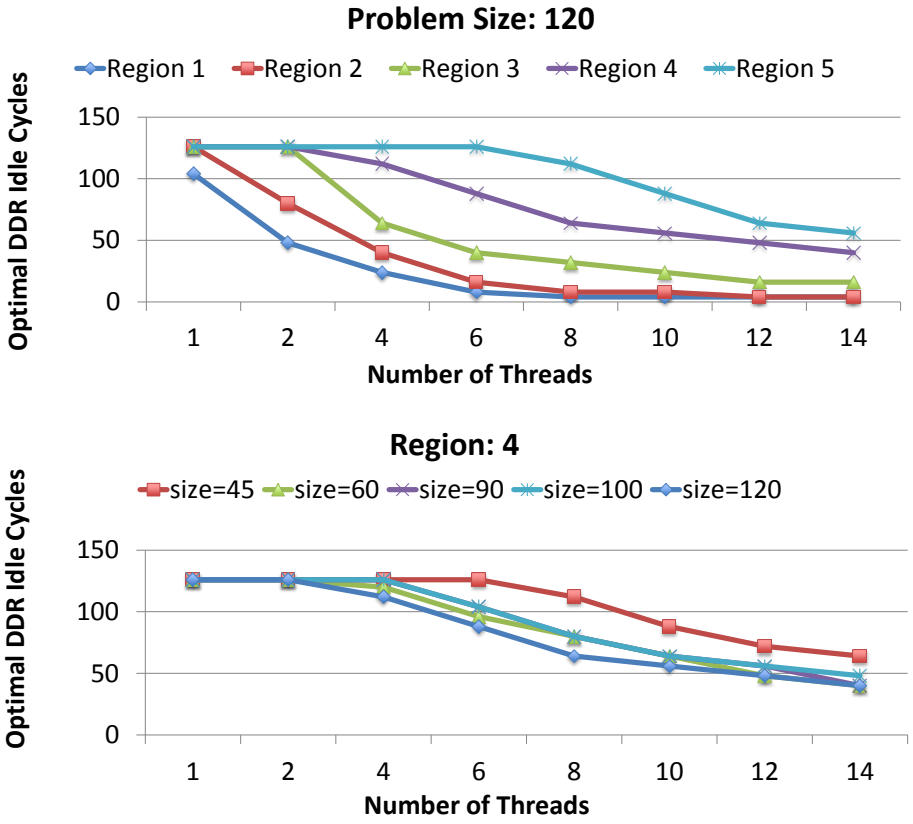


Figure 3.16: Optimal memory throttling as a function of concurrency for the LULESH regions and several problem sizes.

we build a model based on performance counters that captures the compute and memory requirements of LULESH.

Linear regression model

There are several hardware counters that can capture the computation and memory behavior of an application. The counters we chose include the number of CPU cycles, number of floating-point instructions, number of load and store commands that missed the L1 data cache, number of executed instructions, L2 cache misses, misses in the prefetch buffer, and

number of loads and stores to main memory.

These counters are classified as either CPU or memory related. The changes in problem size and level of concurrency can be observed through these counters. For example, increasing number of threads can cause an increase in memory bandwidth. We use these counters to predict the optimal (minimal) memory speed f_{min} for each region within an application. The model is defined as:

$$f_{min} = \sum_{i=1}^{N} w_i * \frac{c_i}{cyc} \tag{3.1}$$

where c_i is a hardware counter value, w_i is the coefficient achieved by offline training, and cyc is the total number of machine cycles executed. Since we consider multiple counters in this model, we use multivariate regression analysis to build the prediction model. Note that f_{min} corresponds to the number of idle cycles needed to throttle the memory.

To train the model (offline), we ran LULESH over a number of problem sizes, threads, and regions collecting more than 400 samples. We focused on problem sizes of 45 and higher, because smaller problem sizes result in regions (especially region 3) that are too small to quantify their power draw.

Figure 3.17 shows the observed optimal memory speed versus the one predicted from our model. Our model's accuracy is reasonable with an R^2 value of 0.67, although for some data samples the error is significant because the relationship between LULESH's execution time (characterized by the chosen performance counters) and memory speed is not necessarily linear. However, most predicted data points are close to the observed values.

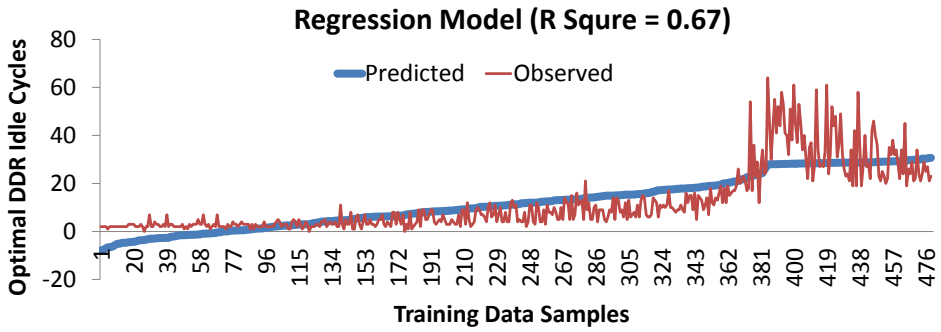


Figure 3.17: Accuracy of our linear regression model.

3.2.5 Impact on performance and power

In this section, we quantify the impact of memory throttling using our model on both performance and power. We show that the power consumption of LULESH can be reduced significantly with a marginal effect on performance. Since the granularity of our power and energy measurements is a node-board, we ran LULESH on 32 nodes: 1 process per node and 4 threads per core. We dedicated one core per node for a tracer thread that measures power. In addition, we only report the *dynamic* portion of power since we cannot change the amount of standby power. We expect that standby power will use a smaller fraction of the overall power in future memory technologies (e.g., PCM).

Effect on performance

When we trained the regression model, we used a threshold of 0.01. If our model is perfect, we should see no more than 1% performance loss compared to the *baseline* running at full memory speed. Our experiments consist of running LULESH with our regression model, which determines the number of idle cycles to inject on a per-region basis. In the first three iterations of each region (LULESH is an iterative code), we collect hardware counter data

that is fed to the model to predict the *optimal* number of idle cycles. We use this number to set the memory speed for the remaining iterations. In Figure 3.18, we show the performance overhead of our approach compared to the execution time of the baseline configuration.

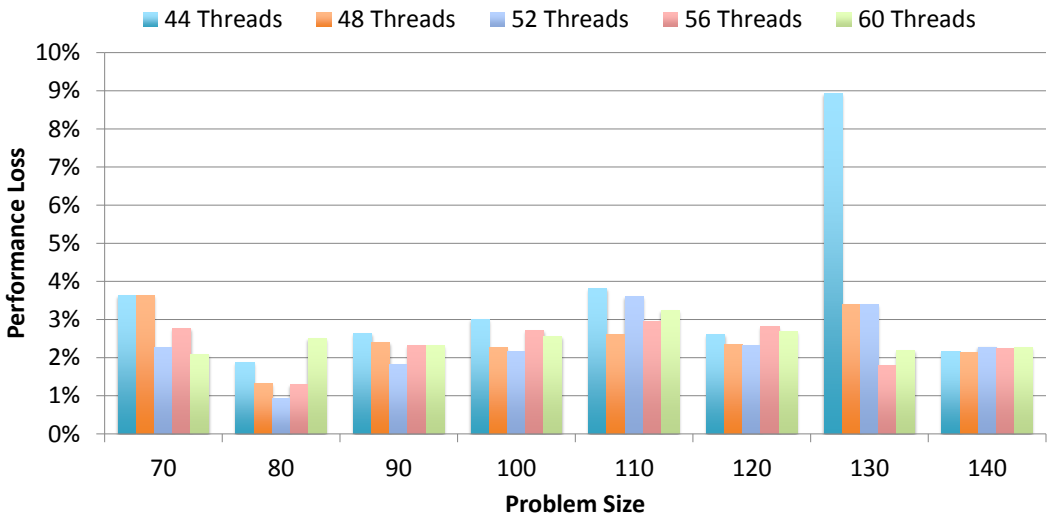


Figure 3.18: Runtime performance degradation by throttling memory according to our model. Each data point includes the aggregated overhead across all five regions.

The problem sizes range from 70 to 140 and the number of threads from 44 to 60. In general, most of our predictions with higher number of threads are within 2% error. The performance loss is close to our desired threshold value of 1%. Higher inaccuracies occur with the smaller number of threads, i.e. 44 threads, because the observed data points are not linearly distributed (see Figure 3.17). In future work we plan on using other models that can capture non-linear behavior. However, even this simple regression model provides a small amount of performance degradation for most cases.

Power and energy analysis

Our goal is to reduce as much power consumption as possible by lowering the memory speed until the performance loss reaches the given threshold. In this section, we first show that memory speed can significantly affect the power consumption of each LULESH region. Then, we demonstrate that both memory power and the total node-board power can be reduced based on our model predictions under various LULESH configurations. Finally, we discuss the tradeoffs between performance and power.

Figure 3.19 shows the power consumption of regions 1 and 4 under different memory speeds. We observe that total board power, CPU power, and memory power all drop significantly as we decrease memory speed. The difference in power may be as large as 300 Watts for the total board power. Although there is great potential to reduce power, we must consider performance (or energy) as well.

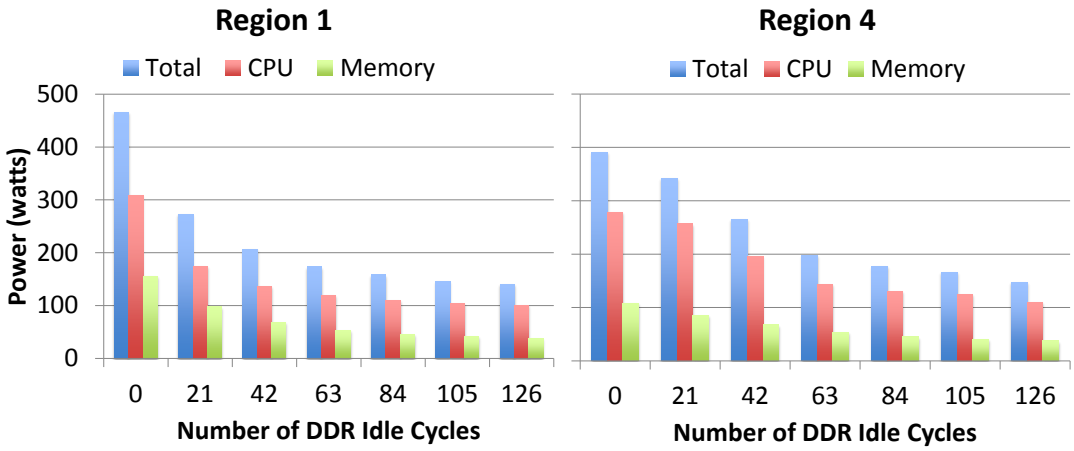


Figure 3.19: Memory throttling's effect on power for LULESH.

We now show what happens to the power consumption, along with performance, using our model-based prediction to throttle the memory of each region. Figure 3.20 shows how the performance, memory power, and total board power change with different number of threads

based on our model's memory speed prediction for a problem size of 130. This figure shows that on the 44 threads configuration, our model prediction causes a significant decrease in performance (less than 10%) with almost no power savings. The total energy consumption in this case is worse than the baseline. For the other four thread configurations, our model prediction leads to small performance loss (within 3%) and large memory and total power savings. The total energy consumption is significantly lower for these cases.

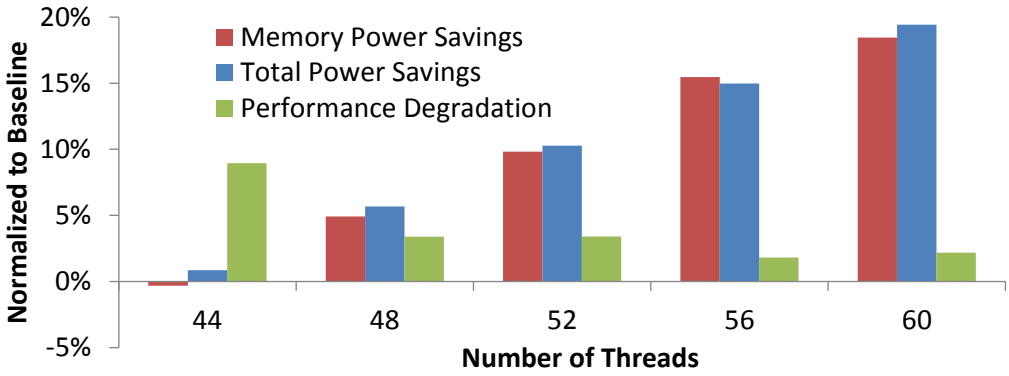


Figure 3.20: Performance and dynamic power tradeoffs with memory throttling using our regression model.

3.2.6 Summary

In this work, we investigate opportunities for shifting power between system components on a supercomputer architecture for explicit hydrodynamics codes. Because of the different computational characteristics within a single application, we employ memory throttling on a per region basis to save power on code regions with low memory bandwidth requirements. To find the optimal (lowest) memory speed for a given region without affecting performance, we use a linear regression model based on performance counters. Our results show that memory throttling can save up to 20% of dynamic power with an average performance loss of 3%. This indicates that power shifting on explicit hydrodynamics present significant opportunities to

improve performance within a fixed power budget. Future work includes quantifying power shifting opportunities on a range of HPC applications via memory throttling and employing machine learning techniques to capture non-linear behavior such as artificial neural networks.

Chapter 4

Parallel Performance Modeling

4.1 Compute–Overlap–Stall Model

4.1.1 COS Model Parameters

The Compute–Overlap–Stall (COS) model estimates parallel execution time as the sum of pure compute time (T_c), overlap time (T_o), and pure stall time (T_s). More generally,

$$T = T_c + T_o + T_s, \tag{4.1}$$

where T is total time for a running application.

Figure 4.1 shows an example execution time profile for a simple, single-threaded application. A single core executes some computation that triggers two separate, non-blocking memory operations. As the code executes, portions of time are spent exclusively on on-chip, in-cache computations; exclusively on off-chip memory operations; and on some form of overlap between computation and memory accesses.

Figure 4.1 provides context for defining terms of the COS model more precisely. T_c is the sum of the execution times of an application spent exclusively on computation[1] or the ***pure***

[1]We include the performance impact of on-chip caches in pure compute time. This simplification significantly reduces the complexity of the COS model while enabling isolation of the performance effects of power-performance operating modes.

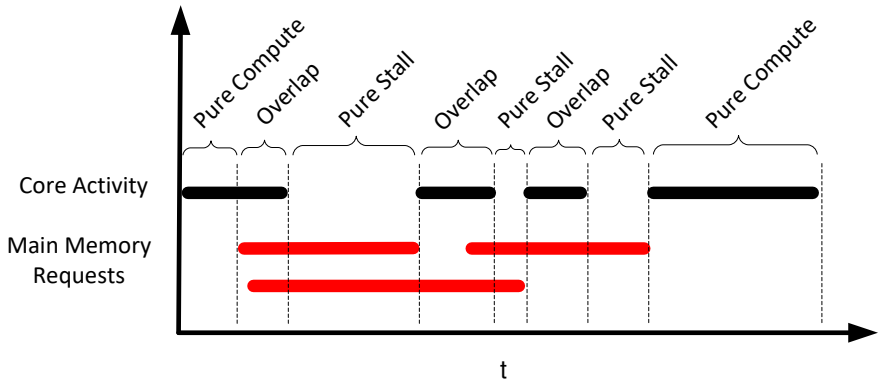


Figure 4.1: An example of a COS Trace for a simple, single-threaded application with hardware support for multiple, simultaneous memory accesses (e.g., multiple loads under misses).

compute time. In this example, T_c is the sum of the pure compute times identified at the start and end of the application's execution. T_o is the sum of the execution times of an application spent overlapping computation and memory operations or the ***overlap time***. In this example, T_o is the sum of the overlap times, there are three of these stage occurrences over the application's execution. T_s is the sum of the execution times of an application spent exclusively on memory stalls or the ***pure stall time***. In this example, T_s is the sum of the pure stall times, there are three of these stage occurrences over the application's execution.

4.1.2 The COS Trace

The ordered summation of the terms of the COS model constitutes a simplified trace of the application. We call this a ***COS Trace***. More precisely, the 8 stage occurrences for the

example in Figure 4.1 are expressed in the following COS Trace:

$$\begin{aligned} T = & T_c(1) + T_o(1) + T_s(1) + T_o(2) + T_s(2) + \\ & T_o(3) + T_s(3) + T_c(2) \end{aligned} \tag{4.2}$$

Analogously, we propose a general COS Trace as follows:

$$T = \sum_{i=1}^{cP} T_c(i) + \sum_{j=1}^{oP} T_o(j) + \sum_{k=1}^{sP} T_s(k) \tag{4.3}$$

where cP, oP, sP are the number of stages corresponding to the three types of time in the COS trace: the pure compute time (T_c), the overlap time (T_o), and the pure stall time (T_s). The index of i, j, k indicates the time order that these stages happen. For Figure 4.1, $cP = 2$, $oP = 3$, and $sP = 3$. Predicting parallel execution time using the COS model involves estimating the effect of a system or application change on the COS Trace[2].

4.1.3 COS Model Notations

In succeeding discussions we will use (f_c) and ($f_c^{'}$) to refer to a starting CPU frequency and the changed CPU frequency respectively. Moreover, Δf_c denotes the change from f_c to $f_c^{'}$. We can define Δf_m and Δt analogously. We use the shorthand $(f_c, f_m, t) \rightarrow (f_c^{'}, f_m^{'}, t^{'})$ to denote changes to DVFS, DMT/DBT, and thread count respectively. For example, (f_c, f_m, t) $\rightarrow (f_c^{'}, f_m, t)$ refers to an isolated change to CPU frequency while $(f_c, f_m, t) \rightarrow (f_c, f_m^{'}, t^{'})$ refers to simultaneous memory throttling and changes to thread counts.

[2]The power-performance operating modes studied include CPU Dynamic Voltage and Frequency Scaling (DVFS) and DRAM Dynamic Memory Frequency Throttling (DMT) on Intel architectures; Dynamic Memory Bandwidth Throttling (DBT) on BG/Q architectures; and Dynamic Concurrency Throttling (DCT) on both architectures.

4.1.4 The Importance of Isolating Overlap

Many existing models of parallel performance ignore overlap [14, 71, 117, 148, 168, 183]. When overlap *is* considered, the effects are either captured in the compute time (T_c) or memory stall time (T_s) parameters. If overlap is included in T_c, then the model assumes Δf_c effects apply equally to the overlap portion. If overlap is included in T_s, then the model assumes Δf_m effects apply equally to the overlap portion.

Figure 4.2 shows the stall time (y-axis) for a code region (R1) of the LULESH OpenMP application kernel [1]. The CPU voltage/frequency increases from left to right (x-axis). The figure shows the measured stall time and the predicted stall time for two best-available performance prediction approaches (stall- and leading-load-based [71, 117, 168]). Notice that both approaches consistently under-predict stall time. Furthermore, in another code region (R2) of the Lulesh OpenMP application (not shown), the same prediction techniques over-predict stall time.

When stall time dominates, these mis-predictions lead to significant inaccuracies in execution time prediction. The effects are exacerbated by the complex computation and memory overlap scenarios that affect stall and compute time and are more common in mixed operating modes (DVFS, DMT, and DCT).

4.1.5 The Challenge of Isolating Overlap

Figure 4.3 shows a simplified example for three CPU frequencies ($f_{c1} < f_{c2} < f_{c3}$) increasing from left to right. In each subfigure, core activity and memory activity are shown separately as a thread progresses in time (x-axis) from left to right. The COS trace is provided for each subfigure.

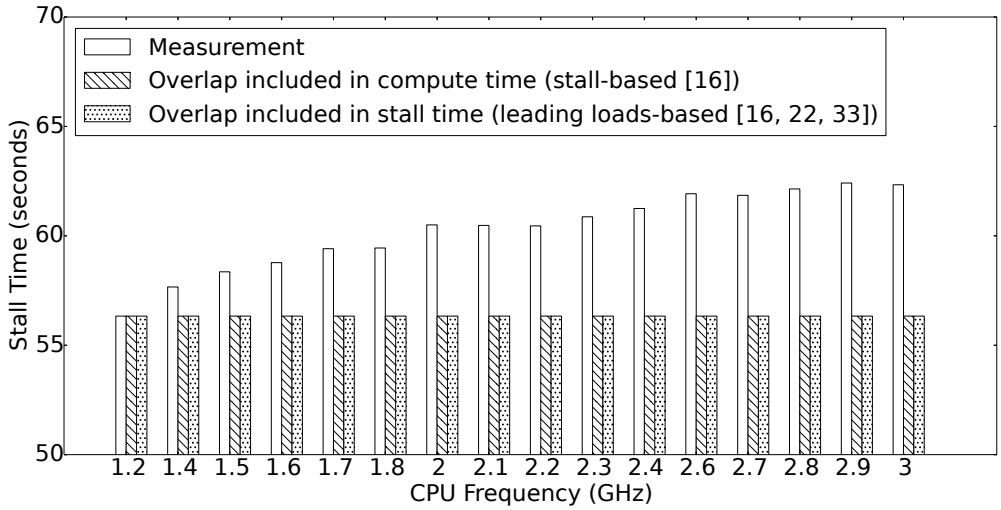


Figure 4.2: Stall time (y-axis) for varying CPU voltage/frequency settings (x-axis) for the LULESH benchmark on an x86 system. When stall time dominates, these mispredictions lead to significant inaccuracies in execution time prediction. The effects are exacerbated by the complex computation and memory overlap scenarios that affect stall and compute time and are more common in mixed operating modes (DVFS, DMT, and DCT).

In the first subfigure, at the lowest CPU frequency f_{c1}, there are 4 distinct compute and memory overlap phases in the COS trace. This indicates regular memory accesses where the CPU is busy with work during the memory stall time. More precisely, the 9 stage occurrences for f_{c1} in Figure 4.3 are expressed in the following COS Trace:

$$\begin{aligned} T = & T_c(1) + T_o(1) + T_c(2) + T_o(2) + T_c(3) + T_o(3) + \\ & T_c(4) + T_o(4) + T_c(5) \end{aligned} \tag{4.4}$$

The change from $(f_c, f_m, t) \rightarrow (f_c^{'}, f_m, t)$ alters the COS Trace (f_{c2} in Figure 4.3) as follows:

$$T = T_c(1) + T_o(1) + T_c(2) \tag{4.5}$$

This reflects a dependency between the resulting COS trace and CPU frequency. In this case, there is a change in the arrival rate of the memory requests due to the CPU fre-

quency changes. In the new configuration, there are no pure compute gaps between memory references leading to a change in the number and length of overlap stages.

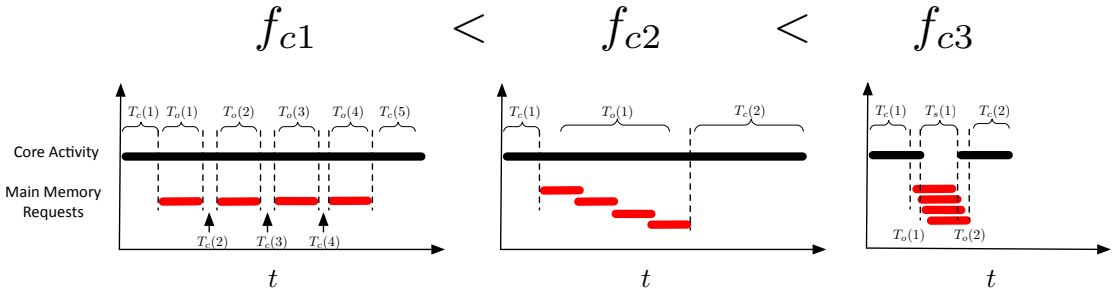


Figure 4.3: Overlap time and pure stall time are related to computation intensity.

Increasing the frequency a second time in this example (f_{c3} in Figure 4.3) alters the COS trace again, resulting in:

$$T = T_c(1) + T_o(1) + T_s(1) + T_o(2) + T_c(2) \tag{4.6}$$

This demonstrates the creation of a pure stall stage that did not exist in the previous two COS traces (f_{c1} and f_{c2}) in Figure 4.3. We observe similar behaviors for memory throttling changes ($(f_c, f_m, t) \rightarrow (f_c, f_m^{'}, t)$) and for thread changes ($(f_c, f_m, t) \rightarrow (f_c,\ f_m, t^{'})$).

4.1.6 The Role of Computational Intensity

Estimating the COS terms for simultaneous changes in operating modes such as $(f_c, f_m, t) \rightarrow (f_c^{'}, f_m^{'}, t^{'})$ is even more challenging than the single CPU speed change described in Section 4.1.5. In theory, each term of the COS trace is affected by all operating mode changes (Δf_c, Δf_m, and Δt). In practice, it depends on the system and application design.

The initial focus of our work is on shared memory systems running multithreaded OpenMP applications where parallel threads are mostly homogeneous and synchronized and the pro-

grams use a bulk-synchronous programming model. This focus leads to some simplifying assumptions while still covering a large set of parallel applications of interest to a broad community of scientists [15, 23, 70, 115, 123, 129, 165].

Table 4.1 shows the application of these assumptions to reduce the set of interactions we need to consider for accurate predictions between system reconfigurations and model parameters. For example, to model $T_c^{'}$ for 7 rows of possible configurations, we need only consider changes to CPU frequency (Δf_c) and thread count (Δt). For $T_o^{'}$ and $T_s^{'}$, these assumptions simplify all predictions except when all operating modes change simultaneously $(f_c, f_m, t) \rightarrow (f_c^{'}, f_m^{'}, t^{'})$ for any Δf_c, Δf_m, and Δt. This explains why our prediction methods on real systems focus on separating overlap time $T_o^{'}$ from stall time $T_s^{'}$ (see Sections 4.1.4 and 4.1.5).

Table 4.1: Effects on COS model parameters of any starting configuration (f_c, f_m, t) to any other operating mode configuration (each row) for changes in processor speed, memory throttling, and number of threads (Δf_c, Δf_m, and Δt). For some configurations, we have additionally identified CI (Computational Intensity) as having significant influence over COS model parameters.

Config	$T_c^{'}$	$T_o^{'}$	$T_s^{'}$
$f_c^{'}, f_m, t$	Δf_c	Δf_c CI	Δf_c CI
$f_c, f_m^{'}, t$		Δf_m CI	Δf_m CI
$f_c, f_m, t^{'}$	Δt	CI Δt	CI Δt
$f_c^{'}, f_m^{'}, t$	Δf_c	Δf_c Δf_m CI	Δf_c Δf_m CI
$f_c, f_m^{'}, t^{'}$	Δt	Δf_m CI Δt	Δf_m CI Δt
$f_c^{'}, f_m, t^{'}$	Δf_c Δt	Δf_c CI Δt	Δf_c CI Δt
$f_c^{'}, f_m^{'}, t^{'}$	Δf_c Δt	Δf_c Δf_m CI Δt	Δf_c Δf_m CI Δt

In addition to the system configuration changes (Δf_c, Δf_m, and Δt), Table 4.1 lists CI as a consideration for both overlap $T_o^{'}$ and pure stall $T_s^{'}$ times. CI here stands for *Computational Intensity*, or the percentage of memory stall time that is overlapped with useful work on the CPU. CI determines how much stall time is affected by CPU speed (Δf_c) and how much is affected by memory throttling (Δf_m).

We conducted statistical analyses to identify a correlation between stall time and measurable

hardware counters available on most x86 architectures [4]. Through exhaustive experimentation for all available configurations $(f_c, f_m, t) \rightarrow (f_c^{'}, f_m^{'}, t^{'})$, we found that two widely available counters—last-level cache misses (LLCM) and time-per-instruction (TPI)— effectively captured the stall time effects of Δf_c, Δf_m, and Δt. This finding is key to the COS model's effectiveness since it enables us to use linear approximation methods to separate pure stall time from overlap stall time. For completeness, we studied the effects of CI on compute overlap but we found that compute time was dominated by effects from CPU speed and thread count and not affected significantly by CI.

4.1.7 Practical Estimation of COS Parameters

We can use the COS trace of Equation 4.3 to predict the parallel execution time (T') of another system configuration for any combination of Δf_c, Δf_m and Δt. Since the variables may not be directly measurable, the challenge is to collect accurate approximations without requiring system design changes or reverting to simulation. In this section we describe one method for predicting T' using direct measurements readily available on most x86 systems.

Several of the parameters of Equation 4.3 are directly measurable. Both total time T and the *pure stall time* T_s are directly measurable using the CPU hardware counters available on most modern platforms [4].

We've also observed in our experimental work that overlap consists of a portion affected by CPU speed changes (related to compute time and denoted as T_{oc}) and another portion affected by memory throttling (related to stall time and denoted as T_{os}). The portions vary according to the computational intensity CI of the application (see Section 4.1.6).

Under these measurements and observations, the operation mode change $(f_c, f_m, t) \rightarrow (f_c^{'}, f_m^{'}, t^{'})$ resulting in predicted time $T^{'}$ becomes:

$$T^{'} = [T_c^{'} + T_{oc}^{'}] + [T_{os}^{'} + T_s^{'}] \tag{4.7}$$

where $[T_c^{'} + T_{oc}^{'}]$ can be approximated as $[T$ - $T_s] \times f_c/f_c^{'}$. Multiplying by the ratio of the CPU speed f_c and the new CPU speed $f_c^{'}$ follows the dependencies $(\Delta f_c, \Delta t)$ listed in Table 4.1 for the $(f_c, f_m, t) \rightarrow (f_c^{'}, f_m^{'}, t^{'})$ configuration for $T_c^{'}$. We will discuss how thread changes affect predictions in Section 4.1.8.

Approximating $[T_{os}^{'} + T_s^{'}]$ is more difficult. Table 4.1 shows that time for the (f_c, f_m, t) $\rightarrow (f_c^{'}, f_m^{'}, t^{'})$ configuration for $T_s^{'}$ is affected by a combination of Δf_c, Δf_m, Δt, and IC. Ignoring the impact of multi-threading again for now (see Section 4.1.8), we propose a linear combination of direct measurements for $LLCM$ and TPI with direct observations of changes to CPU Speed and memory throttling ($f_c^{'}$ and $f_m^{'}$). Recall the $LLCM$ and TPI terms capture the Computational Intensity CI effects on the COS trace. This gives the following approximation for the remaining portion of Equation 4.7:

$$[T_{os}^{'} + T_s^{'}] = \alpha_1 \times LLCM + \alpha_2 \times TPI + \alpha_3 \times f_c^{'} + \alpha_4 \times f_m^{'} \tag{4.8}$$

Combining our approximations for both sets of terms in Equation 4.7, our approximation of $T^{'}$ for a operating mode configuration change $(f_c, f_m, t) \rightarrow (f_c^{'}, f_m^{'}, t^{'})$ is:

$$T^{'} = [T - T_s] \times \frac{f_c}{f_c^{'}} + \alpha_1 \times LLCM + \alpha_2 \times TPI + \alpha_3 \times f_c^{'} + \alpha_4 \times f_m^{'} \tag{4.9}$$

In the next section, we describe how we use training sets and linear regression to identify the alpha parameters in this equation to develop a general model for each application in our set of 19.

4.1.8 Offline Training and Online Prediction

We use a training set measured offline to predict online a larger set of Δf_c, Δf_m, and Δt configurations. Figure 4.4 illustrates this two step process.

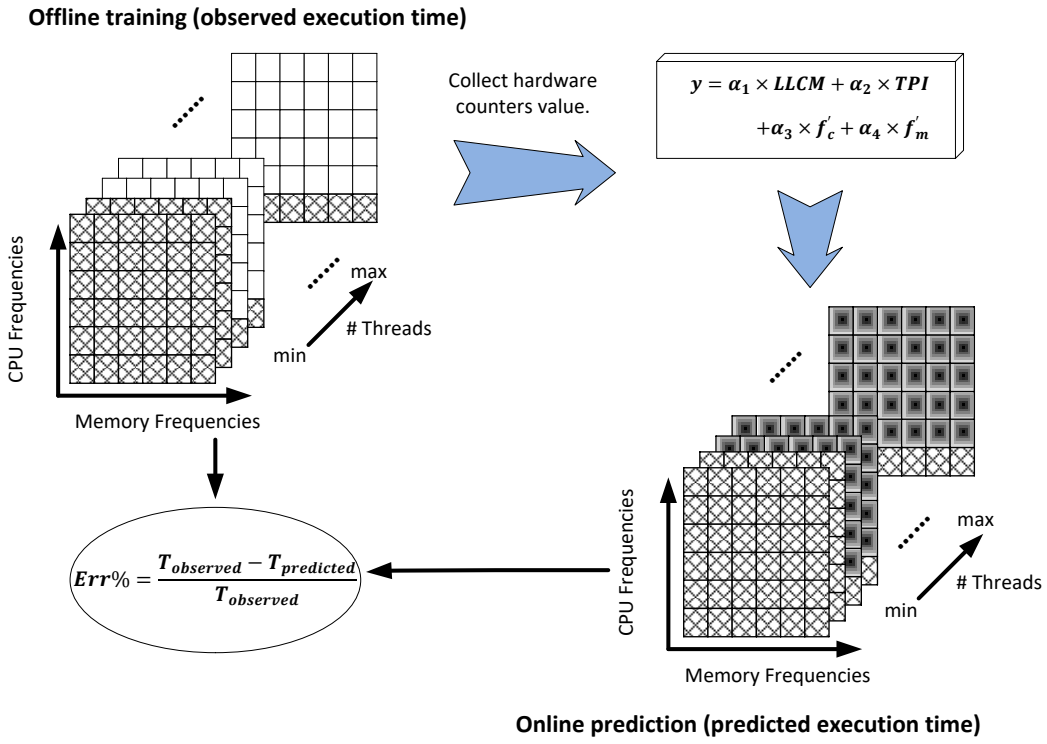


Figure 4.4: Offline training and online prediction.

The astute reader will notice that Equation 4.9 contains no term for the number of threads despite our claim to predict for dynamic concurrency changes. The impact of threads is captured in a set of linear approximations for Equation 4.9 applied to our training sets.

What follows is an explanation of the algorithm we use to predict the simultaneous effects of Δf_c, Δf_m, and Δt configurations.

Figure 4.4 and Algorithm 1 describe our sampling techniques in detail. Basically we gather a set of data for a given application and take samples at various configurations for Δf_c, Δf_m, and Δt. We use this data to conduct linear regression on Equation 4.9 to determine the values of the four α parameters. For each measurement, we simultaneously gather execution time (T), stall time (T_s), $LLCM$ values, and TPI values.

We designed Algorithm 1 to formally describe the process illustrated by Figure 4.4. We define f_c^{min}, f_m^{min}, and t^{min} as the minimum speed setting for CPU, minimum throttling setting for memory, and the smallest number of threads respectively for a training set.

Algorithm 1 Train the COS Model for any application

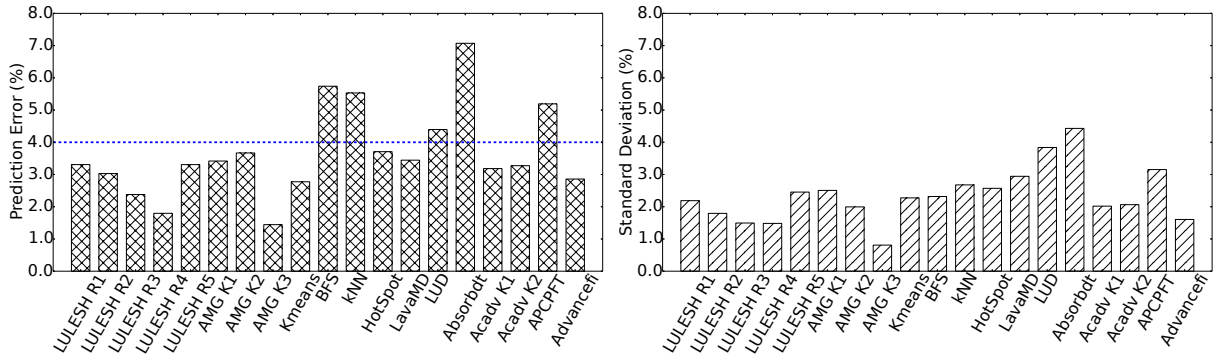
1: **for all** $f_m^{'} \neq f_m^{min}$ **do**
2: Measure $T, T_s, LLCM, TPI$ for $(f_c^{min}, f_m^{min}, t^{min}) \rightarrow (f_c^{'}, f_m^{'}, t^{'})\ \forall\ f_c^{'} \neq f_c^{min}$ and $t^{'} = 4, 6$
3: Measure $T, T_s, LLCM, TPI$ for $(f_c^{min}, f_m^{min}, t^{min}) \rightarrow (f_c^{min}, f_m^{'}, t^{'})\ \forall\ t^{'} \neq t^{min}$
4: **end for**
5: Use measured data and linear regression to find α coefficients for Equation 4.9
6: Use Equation 4.9 to predict any $(f_c, f_m, t) \rightarrow (f_c^{'}, f_m^{'}, t^{'})\ \forall\ f_c, f_m, t$ and $\forall\ f_c^{'}, f_m^{'}, t^{'}$ for this application

In Algorithm 1, thread behavior is captured by the training set. Basically, by reapplying Equation 4.9 to different thread configurations (steps 2 and 3 in Algorithm 1) we are able to capture the effects of threads on the COS model parameters using a combination of direct measurements and linear regression. These effects are incorporated in both $[T - T_s]$ and the $LLCM$ and TPI terms of Equation 4.9. Thread effects are implicitly captured in the algorithmic application of Equation 4.9 and thus not explicitly in the formulation.

For a memory modes, b CPU modes, and c thread settings, we require $a \times b \times 2$ measurements for step 2 in Algorithm 1 and $a \times c$ measurements for step 3 in Algorithm 1. These

measurements are captured visually by the hashed squares on the left side of Figure 4.4. This is compared to our ability to predict $a \times b \times c$ combinations using a single training set (see the darker squares on the right side of Figure 4.4). We have also determined that of the 19 applications studied, only 4-6 models are needed to accurately predict T' for all 19 applications. In future work, we are attempting to reduce the training sets further for online usage. In the remainder of this paper, we compare our predictions with direct measurements and use the resulting COS model for analysis for 19 applications on Intel x86 and IBM BG/Q systems.

4.2 Empirical Model Validation



(a) Average prediction error. (b) Standard deviation of the prediction error.

Figure 4.5: Model prediction accuracy for a wide-range of codes.

In this section, we validate the COS model on a multi-core machine using several application benchmarks with different computational characteristics. We measure the accuracy of the model by comparing the model's prediction versus observed values measured on real hardware.

4.2.1 Machine Characteristics

We validate the COS model on a cluster comprised of Dell PowerEdge R430 servers. Each node has two Intel Xeon E5-2623 v3 (Haswell) processors and 32 GB of DDR4 memory. Each processor has four cores and each core supports two hardware threads. The Haswell processor supports 16 CPU frequencies ranging from 1.2 to 3.0 GHz. The memory system supports three bus frequencies: 1.333, 1.600, and 1.866 GHz.

4.2.2 Application Benchmarks

We employ a set of benchmarks and kernels that represent diverse computational characteristics appearing in high-performance, parallel, scientific applications. The application benchmarks include the following codes:

- LULESH (CORAL benchmark suite[3], 5 code regions)
- AMGmk (CORAL benchmark suite, 3 kernels)
- Rodinia benchmark suite (6 applications)
- pF3D from LLNL (5 kernels)

LULESH is an explicit hydrodynamics proxy application that contains data access patterns and computational characteristics of larger hydrodynamics codes at LLNL [1]. We use five code regions within an OpenMP version of LULESH that represent different phases of the application and consume over 90% of the runtime [128]. These five code regions (R1 to R5) were selected in collaboration with domain scientists to isolate the code regions with a diverse set of computational intensity characteristics.

[3]See https://asc.llnl.gov/CORAL-benchmarks

AMGmk includes three compute intensive kernels from AMG, an algebraic multigrid benchmark application derived directly from the BoomerAMG solver in the Hypre linear solvers library [97]. This code is used broadly in a number of applications [129] of interest to the multi-physics community. The default Laplace-type problem is built from an unstructured grid with various jumps and anisotropy in one part. We label these kernels K1 to K3.

Rodinia is a benchmark suite for heterogeneous computing [40]. We use six OpenMP codes from the domains of data mining, graph algorithms, physics simulation, molecular dynamics, and linear algebra: Kmeans, k-Nearest Neighbors (kNN), Breadth-First Search (BFS), HotSpot, LavaMD, and LU Decomposition (LUD). Components of this application suite such as HotSpot are of high interest to domain scientists for use in structured grid applications [15, 23, 115]. There is also high demand for optimized linear algebra solvers [165, 190] such as kNN, Kmeans, and LUD that are used regularly in many high-performance applications and systems.

pF3D is a massively parallel application that simulates laser-plasma interactions at the National Ignition Facility at LLNL [122]. This simulator aids scientists in tuning plasma and laser beam experiments crucial to experimental physics [123]. The pF3D kernels derive from the functions that consume the most time during a typical pF3D run and are written in OpenMP. We use the following kernels: Absorbdt, Acadv K1, Acadv K2, APCPFT, and Advancefi.

In total we used $5 + 3 + 6 + 5 = 19$ code regions and application kernels to evaluate the proposed model. For simplicity, we refer to these as *codes* or *applications* although they are *application benchmarks*.

4.2.3 Performance Prediction Accuracy

We compare the execution time predicted using modeling with the execution time observed by running the codes. First, for each code, we train its model offline (see Section 4.1.8) using a sample of Δf_c, Δf_m, and Δt as shown in Table 4.2. With these configurations we derive the model coefficients. The selection of thread numbers in the training set enables extrapolation-based performance prediction. At this point, we can use the model to predict the execution time of any given configuration. Second, we run the code under the configurations not in the training set for a total of 225 configurations. Each of these is run 20 times to smooth out system noise effects and the average execution time is calculated. Third, we do this for all 19 codes.

Table 4.2: We use 4- and 6-thread configurations to predict 8, 10, 12, 14, and 16 thread configurations where Δf_c=16 modes, Δf_m=3 modes, Δt=7 thread configurations.

Total	Δf_c (GHz)	Δf_m	Δt (num. threads)
Training configurations			
$16 \times 3 \times 2$	All	All	4, 6
$1 \times 3 \times 5$	1.2	All	8, 10, 12, 14, 16
Configuration space			
$16 \times 3 \times 7$	All	All	4, 6, 8, 10, 12, 14, 16

We define the prediction error of performance as follows (also shown in Figure 4.4):

$$Err\% = \frac{|T_{measure} - T_{predict}|}{T_{measure}}$$

Figure 4.5 shows the model prediction accuracy for all of the codes. Figure 4.5a shows the average prediction error of each code across the entire configuration space not in the training set. Figure 4.5 shows that the average prediction error per code is significantly low: varying from 1.4% to no more than 7%. Most of the codes though have an error lower than 4%. This

demonstrates the proposed model is highly accurate for a broad range of applications. We also measure the standard deviation of the prediction error as shown in Figure 4.5b. The standard deviations for all the codes is within 4.5%. Our proposed model is significantly accurate for the three dimensional configuration space for all 19 applications.

To verify that the tested codes include a wide range of different computational characteristics, we measured the *sensitivity* of a subset of our codes to certain parameters such as processor speed. To capture an application's sensitivity, we focus on *pressure to the memory system* measured as last level cache misses per second. We expect, for example, low memory pressure for compute-intense applications (see Section 4.1.6) and high pressure for memory bandwidth-intense applications.

Figure 4.6 shows last-level cache misses (LLCM) per second as a function of different processor and memory speeds and thread concurrency. We employ two processor speeds (1.2 and 3.0 GHz), two memory speeds (1.333 and 1.866 GHz), and two thread counts (4 and 16). Each configuration is represented as a tuple of the following form:

$$(\text{C: } cpu\_frequency, \text{M: } memory\_frequency, \text{T: } num\_threads)$$

First, we focus on one configuration: (C1.2, M1.333, T4). Codes including kNN, AMG K1 and K2, and LULESH R4 show low memory bandwidth presssure. This matches our expectation since AMG K1 and K2 are compute-intense kernels as is LULESH R4 [130]. While LULESH R1 and R3 are among the ones with the highest usage, Kmeans, BFS, and LULESH R2 exercise higher memory bandwidth utilization. These last two have been shown to be memory bandwidth intensive [128].

Second, we observe that some codes are significantly affected by different parameters such as memory speed and processor speed. LULESH R1 for example shows increased memory

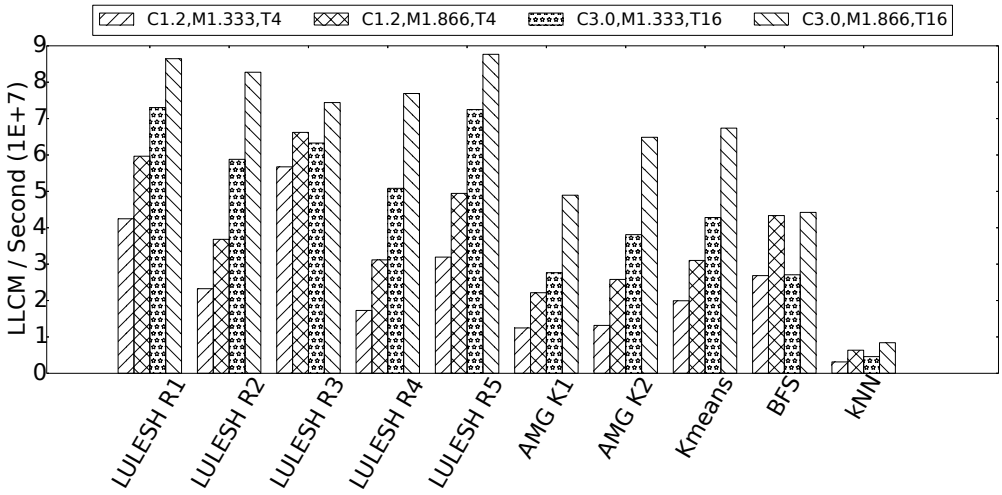
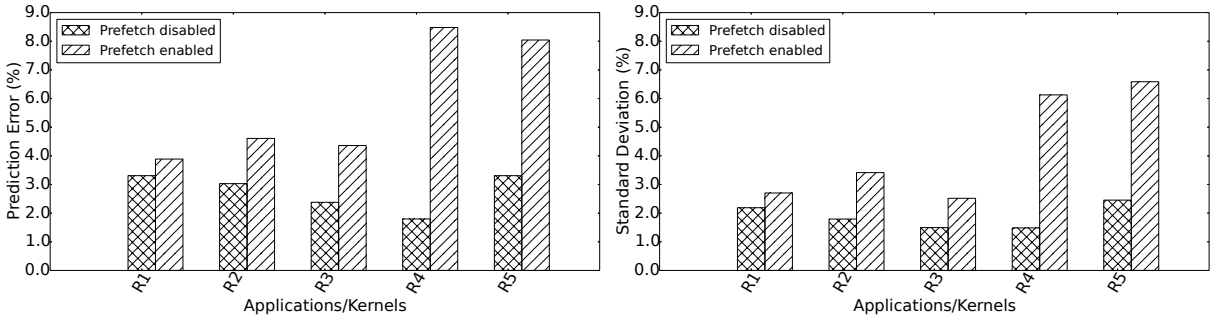


Figure 4.6: Impact of configuration on memory pressure.



(a) Average absolute prediction error.

(b) Standard deviation of the absolute prediction error.

Figure 4.7: Impact of prefetching on prediction accuracy running LULESH.

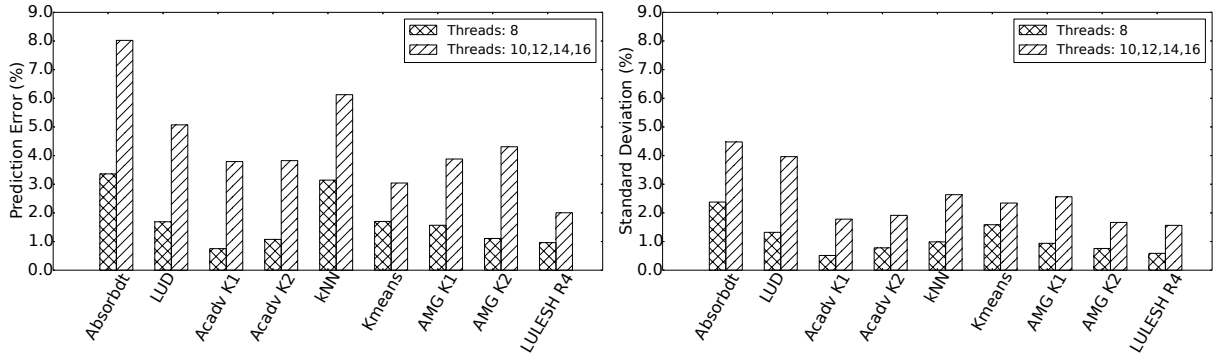
pressure with increases in processor speed and also with increases in memory speed. R1 has a high number of instructions per cycle (IPC) that benefit from the increased processor speed shifting the pressure to the memory system. Except for R3, other regions of LULESH show a similar pattern but at different scales. Kmeans, AMG K1, and AMG K2 show significant sensitivity to processor speed because of their linear algebra computations.

Third, there are codes that show low sensitivity to different configurations. kNN is a clear example of this. BFS is not affected by increases in processor performance since there is little computation during the graph traversals but does show sensitivity to memory performance

as a result of the operations fetching undiscovered graph nodes from memory. LULESH R3 is an interesting case since there are small changes with either processor or memory speed. This is the result of the code being almost exclusively memory bandwidth bound.

Thus, Figure 4.6 shows that the codes studied in this work capture a diverse set of computational characteristics. Furthermore, the resources in the critical path for some of these codes can change significantly with different configurations. For example, AMG K2 run with a 3.0 GHz processor becomes significantly dependent on the memory system when increasing memory frequency from 1.333 to 1.866 GHz. The low prediction error of the proposed COS model shows that we can capture the effect of these complex interactions accurately.

4.2.4 COS on Intel Sensitivity Analysis



(a) Average absolute prediction error.

(b) Standard deviation of the absolute prediction error.

Figure 4.8: Impact of ROB and MSHR on prediction accuracy.

Memory Prefetching

During development, we noticed the COS model accuracy was sensitive to prefetch settings. The effects of hardware and software prefetching on performance are captured in changes to the COS Trace described by Equation 4.3. For example, a successful prefetch could increase

overlap by preemptively importing data from main memory to cache. An incorrect prefetch however causes cache pollution and could lead to more overlap stages and stall stages.

To better understand these effects, we ran LULESH with hardware prefetching enabled and hardware prefetching disabled and analyzed the results using COS. Figure 4.7 shows these results using 4 Intel prefetchers: DCU streamer prefetcher (load data to L1 data cache triggered by an ascending access of recently loaded data), DCU IP prefetcher (load data to L1 data cache based on load instruction and its detected regular stride), adjacent line prefetcher (fetch cache line to L2 and last level cache with the pair line), and hardware (streamer) prefetcher (fetch cache lines to L2 and last level cache based on detection of forward or backward stream of requests from L1).

After enabling all the hardware prefetchers, the accuracy of our predictor worsens as expected. For LULESH R1, the change in average prediction error (Figure 4.7a) and standard deviation (Figure 4.7b) are both very minimal. In contrast, LULESH R4 has the largest differential (4x) in accuracy when prefetching is enabled. This is likely due to a large increase in overlap when prefetching is enabled since the R4 region is dominated by compute when overlap is disabled.

When prefetching is disabled, we get excellent accuracy using an extrapolation technique to predict configurations not observed directly in the training set. To improve the accuracy of COS for prefetching, we switched to an interpolation technique using 4- and 16-thread configurations to predict 6-, 8-, 10-, and 12-thread configurations.

The results validate that the COS model based predictor can successfully capture the impact of DVFS, DMT, and DCT simultaneously with prefetching but at the expense of predictor flexibility. The COS approximation techniques implemented in the Intel systems could be extended to better capture the effects of prefetching overlap on performance using approaches

similar to those used for power-performance modes.

ROB and MSHR

The sizes of the reorder buffer (ROB) and miss status holding register (MSHR) increase with each generation in CPU design. The ROB reorders instructions to increase instruction-level parallelism and the MSHR increases the number of loads that can be handled under a previous miss. These techniques have the potential to increase overlap and can impact the accuracy of the COS predictor on Intel Systems

We picked nine applications to ascertain the sensitivity of COS to the ROB and MSHR. The Intel hyperthreading design enables us to indirectly control the size of the ROB and MSHR. For a single thread per core, the ROB and MSHR are fixed in size. However, if we overload a core with multiple threads, the ROB and MSHR resources are divided among the threads. We exploit this indirect control in our experiments.

In our experimental setup, we identify two basic configurations: 1) 4-, 6-, and 8-threads where at most one OpenMP thread is mapped to a core, and 2) 10-, 12-, 14-, and 16-threads where at least two cores run two OpenMP threads. As mentioned, these Intel machines have 8 cores each with two hardware threads per core.

For these experiments we disable prefetching and use our extrapolation approach with 4- and 6-threads for training. Figure 4.8 shows that both average error and the standard deviation for 8 threads are much better than all the other configurations of threads.

There appears to be a correlation between the least accurate of our earlier experiments, ABSORBDT (Figure 4.5), and the ROB and MSHR results. Though further experimentation is needed, it is likely that ABSORBDT is sensitive to ROB and MSHR sizes and we could consider improvements to our approximations of the COS model that incorporate these

characteristics.

4.3 Cross-architecture validation using IBM's Blue Gene/Q

To demonstrate the portability and scalability of our model we validate COS on IBM's Blue Gene/Q (BG/Q) architecture. BG/Q is a scalable, energy efficient, high-performance system. The BG/Q architecture is capable of dynamic memory bandwidth throttling (DBT), where memory bandwidth is dynamically controlled through insertion of a configurable number of memory idle cycles between each DDR memory request.

BG/Q's DBT is different from the dynamic memory frequency throttling (DMT) common to Intel systems. While memory frequency throttling changes the latency of each main memory access, bandwidth throttling reduces the effective bandwidth through inserting idle cycles (or no-ops or bubbles) in the instruction pipeline. The number of memory idle cycles inserted is called the *throttling threshold* and ranges between 0 and 126. Studies have shown this parameter can affect the performance of applications as well as their power consumption [133].

The throttling threshold affects those memory accesses that occur within the threshold window. For instance, if the time between two dependent memory requests at the memory controller is larger than the throttling threshold, the latency of these memory requests is not affected. When the time between two memory requests is smaller than the threshold, the latency of the second memory request would increase by the configurable number of memory idle cycles.

Unlike the Intel system, BG/Q is not capable of CPU frequency scaling and thus we limit our validation of COS to variations in memory bandwidth and thread concurrency. BG/Q has only two levels of cache, L1 and L2, compared to 3 levels of cache on our x86 experimental

system.

4.3.1 Approximating the COS Trace

Assume we change memory speed from f_m to $f_m^{'}$ (Δf_m) using DBT. To illustrate the effect on performance, Figure 4.9 shows the execution time in cycles (y-axis) for different throttling thresholds represented by number of memory idle cycles (x-axis) for all regions of the LULESH application. For each region (R1, R2, R3, R4, R5) of LULESH, two different phases can be distinguished: 1) a nearly flat or constant segment in the function at low thresholds and 2) a linearly increasing function at a threshold that appears to be different for each region. This forms a hockey stick shaped function for each region with a different inflection point. In a way, this is an example of Amdahl's law applied to an architectural enhancement. A portion of the code (phase 1 in this example) *is not affected* by the enhancement (e.g., insertion of memory idle cycles) while a portion of the code (phase 2 in this example) *is affected* by the enhancement. While this is an oversimplification in some ways, it implies that we can potentially use a piece-wise function to approximate the performance for these codes if we can identify the inflection point (i.e., the number of memory idle cycles) where performance loss begins.

Following a series of experiments, we determined the inflection points correlate to characteristics of a region's memory access behavior. We approximate the COS Trace expressed by Equation 4.3 using a piecewise function of performance:

$$T = \begin{cases} t_0 & \text{if } f_m \leq a \\ bf_m + t_0 & \text{if } f_m > a \end{cases} \tag{4.10}$$

where t_0 is the performance with no memory throttling, a is the threshold of the inflection

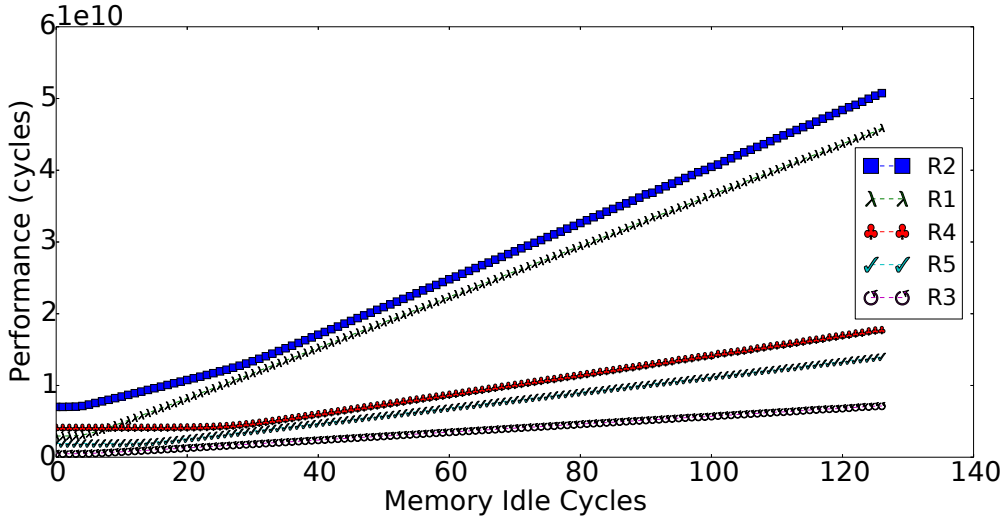
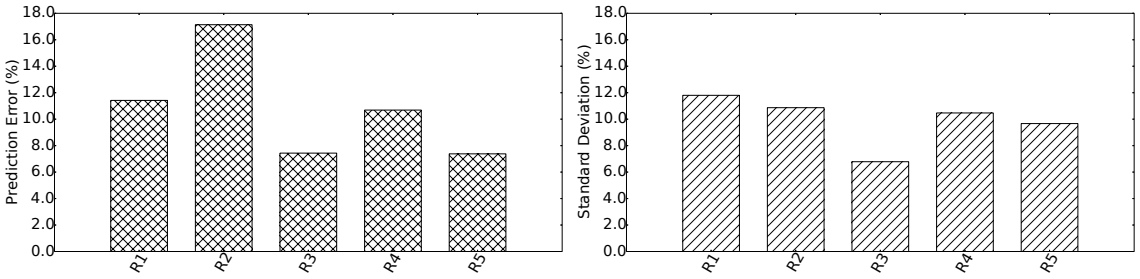


Figure 4.9: Impact of memory bandwidth throttling on LULESH.



(a) Average prediction error.

(b) Standard deviation of the prediction error.

Figure 4.10: Multivariate linear regression of LULESH on IBM BG/Q system.

point, and b is the slope of the linear function.

For the constant function ($T = t_0$), memory throttling has little impact on the COS Trace: performance does not change by inserting memory idle cycles. This can be explained with the following two cases. First, the gap between most of the application memory accesses is larger than the throttling threshold. The number of inserted memory idle cycles is too small to cause delays in memory accesses (T_s is not changing) and thus total execution time. In this case, inserting idle cycles does not change T_c, T_o, and T_s. Second, the gap between memory accesses is smaller than the throttling threshold, but the memory accesses overlap

with processor computation. Inserting idle cycles can delay issuing new memory requests but does not change the length of any of the three stages, T_c, T_o, and T_s.

The inflection point a in Equation 4.10 depends on the memory access patterns of applications. A correlation analysis among some critical compute/memory related hardware events (e.g. floating point operations, L2 cache misses per second, etc.) shows that its value is highly related to memory intensity: *L2 cache misses (L2M) per instruction (INST)*. By applying linear regression, we can approximate the value of a with the following:

$$a = \alpha \times \frac{L2M}{INST} + c_1$$

where α is a coefficient and c_1 is a constant and both will be determined using linear regression.

The impact of memory throttling on the second segment is linear ($T = bf_m + t_0$). This can be explained with the COS Trace as follows. For a sufficiently large number of idle cycles, application memory accesses cannot overlap with computation. In this case, the length of the pure compute stages would not change with memory throttling; the length of the overlap stages would be zero; and the length of the pure stall stages would change linearly with the number of memory idle cycles inserted. Approximating the number of memory accesses with L2 misses, the impact on the COS Trace can be expressed as follows:

$$\Delta T_c = 0$$

$$\Delta T_o = 0$$

$$\Delta T_s = \beta \times L2M \times \Delta f_m$$

where ΔT_c, ΔT_o, and ΔT_s are the resulting change to execution time for each respective

phase. Thus, we can approximate the value of b as follows:

$$b = \beta \times L2M + c_2$$

where β is a coefficient and c_2 is a constant and both will be determined using linear regression.

Based on the equations above, we can predict performance using Equation 4.10 as follows:

$$T = \begin{cases} t_0 & \text{if } f_m \leq \alpha \times L2M/INST + c_1 \\ (\beta \times L2M + c_2) \times f_m + t_0 & \text{if } f_m > \alpha \times L2M/INST + c_1 \end{cases} \tag{4.11}$$

4.3.2 Offline Training and Online Prediction

We apply linear regression to approximate the model coefficients of Equation 4.11. The configuration space includes two parameters: the throttling threshold (Δf_m) and the number of threads (Δt). The threshold ranges from 0 to 126 idle cycles and the number of threads from 4 to 64 with an interval of 4. The details of the training configurations and the overall configuration space is given in Table 4.3. We select minimal number of configurations to train the model and do extrapolation-based prediction for the other configurations. We use the five code regions of LULESH to train the model. Each region has its own trained coefficients.

We use the model to predict the performance of the five code regions of LULESH for those configurations in the configuration space that are not in the training set. To measure the accuracy of the model, we compare these predicted values with the performance measured by running the same configurations on the machine.

Table 4.3: The training configurations and the overall configuration space on BG/Q. The *Total* column shows the number of configurations.

Total	Δf_m	Δt (num. threads)
	Training inflection point a	
127×16	0 - 126 cycles	4, 8, 12, ..., 64
	Training slope b for Region 1	
27×14	100 - 126 cycles	12, 16, ..., 64
	Training slope b for Region 2	
16×10	100 - 115 cycles	28, 32, ..., 64
	Training slope b for Region 3	
11×12	35 - 45 cycles	20, 24, ..., 64
	Training slope b for Region 4	
16×9	100 - 115 cycles	32, 36 , ..., 64
	Training slope b for Region 5	
11×11	65 - 75 cycles	24, 28, ..., 64
	Configuration space	
127×16	0 - 126 cycles	4, 8, 12, ..., 64

Figure 4.10 shows the average error and standard deviation of our model. Four of the five code regions show a reasonable average error, around 10% or less. Region 2, however, shows a large average error of 17%.

There are several factors that affect the model accuracy of the BG/Q implementation of the COS model. First, our experiments on BG/Q included prefetching, which makes the overlap time T_o more complex as observed on the Intel system. On BG/Q we used L2 cache misses to represent memory pressure similar to the Intel system. We could relax this requirement and use the number of load and store operations on BG/Q for our approximations. This may improve accuracy.

Second, in some cases the value of a may not be strictly constant but a linear function with a small slope value. The number of idle cycles that can be inserted (integers) does not provide fine-enough granularity to estimate a more accurately.

Third, we used a small number of sample configurations for training a because we limited our configuration space to only a subset of the available number of threads. We expect that a larger space using all 64 (from 1 to 64 threads) configurations along the Δt dimension would have resulted in better accuracy.

4.4 Limitations and Discussion

We have demonstrated the use and accuracy of the COS model for predicting the performance of a set of DVFS, DMT/DBT, and DCT configurations. The model can be used in a software or hardware implementation to allocate or deallocate resources to the working threads in a parallel application. This is an advantage to a parallel scheduler or runtime system.

As mentioned the key concept of the COS model is the isolation of overlap. While in an abstract sense this is straight forward, we show in Sections 4.1.4 – 4.1.6 that empirically isolating overlap is wrought with challenges. We resolved a number of these challenges using the assumption of regular parallel applications where threads are mostly homogeneous and computation proceeds in a bulk synchronous way with no other dependencies among threads. This leaves a number of limitations to the model that must be addressed to consider irregular parallel codes (e.g., heterogeneous threads, asynchronous, cross-thread dependencies).

Overlap types In our earlier discussions, we simplified the definition of overlap into computation overlap and memory overlap. In general, overlap can also occur between multiple threads on a single core/CPU or across multiple cores/CPUs accessing the same memory. Under our assumptions, these don't affect the COS Trace much, but these must be considered for irregular parallel codes.

Computational intensity Computational intensity (CI) has impact on the overlap as

discussed earlier. A key insight gained from this work is that CI can be used to predict the impact of simultaneous configuration changes in CPU, memory, and threads on overlap. More overlap types, combined with irregular codes, are likely to make accurate prediction more challenging. There could also be non–CI effects that we've not accounted for in parallel irregular applications.

Role of Co-design These challenges could be alleviated somewhat by improvements in our ability to directly measure the overlap of parallel codes. This could be accomplished in software, but would be most effective when co-designed with hardware. Our work indicates that there are meaningful representative hardware counters that give insight to overlap and computational intensity, but they are indirect at best. Furthermore, this data is usually limited to an individual thread with no context for other concurrent threads on the same core or CPU. Mechanisms for tracking this type of information could vastly improve our understanding of overlap as well as our ability to optimize parallel applications and systems.

4.5 Summary

In this chapter, we propose the COS Model of parallel performance to accurately capture the combine effects of DVFS, DMT/DBT, and thread concurrency on real systems. We applied the COS model to both Intel and IBM architectures within 7% and 17% accuracy for a set of 19 important applications. The key insight to the COS model is the separation of memory and compute overlap from pure compute and pure memory stalls. This separation enables more accurate approximations and a straightforward methodology that is capable of modeling the complexity introduced with concurrency. A key limitation of the model is the focus on structured parallel codes that while representative of many important applications precludes accurate use on irregular parallel codes for now. Despite the limitations, we provide

strong evidence that the fundamental focus on overlap in the COS model will be key to steering future high-performance systems and applications to maximize their efficiencies. In future work, we plan to explore extending the COS model to irregular parallel applications in both OpenMP and MPI. We also plan to adapt the techniques described for use in runtime systems.

Chapter 5

Runtime System Design

In this chapter, we develop a runtime system for improving efficiency by leveraging an automatic performance prediction technique from Chapter 4. Runtime system for steering DVFS, DMT, and DCT must predict application performance with low overhead and high accuracy, and adapt to various application characteristics. However, using the performance prediction approach in Chapter 4 results in tradeoffs between performance prediction accuracy and overhead. In this chapter, we develop two key metrics for capturing application characteristics and apply machine learning techniques to attain low-overhead, accurate performance prediction.

5.1 Characterizing C/O/S

5.1.1 Performance Impact

Reducing CPU speed, memory speed, thread concurrency level, or a combination of them can improve efficiency but lead to performance degradation. For example, we can configure all three components to minimize power use but the performance loss will be substantial. Given the expected performance degradation, we want to identify configurations of CPU speed (f_c), memory speed (f_m), and thread concurrency (t) that meet a performance target (T_{exp}). In the following sections, if not mentioned, we mean specifically performance degradation when

Table 5.1: Notations

f_c	CPU speed.
f_m	Memory speed.
t	Thread concurrency level.
T	Performance (i.e., the total execution time) under original configuration (f_c, f_m, t).
$T^{'}$	Performance under new configuration $(f_c^{'}, f_m^{'}, t^{'})$.
T_c	Pure compute time.
T_o	Overlap time.
T_s	Pure stall time.
PI	Performance impact from original configuration to new configuration.
SI	Stall intensity.
MTR	Memory transaction rate.
$LLCM/s$	Total number of last-level cache misses per second.

we talk about "performance impact". In our work, we define the performance impact (PI) using Equation 5.1:

$$PI = \frac{T^{'} - T}{T} \tag{5.1}$$

where T is the total time under the old configuration (f_c, f_m, t) corresponding to CPU speed, memory speed, and thread concurrency, and $T^{'}$ is the total time under a new configuration $(f_c^{'}, f_m^{'}, t^{'})$. In order to regulate the power consumption within a power budget, we look for a new configuration whose power consumption meets the requirement and whose performance impact (PI) is as small as possible (the best performance under a power cap):

$$PI = \frac{T^{'} - T}{T} = \frac{\Delta T}{T} \tag{5.2}$$

According to the definition of the COS model, the total time T is the sum of pure compute time (T_c), overlap time (T_o), and pure stall time (T_s). So, we can revise the previous equation

to get Equation 5.3:

$$PI = \frac{\Delta T}{T} = \frac{\Delta T_c + \Delta T_o + \Delta T_s}{T} = PI_c + PI_o + PI_s \tag{5.3}$$

As in the COS model in Chapter 4, we must separate and predict the impact of reducing CPU speed, memory speed, thread concurrency, or any combination on PI_c, PI_o, and/or PI_s. We begin by identifying critical performance metrics that strongly correlate to performance impact. We initially focus on three scenarios of configuration change: reducing CPU speed only to $(f_c^{'}, f_m, t)$, reducing memory speed only to $(f_c, f_m^{'}, t)$, and reducing thread concurrency level only to $(f_c, f_m, t^{'})$.

5.1.2 Characterizing Performance Impact on C/O/S

Measuring pure compute time, overlap time, and pure stall time directly is not possible on real systems due to limited hardware support and extreme system complexity.

Approximation approaches, such as linear regression, can be used to estimate the time of pure compute, overlap, and pure stall times through extensive training as described in Chapter 4. However, tuning a runtime system requires fast decision making with a very limited amount of information. Furthermore, no single, offline training set we've found can accurately predict all possible application scenarios. Existing methods for approximating C/O/S or any combination of them (e.g., $C+O$ or $O+S$) cannot meet the requirements of practical runtime control systems.

In this chapter, we present two important metrics for capturing the performance impact (PI) of CPU speed, memory speed, and thread concurrency. We will leverage the correlation between a combination of the two metrics and the performance impact on C/O/S.

Stall Intensity (SI) is defined as the pure stall time T_s over the total time of application execution T. SI gives the application proportion of pure stall time in the total time, which ranges from 0 to 1. An application with high stall intensity suffers from latency caused by slow memory accesses, while low stall intensity indicates that either the application has fewer memory transactions or most of them overlap with CPU activities.

Memory Transaction Rate (MTR) captures the number of main memory accesses per second. MTR give the application throughput of the memory subsystem. Typically, memory-intensive applications (e.g., STREAM benchmark) have a higher MTR than compute-intensive applications (e.g., LINPACK) with a lower MTR. We estimate the value of MTR using the total number of last-level cache misses per second (LLCM/s). We have disabled prefetching to reduce the complexity of overlap prediction. However, as in Chapter 4, we plan to revisit this simplified assumption.

While SI and MTR in isolation provide insights to capture the system and application behavior, taken together they can help us characterize the impact of configurations on C/O/S. In Table 5.2, we list four domains with combinations of low or high for SI and MTR. Correspondingly, in Figure 5.1, we show four conceptual examples of application profiles exhibiting the characteristics of the scenarios in Table 5.2. For each scenario in Figure 5.1, the x-axis is time increasing left to right.. The black line is the CPU active time for a single thread. Each red line is the access time for a thread's main memory transaction. Depending on the behavior of applications and the configuration, the number of red lines and their distributions vary.

Table 5.2: Characterizing Impact on C/O/S.

		MTR (LLCM/s)	
		low	high
Stall Intensity	high	①	③
	low	②	④

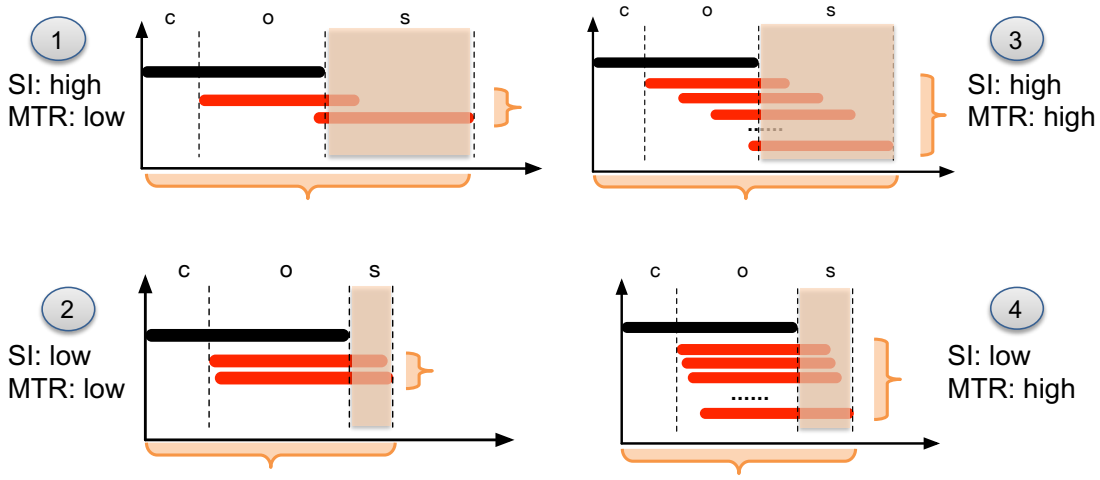


Figure 5.1: Characterizing Impact on C/O/S.

In the first scenario (labeled 1), SI is high while MTR is very low. The proportion of overlap time depends on the time when the first memory transaction is issued. Early issuing of the first memory transaction can make the proportion of the overlap time increase. Later issuing of the first memory transaction can make the proportion of overlap time decrease.

The second scenario (labeled 2) in Figure 5.1 is the example of applications with both low SI and low MTR. Low SI indicates that the application spends most of its time in CPU processing instead of waiting for memory transactions to complete. Low MTR indicates that there exist very few memory transactions per second to the main memory. Under this scenario, the amount of pure stall time is very small, while the amount of overlap can be very high because most of the latency caused by memory transactions is overlap time. The amount of pure compute time is relatively large due to the small total time overall.

Scenario 3 (labeled 3) has both high SI and high MTR. Compared with scenario 1, it has more memory transactions issued within the same amount of time (i.e., high MTR). The large amount of memory transactions and their distribution leads to high proportion of stall time over total time (i.e., high SI).

The fourth scenario (labeled 4) has low SI and high MTR. In this example, we assume the pure compute time stays roughly the same as in the second scenario and the first memory transaction begins at about the same time point as well. Under this scenario, the application has very low SI even though it has more memory transactions. This is possible because all the memory transactions happen within a short span of time (i.e., high MTR) and a relatively large amount of stall time over multiple transactions is overlapped..

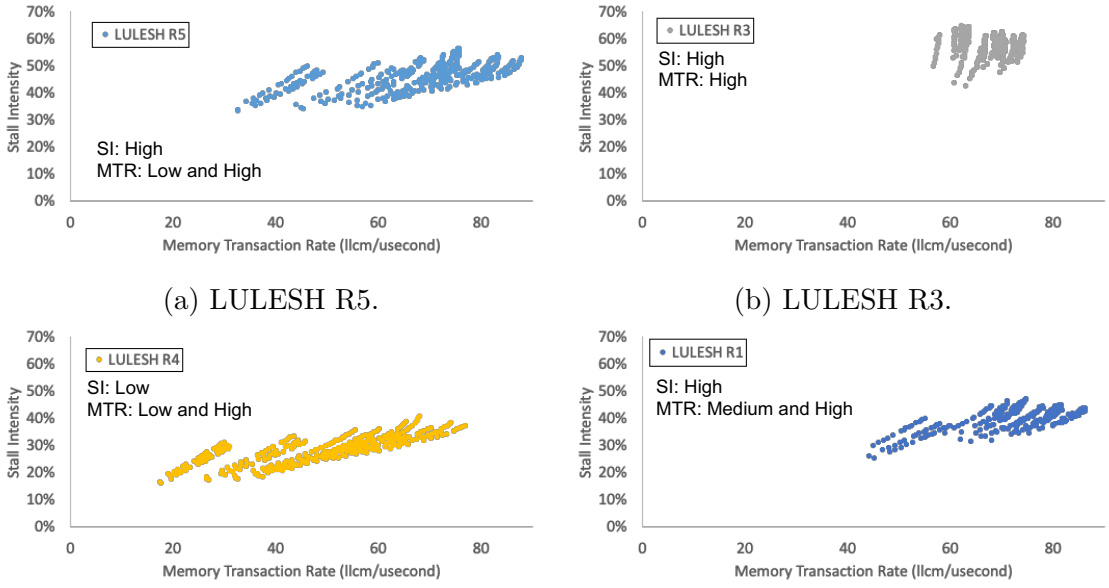
5.1.3 SI and MTR Application Signatures

Before we use SI and MTR in our COS predictors at runtime, we demonstrate their use in characterizing applications. Figure 5.2 shows the sensitivity of four distinctive code regions representing the four scenarios of Figure 5.1.

In this figure, we profile four different kernels from the LULESH benchmark under different configurations of (f_c, f_m, t). In the case of a single scenario, such as LULESH code region 5 (LULESH R5 in Figure 5.2a), MTR changes significantly from low to high under different (f_c, f_m, t) configurations, while the change to SI is about 15% of total time at most. Also, the range of SI indicates that LULESH R5 maintains an SI higher than 40% of total time no matter what configuration we choose. Comparatively, LULESH R4 has a broad dispersion of memory transaction rates (Figure 5.2c) under different (f_c, f_m, t) configurations. However, the SI changes are generally lower than that of LULESH R5. On the other hand, the behavior of LULESH R3 is more stable, as almost all of the measured (MTR, SI) fall into the quadrant of high SI and high MTR. We can observe that, even for a single application, the runtime behavior (i.e., MTR) can change significantly. That means an application can independently transition among the different scenarios shown in Figure 5.1 or transition among different scenarios with the changes of configuration (f_c, f_m, t). Furthermore, SI

and MTR application signatures vary leading to additional scenarios and transitions among scenarios. These transition limit the accuracy of linear approximation methods such as linear regression for predicting the performance impact.

We can also observe that the wider the points are spread throughout the four quadrants, the more nonlinear it is. LULESH R3 has the best linearity, since all of its points are located in the high/high quadrant. When we consider modeling multiple applications, the changes of SI and MTR are both significant. This brings extreme nonlinearity to the relationship.



(a) LULESH R5.

(b) LULESH R3.

(c) LULESH R4.

(d) LULESH R1.

Figure 5.2: Non-linear effects of considering multiple applications.

5.2 Methodology

5.2.1 Correlation Analysis

To illustrate the limits of linear approximation further, we used the multivariate linear regression to approximate the relationship between performance impact (the dependent variable) and (f_c, f_m, t, SI, MTR). An R^2 value expresses the linear relationship among dependent and independent varaibles as a range from 0 (the most nonlinear) to 1 (perfectly linear). Figure 5.3 shows R^2 value and prediction accuracy for each single application (or code region). We measure R^2 values for five kernels from LULESH and five kernels from MILCmk and use linear regression to approximate performance impact. The blue bars in Figure 5.3 are the R^2 value for each case. The average performance impact is the mean of performance impact over various configurations. The maximum approximation error is the maximum absolute difference between measured performance impact and the value of the approximation. The values are labeled on top of each bar. We should note that the training set and prediction set of data are the same since we are evaluating the linearity of data rather than doing real prediction.

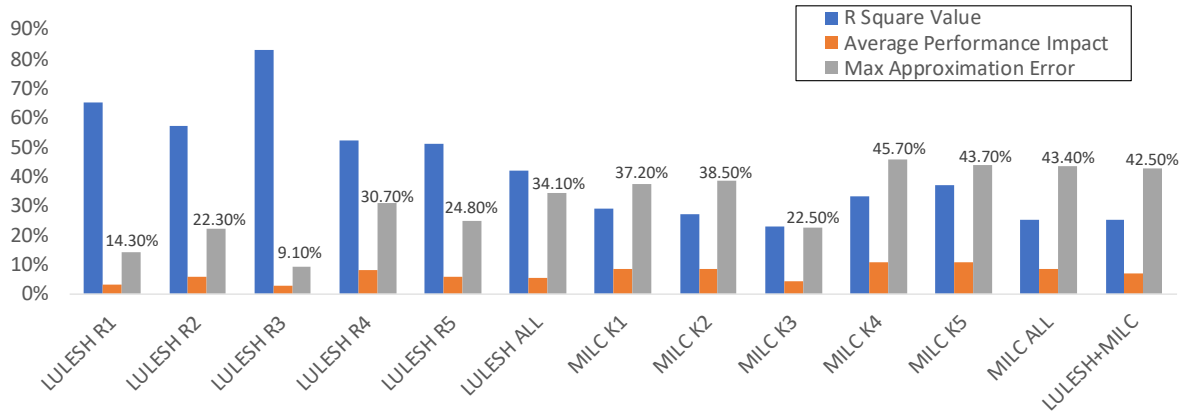


Figure 5.3: Accuracy of linear approximation.

5.2.2 Artificial Neural Network (ANN) Model

The artificial neural network [100] is a nonlinear method that emulates the way the human brain transmits and processes information to resolve complex (usually nonlinear) modeling problems. The most basic function unit is a neuron with input and/or output linking from/to other neurons. The connection between neurons has a weighted value, which indicates the relationship between the value carried by the two neurons. The structure of a directed graph from multiple neurons to one target neuron forms the function of:

$$Y = \sum_{i=1}^{n} w_i \times X_i \tag{5.4}$$

where Y is the value of the target neuron, X_i is the value of each individual neuron that points to the target neuron, and w_i is the coefficient of each connection from the source neuron to the target neuron. When the coefficients are all constant, the equation is the same as linear regression. Since the ANN model has multiple layers and dynamically adjustable coefficients, it can effectively capture the nonlinear relationship between the input variables and the output variable. For example, the X_i (associated with neuron i in its layer) in Equation 5.4 is the weighted summation of all the neurons that point to that neuron. These neurons are from the layer before the layer to which the target neuron belongs.

Usually, a neural network can model more complicated nonlinear functions when it has more layers. When the number of layers becomes large enough, it is commonly called "deep learning". Deep learning, though it can improve the accuracy of nonlinear modeling, suffers from low performance due to its complex structure and demands for a large amount of training data.

In our work, we use an ANN with four layers to model the performance impact function. The neurons and connections form a complete directed acyclic graph. The structure of ANN

that we use is shown in Figure 5.4. The input layer (first layer) has five neurons, each of which is a parameter we identified as correlating to performance. These include the change of CPU speed (Δf_c), the change of memory speed (Δf_m), the change of thread concurrency (Δt), stall intensity (SI), and memory transaction rate (MTR).

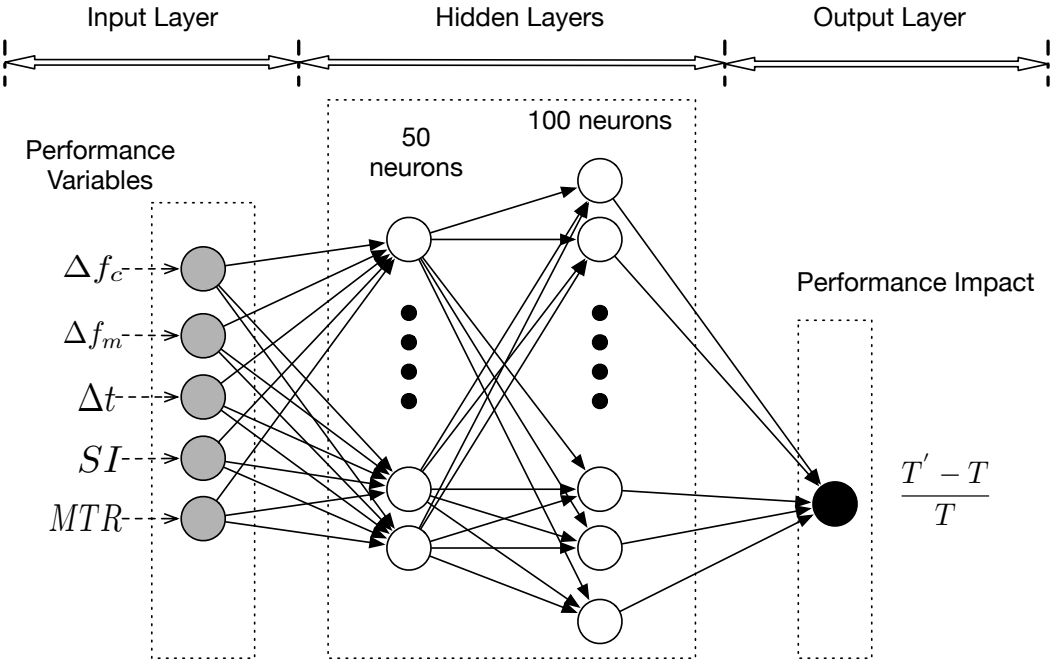


Figure 5.4: Structure of artificial neural network.

Our ANN model has two hidden layers. The first hidden layer has 50 neurons and the second one has 100 neurons. We chose the number of hidden layers and the number of neurons in each layer via tuning of the prediction accuracy of our model. We found that using more than two hidden layers did not result in any improvement in accuracy but significantly slowed the training process. Our findings indicate that five parameters using 50 neurons and 100 neurons in two hidden layers is enough to capture the nonlinear relationships (i.e.,performance impact).

The output layer has only one neuron and is the target we want to predict: the performance impact. Since the value of the performance impact is a real number, we use the regression

mode of the ANN rather than the classification mode [88].

In theory, our ANN first divides the training set of data into batches. As each batch of data is fed sequentially to the model, the coefficients get updated according to the difference between the predicted value and the training data. After a certain number of iterations (depending on the structure of the neural network and the training data), the coefficients converge to a relatively stable state.

5.2.3 Runtime Control Strategy

Though these scenarios in Figure 5.1 may be more nuanced in practice, we can use SI and MTR to classify an application (or phase of an application) with regard to the effect of changes to CPU speed (f_c), memory speed (f_m), and thread concurrency (t) on C/O/S. Additionally, changes to f_c, f_m, and t can manipulate SI and MTR to achieve different percentages of pure compute time, overlap time, and pure stall time versus total time. Since SI and MTR can be accurately and quickly measured at runtime with manageable overhead, they can be used as inputs to a runtime system that predicts the effects of Δf_c, Δf_m, and Δt on power and performance. More precisely, we will use theses insights to predict the performance impact ($\Delta PI_c, \Delta PI_o, \Delta PI_s$) for configuration changes ($\Delta f_c, \Delta f_m, \Delta t$). Hence, we can rewrite Equation 5.3 as follows:

$$\begin{aligned} PI =& fun_c(\Delta f_c, \Delta t, SI, MTR) + fun_o(\Delta f_c, \Delta f_m, \Delta t, SI, MTR) + \\ & fun_s(\Delta f_m, \Delta t, SI, MTR) \\ =& fun_{cos}(\Delta f_c, \Delta f_m, \Delta t, SI, MTR) \end{aligned} \tag{5.5}$$

For different applications, the coefficients in the three functions (fun_c, fun_o, fun_s) are dif-

ferent due to varying application behaviors and memory access patterns. The coefficients depend on the portion of each type of time (C/O/S) as well as the program behavior captured by the combination of (SI, MTR).

Due to limited hardware support on real systems, we cannot measure pure compute time, overlap time, and pure stall time separately. That makes the modeling of the functions $fun_c(), fun_o(), fun_s()$ in Equation 5.5 very challenging. Instead of modeling these terms separately, we estimate the overall performance impact by considering the factors that affect PI_c, PI_o, PI_s. The function fun_{cos} can capture that relationship. Since we want to apply the function to predict runtime performance, we prefer a minimal amount of runtime data collection to predict a large set of applications.

ANN-based performance prediction can accurately capture the performance impact of reducing CPU speed, memory speed, or thread concurrency under any configuration. We leverage our accurate prediction in a runtime system design to maximize performance under power capping.

Existing power capping solutions mostly rely on a hardware feature capping CPU and DRAM power consumption together or separately, i.e., Intel RAPL. By setting the maximal allowed power consumption of the system, for example, the system will dynamically adjust the CPU speed based on an internal power model while monitoring the execution of applications. Similarly, the system can also throttle the speed of the memory subsystem if the power cap for DRAM is preset. However, as we learned in Chapter 4, independent throttling can result in over correction and significant performance loss. Moreover, to maximize efficiency (and performance) a power capping runtime system should consider coordinating not only DVFS and DMT but also DCT. As before this is challenging due to the complex combined effects of throttling techniques on performance. Furthermore, predicting unseen applications at runtime adds to the difficulty.

We propose a greedy walk algorithm that can make step-wise throttling decisions using our ANN model shown in Algorithm 2.

Algorithm 2 Algorithm of Greedy-Walk with CPU RAPL

1: **Initialization:** Set the system power budget.
2: Collect the power consumption and parameters of the application under the configuration $(f_c^{max}, f_m^{max}, t^{max})$.
3: Start of the **selection phase** of configurations:
4: **Before** the beginning of an iteration in the application, predict the *pivot* configuration (f_c^i, f_m^j, t^k) by only doing CPU DVFS.
5: The pivot configuration is the configuration where at least one configuration from (f_c^{i+1}, f_m^j, t^k), (f_c^i, f_m^{j+1}, t^k), and (f_c^i, f_m^j, t^{k+1}) can meet the power budget.
6: Set the system and application to the pivot configuration.
7: **Start** the iteration of application and collect the performance and parameters by the end of this iteration.
8: Predict the performance of configurations (f_c^{i+1}, f_m^j, t^k), (f_c^i, f_m^{j+1}, t^k), and (f_c^i, f_m^j, t^{k+1}). Select the configuration that both meets the power budget and has maximal performance.
9: Start the **running phase** for the rest of the iterations under the selected configuration.

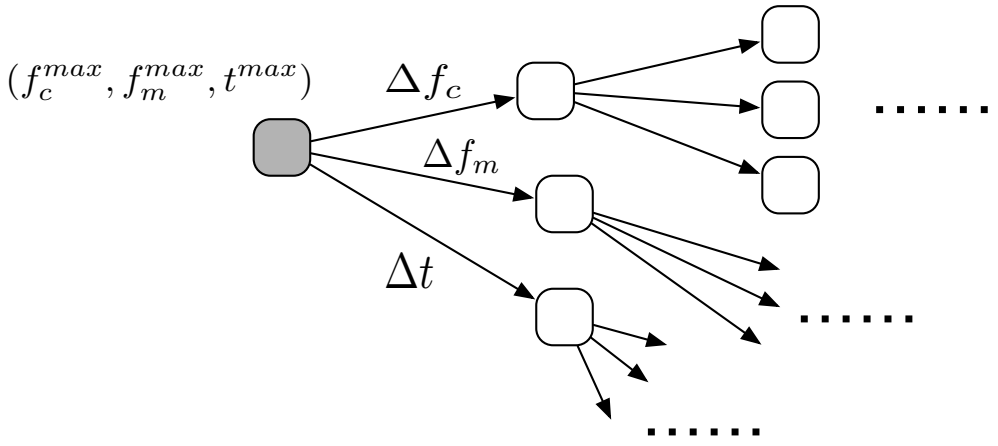


Figure 5.5: Illustration of step-wise selection of configurations.

The algorithm relies on a power model to predict the power consumption under different configurations. In theory, the algorithm should find the configuration that delivers better performance under the power budget than the default CPU DVFS-based RAPL tool. Whenever the power budget is given, our algorithm first identifies the pivot configuration in which

we have at least one choice of further throttling: DVFS, DMT, or DCT. If we have multiple choices, then we apply performance prediction to find the next configuration: reduce CPU speed, memory speed, or thread concurrency based on the pivot configuration.

Figure 5.5 illustrates the step-wise selection of a configuration. The execution of an application starts by running at the full/highest configuration ($f_c^{max}, f_m^{max}, t^{max}$). With the given power budget and power consumption information, the algorithm decides which is the pivot configuration. In this work, we always choose to reduce CPU speed f_c to compare with the CPU DVFS-based RAPL power capping technique. Then we apply our model to predict the new configuration with the best performance under the power budget.

A Power model is one necessary component for implementing power capping. Intel RAPL dynamically throttles the system performance based on the model's power prediction under different CPU and memory speeds. Since the power model used by RAPL is not publicly available, we have measured and built a linear regression-based power model for different configurations in order to conduct a comparison with RAPL solutions. A RAPL-like approach was selected in this work as the best available practical approach to modeling power at runtime. While our approach requires a power model to optimize for performance under a power cap, we emphasize the approach is general and can incorporate any power model to improve accuracy.

5.3 Validation

In this section, we compare the runtime control algorithm with a RAPL-like baseline power capping approach. We first introduce the details about the system we are using and the domain applications. Also, we provide details about the ANN tools, network topology, and other configurations. Moreover, we show the accuracy of using an ANN-based predictor to

estimate performance impact for different configurations (f_c, f_m, t) and for different applications. Finally, we compare the accuracy and efficiency of our method to an empirical approach.

5.3.1 Experimental Setup

System Setup

We conduct all of our experiments on the "Wendy" cluster of systems. It is the same x86 systems we used in Chapter 4. Wendy has 12 homogeneous nodes with two Intel Xeon E5-2623 v3 Haswell CPUs and 32 GB of DDR4 memory. The CPU supports 16 different CPU frequencies ranging from 1.2 GHz to 3.0 GHz. The two sockets has 8 CPU cores in total and support 16 hyperthreads. The memory system supports three bus frequencies: 1333, 1600, and 1866 MHz.

Applications/Kernels

In order to cover the application behavior as completely and diversely as possible, we select 37 parallel applications/kernels. 19 of them are new kernels in addition to the kernels we investigated in Chapter 4.

LULESH is a proxy application for an explicit hydradynamics application developed at LLNL [1]. We choose five different code regions/kernels which represent different data access patterns and computational characteristics.

AMGmk is a compute-intensive benchmark with three different kernels from AMG, which is derived from BoomerAMG solver in the Hypre linear solvers library [97].

Rodinia is a benchmark suite with applications from different domains. It supports not

only OpenMP, but also, MPI, CUDA, and OpenCL. We selected six OpenMP applications from them: Kmeans as data mining application, k-Nearest Neighbors (kNN) as data mining application, Breadth-First Search (BFS) as graph traversal application, HotSpot as physics simulation application, LavaMD as molecular dynamics application, and LU Decomposition (LUD) as linear algebra application.

pF3D is a parallel application that simulates laser-plasma interactions at the National Ignition Facility at LLNL. We identify four kernels from the application. They consume most of the execution time. These kernels are Absorbdt, Acadv, APCPFT, and Advancefi.

NASA Parallel Benchmark Suite contains several scientific parallel applications that have different computational characteristics. We select the OpenMP applications of *LU, SP, FT, CG,* and *BT*. Specifically, we identify five different kernels from the application *BT*.

CLOMP is an OpenMP benchmark for measuring the overhead and other performance impacts due to threading. It has different data access patterns when the threading changes.

MILCmk has some representative kernels for the application of MIMD Lattice Computation (MILC). We focus on five kernels. The application simulates the fundamental theory of the strong nuclear force and quantum chromodynamics.

XSBench is a parallel application for Monte Carlo Neutron Transport. It is used for evaluating the performance of the memory subsystem.

DGEMM is from the Crossroads benchmark suite. It is an OpenMP version dense matrix multiplication benchmark for measuring computation performance with a large number of floating point operations.

Machine Learning Tools

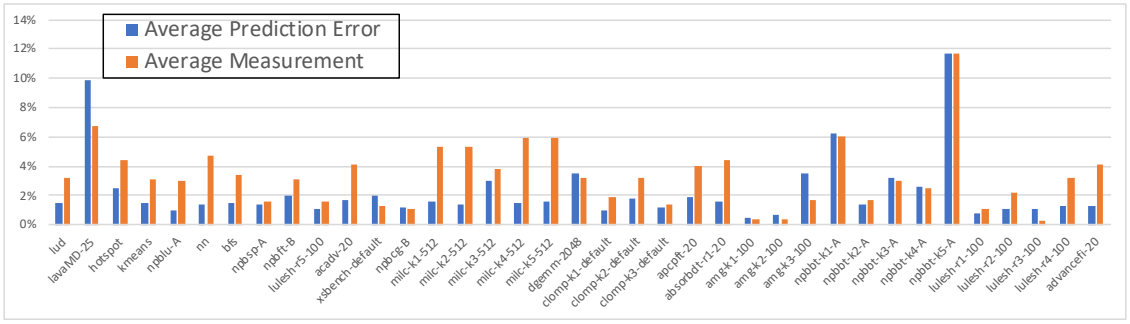
For fast and efficient experimentation, we choose Keras as the high-level neural network library. It uses Theano as the backend engine for building models and data prediction. Keras also supports other neural network engines such as TensorFlow and CNTK.

5.3.2 Accuracy of the ANN Model

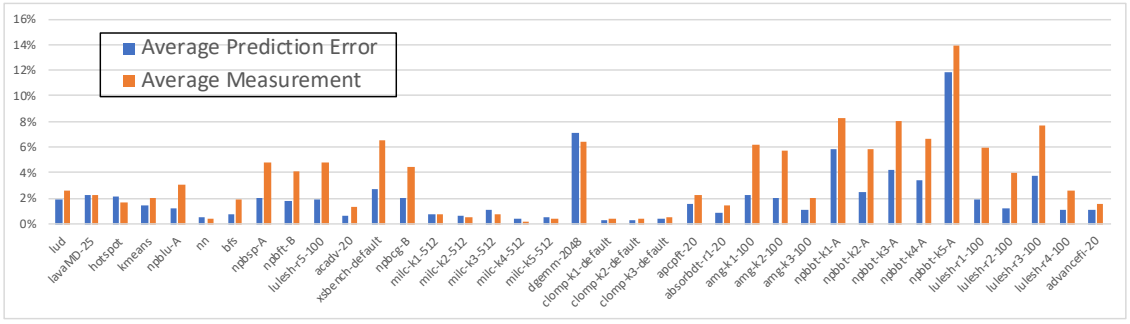
The number of data entries for training the neural network model is very important. A large amount of data can help to achieve a more accurate nonlinear regression. To validate our model with 37 applications, we select one application for prediction and use the other 36 applications to train the ANN model. In this way, the scale of the training data set is in the tens of thousands.

Based on numerous attempts, we found that predicting the exponential configuration space of all permutations of (f_c, f_m, t) had an impractical amount of overhead. Through experimentation, we found predicting the performance of a single change using the COS-based method provided a balance of optimization accuracy and overhead. Hence, instead of predicting the performance impact from a base configuration to any new configuration, we model the new configurations caused by: 1) reducing CPU frequency by one level, 2) reducing memory frequency by one level, or 3) reducing thread concurrency by one level. Following such rule makes the feature space of $(\Delta f_c, \Delta f_m, \Delta t)$ much smaller, which helps improve the model's accuracy. For example, on our system, we reduce the configuration space by 80%.

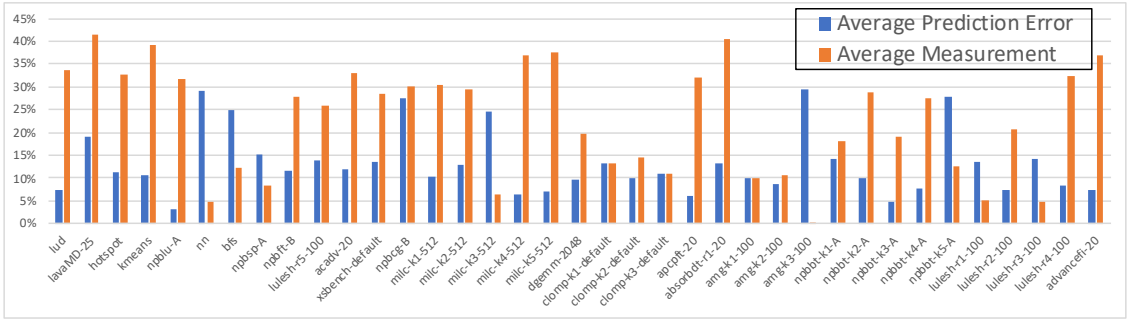
Another optimization of our ANN model is done by separating the training for DVFS, DMT, and DCT, so that the model can handle less nonlinear effects. For example, we train the model using all the data entries when only Δf_c is applied to predict the impact of DVFS on all applications.



(a) Model for DVFS



(b) Model for DMT



(c) Model for DCT

Figure 5.6: Accuracy of performance impact models under DVFS, DMT, and DCT.

After all the optimizations for improving model accuracy, Figure 5.6 shows the prediction accuracy of performance impact for 37 applications. In each figure, we show the average prediction error of performance impact (blue bars) and the average measured performance impact (orange bars). If one application has a very low average prediction error and a very high average measured value, such as MILC-K2 in Figure 5.6a, it means the prediction is very accurate. As can be seen by comparing the three figures in Figure 5.6, the predictions

of MILC-K2 for DVFS and DCT are more accurate than for DMT. Prediction of NPB-LU is very accurate no matter the throttling decision.

Since we choose to reduce the feature space of configuration changes $(\Delta f_c, \Delta f_m, \Delta t)$, the prediction error is very small. Most of the time, the prediction error is within 2%. The example is the five kernels of MILCmk shown in Figure 5.6b. In this case, even though the prediction error bar is greater than or equal to the measured value, the performance prediction is still very accurate.

In the results, we can see a few cases of bad predictions, where the average prediction error is higher than the average measurement, such as LavaMD and NPB-BT-K5. Such bad predictions are usually the same for all three throttling decisions. The main issue here is the training set is not representative of these applications.

5.3.3 Comparison to RAPL-Based Method

The limitation of RAPL-based power capping techniques is throttling decisions are made only according to power consumption. For example, RAPL-based DVFS throttles CPU speed to meet the power budget. By extending the power model, we can apply this naive approach to DMT and DCT. In Figure 5.6, we compare the measured performance impact due to DVFS, DMT, and DCT to this naive approach. Some applications, such as MILC kernels and pF3D kernels, have less performance impact under DMT than under DVFS or DCT. This means that if we have multiple choices under power capping, we should throttle DMT to achieve the best performance. In contrast, some applications, such as AMGmk kernels, are affected by DVFS the least.

Figure 5.7 shows the performance improvement of our approach over the naive RAPL-based method. In this comparison, we cap the power consumption to be less than 97% of peak

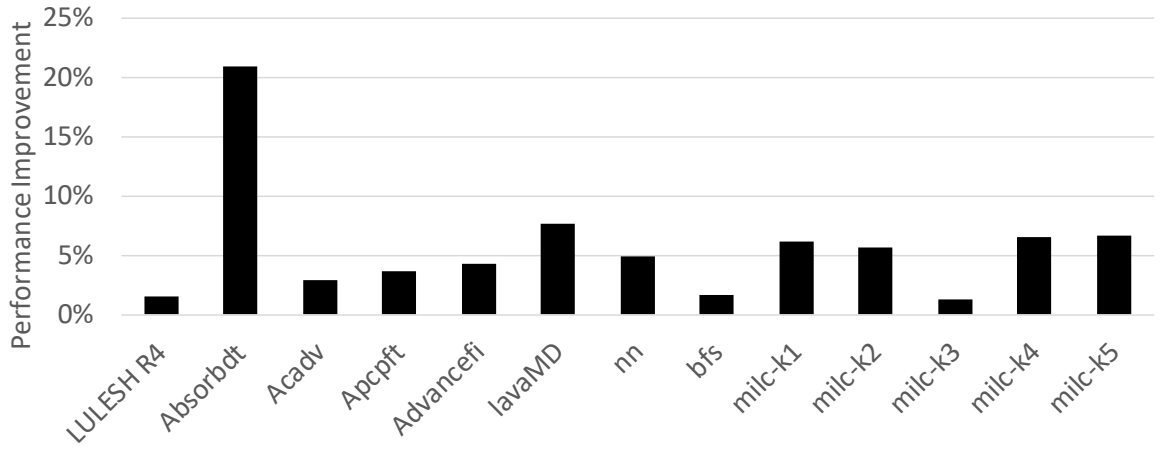


Figure 5.7: Performance comparison between RAPL-based approach and our RT system.

power consumption (i.e., under $(f_c^{max}, f_,^{max}, t^{max})$). The performance improvement is the difference between the performance under our runtime control and that of the RAPL-based decision. For those applications on which DMT has less impact than DVFS, reducing memory speed rather than CPU speed can lead to better performance. The improvement is up to 21%, which is quite significant when we are comparing the difference in performance between reducing CPU speed or memory speed.

5.4 Summary

In this chapter, we characterized the performance impacts of DVFS, DMT, and DCT on pure compute time, overlap time, and pure stall time. We further identified key metrics for quantifying these impacts and applied an artificial neural network model to predict the performance impact. Accordingly, we proposed a runtime control strategy to improve performance under power capping. We compare our method with existing naive RAPL-based power capping techniques. Our method achieves up to 21% performance improvement by coordinating DVFS, DMT, and DCT together.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

The demands for system efficiency motivate coordinating multiple power management techniques for optimization. This dissertation presents a series of methods to solve this problem of efficiency optimization. We begin with empirical analysis to explore the tradeoffs between performance and energy for HPC applications on Intel and IBM systems. Further, we develop an accurate and insightful parallel performance model to predict the performance under simultaneous control of DVFS, DMT, and DCT. Next, we propose a runtime control strategy to optimize the performance of parallel applications under power capping.

We conduct empirical analysis of the tradeoffs between performance and energy consumption on both Intel systems with accelerators (i.e. Intel Xeon Phi and Nvidia GPGPU) and IBM BG/Q systems. We extend PowerPack to profile power and performance at function-level for critical components of the systems. By applying different throttling techniques, we observe sweet spots of energy efficiency and show insights for optimizing system efficiency. Our results indicate that although the Intel Xeon Phi typically outperforms the older GPU, the results are mixed with regard to the energy efficiency. We also show that memory throttling techniques of IBM BG/Q system can effectively reduce energy consumption under certain performance loss tolerances.

Coordinating multiple power management techniques shows promising results for improving system efficiency based on our analyses. We propose a new parallel performance model for accurately capturing the combined effects of DVFS, DMT, and DCT. We separate the overlap time (O) of computation and main memory accesses from the pure compute time (C) and the pure stall time (S). Due to the limited support on real systems, we cannot measure C, O, and S separately. We apply a multivariate linear regression model to approximate the C, O, and S under different settings of DVFS, DMT, and DCT. We validate our approach using 19 HPC applications/kernels on both Intel x86 systems and IBM BG/Q systems. The average absolute prediction error for performance is up to 7% on Intel systems and 17% on IBM systems.

Dynamical coordination of DVFS, DMT, and DCT at runtime requires low overhead performance prediction with small prediction errors. We design the runtime control strategy to meet the two requirements. We characterize the performance impacts of DVFS, DMT, and DCT on the pure compute time, the overlap time, and the pure stall time. We further identify the key metrics for quantifying such impacts. By applying neural network techniques to model the nonlinear relationships between system metrics and performance impacts, we can achieve very low prediction errors for new parallel applications. We also propose a runtime control strategy to improve the performance under power capping. We compare our method with existing, albeit naive, RAPL-based power capping techniques. Our method achieves up to 21% performance improvement by coordinating DVFS, DMT, and DCT together.

6.2 Future Work

6.2.1 Extended Performance Model for MPI/OpenMP Applications

The ultimate goal of coordinating multiple power management techniques is to improve system efficiency on large scale high performance systems. Parallel performance models at the node level can contribute but are not enough for predicting performance of parallel applications on multi-node high performance clusters. An analytical parallel performance model for MPI/OpenMP applications can separate the impact of DVFS, DMT, and DCT on the pure compute time, the pure memory stall time, the pure network stall time, and different types of overlap time. The scenarios of overlap are much more complicated when considering multiple nodes. For example, overlap can happen between computation and memory accesses, computation and network communication, memory accesses and network communication, etc. A COS-based parallel performance model for MPI/OpenMP applications can better capture the impact of DVFS, DMT, and DCT on the performance.

6.2.2 Improved Runtime Performance Prediction under DVFS, DMT, and DCT

Runtime behaviors of parallel applications under DVFS, DMT, and DCT are complicated and cannot be fully captured by existing performance metrics. In our runtime system design chapter, we develop two new metrics to analytically model the performance impact of all parallel applications. Though the prediction error of performance is low in general, there are some applications that yield high errors. To improve the accuracy of performance prediction, we can cover more applications with different runtime characteristics at the training stage

of our ANN model. With a better trained ANN model, the performance prediction accuracy for new applications could improve significantly.

6.2.3 Efficiency Optimization on Heterogeneous Architectures

Heterogeneous systems, such as GPGPUs, AMD APUs, and Intel Xeon Phi's, can significantly improve system performance by coordinating workload between host CPUs and accelerators. However, efficiency is highly dependent on the application characteristics. There are at least five throttling techniques (DVFS, DMT, and DCT on both host system and accelerators) available on heterogeneous systems. Using COS model can effectively improve the efficiency on these systems.

6.2.4 Smarter Runtime Control Algorithm for Performance Optimization

Our runtime control strategy based on the greedy algorithm proposed in Chapter 5 can find a more efficient configuration of DVFS, DMT, and DCT than a RAPL-based naive approach to power capping. However, greedy algorithms cannot always guarantee optimal solutions. With the aid of improved hardware measurements of higher fidelity in our techniques, we can likely improve these techniques substantially.

Bibliography

[1] Hydrodynamics Challenge Problem, Lawrence Livermore National Laboratory. Technical Report LLNL-TR-490254.

[2] Top500 list. https://www.top500.org/green500. Accessed: 2017-11.

[3] Infiniband mellanox. http://www.mellanox.com/page/infiniband_cards_overview. Accessed: 2017-11.

[4] Intel 64 and IA-32 Architectures Developer's Manual, Intel Corporation. Technical Report Volume 3B, Part 2.

[5] Intel Xeon Phi White Paper. http://software.intel.com/sites/default/files/article/373934/system-administration-for-the-intel-xeon-phi-coprocessor.pdf.

[6] Perf Counter. https://perf.wiki.kernel.org/index.php/Main_Page.

[7] Top500 list. http://www.top500.org. Accessed: 2017-11.

[8] Hydrodynamics Challenge Problem. Technical Report LLNL-TR-490254, Lawrence Livermore National Laboratory, Livermore, CA, July 2011.

[9] M. A. Abou-Of, A. H. Taha, and A. A. Sedky. Trade-off between low power and energy efficiency in benchmarking. In *2016 7th International Conference on Information and Communication Systems (ICICS)*, pages 322–326, April 2016. doi: 10.1109/IACS.2016.7476072.

[10] Jacob T Adriaens, Katherine Compton, Nam Sung Kim, and Michael J Schulte. The case for gpgpu spatial multitasking. In *High Performance Computer Architecture (HPCA), 2012 IEEE 18th International Symposium on*, pages 1–12. IEEE, 2012.

[11] Adnan Agbaria, Yosi Ben-Asher, and Ilan Newman. Communication—Processor Tradeoffs in a Limited Resources PRAM. *Algorithmica*, 34(3):276–297, 2002.

[12] S. Agwa, E. Yahya, and Y. Ismail. Power Efficient AES Core For IoT Constrained Devices Implemented in 130nm CMOS. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4, May 2017. doi: 10.1109/ISCAS.2017.8050361.

[13] Fahian Ahmed, Saddam Quirem, Bum Joo Shin, Duk Joo Son, Young Choon Woo, Byeong Kil Lee, and Wan Choi. A study of cuda acceleration and impact of data transfer overhead in heterogeneous environment. In *Workshop on Unique Chips and Systems UCAS-7*, page 16, 2012.

[14] Shoaib Akram, Jennifer B Sartor, and Lieven Eeckhout. Dvfs performance prediction for managed multithreaded applications. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 12–23, 2016.

[15] Sadaf R. Alam and Jeffrey S. Vetter. An analysis of system balance requirements for scientific applications. In *International Conference on Parallel Processing*, ICPP'06, Columbus, OH, August 2006. IEEE.

[16] Albert Alexandrov, Mihai F Ionescu, Klaus E Schauser, and Chris Scheiman. Loggp: incorporating long messages into the logp model—one step closer towards a realistic model for parallel computation. In *Proceedings of the seventh annual ACM symposium on Parallel algorithms and architectures*, pages 95–105. ACM, 1995.

[17] P. Alonso, M. F. Dolz, F. D. Igual, R. Mayo, and E. S. Quintana-Orti. Saving energy in the lu factorization with partial pivoting on multi-core processors. In *2012 20th Euromicro International Conference on Parallel, Distributed and Network-based Processing*, pages 353–358, Feb 2012. doi: 10.1109/PDP.2012.28.

[18] Pedro Alonso, Manuel F. Dolz, Francisco D. Igual, Rafael Mayo, and Enrique S. Quintana-Ortí. Dvfs-control techniques for dense linear algebra operations on multi-core processors. *Computer Science - Research and Development*, 27(4):289–298, Nov 2012. ISSN 1865-2042. doi: 10.1007/s00450-011-0188-7. URL https://doi.org/10.1007/s00450-011-0188-7.

[19] Frehiwot Melak Arega, Markus Haehnel, and Waltenegus Dargie. Dynamic power management in a heterogeneous processor architecture. In *International Conference on Architecture of Computing Systems*, pages 248–260. Springer, 2017.

[20] P Arroba, JM Moya, JL Ayala, and R Buyya. Proactive power and thermal aware optimizations for energy-efficient cloud computing. In *Design Automation and Test in Europe. DATE*, 2016.

[21] P. E. Bailey, D. K. Lowenthal, V. Ravi, B. Rountree, M. Schulz, and B. R. d. Supinski. Adaptive configuration selection for power-constrained heterogeneous systems. In *2014 43rd International Conference on Parallel Processing*, pages 371–380, Sept 2014. doi: 10.1109/ICPP.2014.46.

[22] Peter E. Bailey, Aniruddha Marathe, David K. Lowenthal, Barry Rountree, and Martin Schulz. Finding the limits of power-constrained application performance. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '15, pages 79:1–79:12, New York, NY, USA,

2015. ACM. ISBN 978-1-4503-3723-6. doi: 10.1145/2807591.2807637. URL http://doi.acm.org/10.1145/2807591.2807637.

[23] Teresa Bailey, W. Daryl Hawkins, Marvin L. Adams, Peter N. Brown, Adam J. Kunen, Michael P. Adams, Timmie Smith, Nancy Amato, and Lawrence Rauchwerger. Validation of full-domain massively parallel transport sweep algorithms. In *American Nuclear Society Winter Meeting and Nuclear Technology Expo*, Anaheim, CA, November 2014.

[24] Prasanna Balaprakash, A. Tiwari, and S. M. Wild. Multi-objective optimization of HPC kernels for performance, power, and energy. 2013.

[25] Sulieman Bani-Ahmad and Saleh Sa'adeh. Scalability of the dvfs power management technique as applied to 3-tier data center architecture in cloud computing. *Journal of Computer and Communications*, 5:69–93, 2017.

[26] M. A. S. Bari, N. Chaimov, A. M. Malik, K. A. Huck, B. Chapman, A. D. Malony, and O. Sarood. Arcs: Adaptive runtime configuration selection for power-constrained openmp applications. In *2016 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 461–470, Sept 2016. doi: 10.1109/CLUSTER.2016.39.

[27] Armin Baumker and Wolfgang Dittrich. Fully dynamic search trees for an extension of the bsp model. In *Proceedings of the eighth annual ACM symposium on Parallel algorithms and architectures*, pages 233–242. ACM, 1996.

[28] D. Bedard, M. Y. Lim, R. Fowler, and A. Porterfield. Powermon: Fine-grained and integrated power monitoring for commodity computer systems. In *Proceedings of the IEEE SoutheastCon 2010 (SoutheastCon)*, pages 479–484, March 2010. doi: 10.1109/SECON.2010.5453824.

[29] R. Begum, D. Werner, M. Hempstead, G. Prasad, and G. Challen. Energy-performance

trade-offs on energy-constrained devices with multi-component dvfs. In *2015 IEEE International Symposium on Workload Characterization*, pages 34–43, Oct 2015. doi: 10.1109/IISWC.2015.10.

[30] Rizwana Begum, Mark Hempstead, Guru Prasad Srinivasa, and Geoffrey Challen. Algorithms for cpu and dram dvfs under inefficiency constraints. In *Computer Design (ICCD), 2016 IEEE 34th International Conference on*, pages 161–168. IEEE, 2016.

[31] R. Bertran, Y. Sugawara, H. M. Jacobson, A. Buyuktosunoglu, and P. Bose. Application-level power and performance characterization and optimization on IBM Blue Gene/Q systems. *IBM Journal of Research and Development*, 57(1/2), Jan-Mar 2013.

[32] S. Bhalachandra, A. Porterfield, S. L. Olivier, and J. F. Prins. An adaptive core-specific runtime for energy efficiency. In *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 947–956, May 2017. doi: 10.1109/IPDPS.2017.114.

[33] Andrea Borghesi, Andrea Bartolini, Michele Lombardi, Michela Milano, and Luca Benini. Scheduling-based power capping in high performance computing systems. *Sustainable Computing: Informatics and Systems*, 19:1 – 13, 2018. ISSN 2210-5379. doi: https://doi.org/10.1016/j.suscom.2018.05.007. URL http://www.sciencedirect.com/science/article/pii/S2210537917302317.

[34] Kirk W Cameron and Rong Ge. Predicting and evaluating distributed communication performance. In *Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, page 43. IEEE Computer Society, 2004.

[35] Kirk W Cameron, Rong Ge, and Xian-He Sun. log_nP and log_3P: Accurate analytical

models of point-to-point communication in distributed systems. *IEEE Transactions on Computers*, 56(3), 2007.

[36] Kirk W. Cameron, Ali Anwar, Yue Cheng, Bo Li, Li Xu, Ananth Uday, Thomas Lux, Yili Hong, Layne T. Watson, and Ali R. Butt. Moana: Modeling and analyzing i/o variability in parallel system experimental design. IEEE Transactions on Parallel and Distributed Systems (TPDS). IEEE, 2019.

[37] Fabio Campi. *Power-Shaping Configurable Microprocessors for IoT Devices*, pages 3–26. Springer International Publishing, Cham, 2017. ISBN 978-3-319-42304-3. doi: 10.1007/978-3-319-42304-3_1. URL https://doi.org/10.1007/978-3-319-42304-3_1.

[38] Enrique V. Carrera, Eduardo Pinheiro, and Ricardo Bianchini. Conserving disk energy in network servers. In *Proceedings of the 17th Annual International Conference on Supercomputing*, ICS '03, pages 86–97, New York, NY, USA, 2003. ACM. ISBN 1-58113-733-8. doi: 10.1145/782814.782829. URL http://doi.acm.org/10.1145/782814.782829.

[39] Aaron Carroll and Gernot Heiser. An analysis of power consumption in a smartphone. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'10, pages 21–21, Berkeley, CA, USA, 2010. USENIX Association. URL http://dl.acm.org/citation.cfm?id=1855840.1855861.

[40] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S. H. Lee, and K. Skadron. Rodinia: A benchmark suite for heterogeneous computing. In *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, pages 44–54, Oct 2009. doi: 10.1109/IISWC.2009.5306797.

[41] Shuai Che, M. Boyer, Jiayuan Meng, D. Tarjan, J.W. Sheaffer, Sang-Ha Lee, and K. Skadron. Rodinia: A benchmark suite for heterogeneous computing. In *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, pages 44–54, 2009. doi: 10.1109/IISWC.2009.5306797.

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

[43] Kihwan Choi, Ramakrishna Soma, and Massoud Pedram. Dynamic voltage and frequency scaling based on workload decomposition. In *Proceedings of the 2004 International Symposium on Low Power Electronics and Design*, ISLPED '04, pages 174–179, New York, NY, USA, 2004. ACM. ISBN 1-58113-929-2. doi: 10.1145/1013235.1013282. URL http://doi.acm.org/10.1145/1013235.1013282.

[44] Wu chun Feng and K.W. Cameron. The Green500 list: Encouraging sustainable supercomputing. *Computer*, 40(12):50–55, 2007. ISSN 0018-9162. doi: 10.1109/MC.2007.445.

[45] P. Cicotti, A. Tiwari, and L. Carrington. Efficient speed (es): Adaptive dvfs and clock modulation for energy efficiency. In *2014 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 158–166, Sept 2014. doi: 10.1109/CLUSTER.2014.6968750.

[46] R. Cochran, C. Hankendi, A. K. Coskun, and S. Reda. Pack amp; cap: Adaptive dvfs and thread packing under power caps. In *2011 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 175–185, Dec 2011.

[47] Ryan Cochran, Can Hankendi, Ayse Coskun, and Sherief Reda. Identifying the optimal energy-efficient operating points of parallel workloads. In *Proceedings of*

the International Conference on Computer-Aided Design, ICCAD '11, pages 608–615, Piscataway, NJ, USA, 2011. IEEE Press. ISBN 978-1-4577-1398-9. URL http://dl.acm.org/citation.cfm?id=2132325.2132464.

[48] Seth Colaner. Intel's Xeon Phi Powered Beacon is World's Most Efficient Supercomputer. http://hothardware.com/News/Intels-Xeon-Phi-Powered-Beacon-is-Worlds-Most-Efficient-Supercomputer/, 2012.

[49] Richard Cole and Ofer Zajicek. The apram: Incorporating asynchrony into the pram model. In *Proceedings of the first annual ACM symposium on Parallel algorithms and architectures*, pages 169–178. ACM, 1989.

[50] Sylvain Collange, David Defour, and Arnaud Tisserand. Power consumption of GPUs from a software perspective. In *Proceedings of the 9th International Conference on Computational Science: Part I*, ICCS '09, pages 914–923, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 978-3-642-01969-2. doi: 10.1007/978-3-642-01970-8_92. URL http://dx.doi.org/10.1007/978-3-642-01970-8_92.

[51] Tim Cramer, Dirk Schmidl, Michael Klemm, and Dieter an Mey. Openmp programming on intel r xeon phi tm coprocessors: An early performance comparison. 2012.

[52] David Culler, Richard Karp, David Patterson, Abhijit Sahay, Klaus Erik Schauser, Eunice Santos, Ramesh Subramonian, and Thorsten Von Eicken. Logp: Towards a realistic model of parallel computation. In *ACM Sigplan Notices*, volume 28, pages 1–12. ACM, 1993.

[53] Matthew Curtis-Maury, James Dzierwa, Christos D. Antonopoulos, and Dimitrios S. Nikolopoulos. Online power-performance adaptation of multithreaded programs using

hardware event-based prediction. In *Proceedings of the 20th Annual International Conference on Supercomputing*, ICS '06, pages 157–166, New York, NY, USA, 2006. ACM. ISBN 1-59593-282-8. doi: 10.1145/1183401.1183426. URL http://doi.acm.org/10.1145/1183401.1183426.

[54] Matthew Curtis-Maury, Ankur Shah, Filip Blagojevic, Dimitrios S. Nikolopoulos, Bronis R. de Supinski, and Martin Schulz. Prediction models for multi-dimensional power-performance optimization on many cores. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, PACT '08, pages 250–259, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-282-5. doi: 10.1145/1454115.1454151. URL http://doi.acm.org/10.1145/1454115.1454151.

[55] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le. Rapl: Memory power estimation and capping. In *2010 ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED)*, pages 189–194, Aug 2010. doi: 10.1145/1840845.1840883.

[56] Howard David, Chris Fallin, Eugene Gorbatov, Ulf R. Hanebutte, and Onur Mutlu. Memory power management via dynamic voltage/frequency scaling. In *Proceedings of the 8th ACM International Conference on Autonomic Computing*, ICAC '11, pages 31–40, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0607-2. doi: 10.1145/1998582.1998590. URL http://doi.acm.org/10.1145/1998582.1998590.

[57] Pilar De la Torre and Clyde P Kruskal. Submachine locality in the bulk synchronous setting. In *European Conference on Parallel Processing*, pages 352–358. Springer, 1996.

[58] Daniele De Sensi, Massimo Torquati, and Marco Danelutto. A reconfiguration algorithm for power-aware parallel applications. *ACM Trans. Archit. Code Optim.*,

13(4):43:1–43:25, December 2016. ISSN 1544-3566. doi: 10.1145/3004054. URL http://doi.acm.org/10.1145/3004054.

[59] Michael Deisher, Mikhail Smelyanskiy, Brian Nickerson, VictorW. Lee, Michael Chuvelev, and Pradeep Dubey. Designing and dynamically load balancing hybrid LU for multi/many-core. *Computer Science - Research and Development*, 26(3-4):211–220, 2011. ISSN 1865-2034. doi: 10.1007/s00450-011-0169-x. URL http://dx.doi.org/10.1007/s00450-011-0169-x.

[60] Qingyuan Deng, David Meisner, Luiz Ramos, Thomas F. Wenisch, and Ricardo Bianchini. Memscale: Active low-power modes for main memory. In *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVI, pages 225–238, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0266-1. doi: 10.1145/1950365.1950392. URL http://doi.acm.org/10.1145/1950365.1950392.

[61] Qingyuan Deng, David Meisner, Abhishek Bhattacharjee, Thomas F. Wenisch, and Ricardo Bianchini. Multiscale: Memory system dvfs with multiple memory controllers. In *Proceedings of the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design*, ISLPED '12, pages 297–302, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1249-3. doi: 10.1145/2333660.2333727. URL http://doi.acm.org/10.1145/2333660.2333727.

[62] Qingyuan Deng, David Meisner, Abhishek Bhattacharjee, Thomas F. Wenisch, and Ricardo Bianchini. Coscale: Coordinating cpu and memory system dvfs in server systems. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-45, pages 143–154, Washington, DC, USA, 2012. IEEE

Computer Society. ISBN 978-0-7695-4924-8. doi: 10.1109/MICRO.2012.22. URL http://dx.doi.org/10.1109/MICRO.2012.22.

[63] V. Devadas and H. Aydin. On the interplay of voltage/frequency scaling and device power management for frame-based real-time embedded applications. *IEEE Transactions on Computers*, 61(1):31–44, Jan 2012. ISSN 0018-9340. doi: 10.1109/TC.2010.248.

[64] Gaurav Dhiman and Tajana Simunic Rosing. Dynamic voltage frequency scaling for multi-tasking systems using online learning. In *Proceedings of the 2007 International Symposium on Low Power Electronics and Design*, ISLPED '07, pages 207–212, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-709-4. doi: 10.1145/1283780.1283825. URL http://doi.acm.org/10.1145/1283780.1283825.

[65] Gaurav Dhiman, Kishore Kumar Pusukuri, and Tajana Rosing. Analysis of dynamic voltage scaling for system level energy management. *USENIX HotPower*, 8, 2008.

[66] Bruno Diniz, Dorgival Guedes, Wagner Meira, Jr., and Ricardo Bianchini. Limiting the power consumption of main memory. In *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ISCA '07, pages 290–301, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-706-3. doi: 10.1145/1250662.1250699. URL http://doi.acm.org/10.1145/1250662.1250699.

[67] K. Elangovan, I. Rodero, M. Parashar, F. Guim, and I. Hernandez. Adaptive memory power management techniques for hpc workloads. In *High Performance Computing (HiPC), 2011 18th International Conference on*, pages 1–11, Dec 2011. doi: 10.1109/HiPC.2011.6152740.

[68] Daniel A. Ellsworth, Allen D. Malony, Barry Rountree, and Martin Schulz. Pow: System-wide dynamic reallocation of limited power in hpc. In *Proceedings of the 24th*

International Symposium on High-Performance Parallel and Distributed Computing, HPDC '15, pages 145–148, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3550-8. doi: 10.1145/2749246.2749277. URL http://doi.acm.org/10.1145/2749246.2749277.

[69] Furkan Ercan, Neven Abou Gazala, and Howard David. An integrated approach to system-level cpu and memory energy efficiency on computing systems. In *Energy Aware Computing, 2012 International Conference on*, pages 1–6, Dec 2012. doi: 10.1109/ICEAC.2012.6471018.

[70] Constantinos Evangelinos, Robert Walkup, Vipin Sachdeva, Kirk E. Jordan, Hormozd Gahvari, I-Hsin Chung, Michael P. Perrone, Ligang Lu, Lurng-Kuo Liu, and Karen A. Magerlein. Determination of performance characteristics of scientific applications on IBM blue gene/q. *IBM Journal of Research and Development*, 57(1/2):9, 2013. doi: 10.1147/JRD.2012.2229901. URL http://dx.doi.org/10.1147/JRD.2012.2229901.

[71] S. Eyerman and L. Eeckhout. A counter architecture for online dvfs profitability estimation. *Computers, IEEE Transactions on*, 59(11):1576–1583, Nov 2010. ISSN 0018-9340. doi: 10.1109/TC.2010.65.

[72] Xiaobo Fan, Carla Ellis, and Alvin Lebeck. Memory controller policies for dram power management. In *Proceedings of the 2001 International Symposium on Low Power Electronics and Design*, ISLPED '01, pages 129–134, New York, NY, USA, 2001. ACM. ISBN 1-58113-371-5. doi: 10.1145/383082.383118. URL http://doi.acm.org/10.1145/383082.383118.

[73] Qiu Fang, Jun Wang, Qi Gong, and Mengxuan Song. Thermal-aware energy management of hpc data center via two-time-scale control. *IEEE Transactions on Industrial Informatics*, 2017.

[74] Michael Feldman. HP, Intel score petaflop supercomputer at DOE Lab. http://www.hpcwire.com/hpcwire/2012-09-05/hp_intel_score_petaflop_supercomputer_at_doe_lab.html, 2012.

[75] Steven Fortune and James Wyllie. Parallelism in random access machines. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 114–118. ACM, 1978.

[76] Matthew I Frank, Anant Agarwal, and Mary K Vernon. *LoPC: modeling contention in parallel algorithms*, volume 32. ACM, 1997.

[77] V. W. Freeh, T. K. Bletsch, and F. L. R. III. Scaling and packing on a chip multiprocessor. In *2007 IEEE International Parallel and Distributed Processing Symposium*, pages 1–8, March 2007. doi: 10.1109/IPDPS.2007.370539.

[78] K. Fujimoto, S. Takamaeda-Yamazaki, and Y. Nakashima. Stop the world: A lightweight runtime power-capping mechanism for fpgas. In *2016 Fourth International Symposium on Computing and Networking (CANDAR)*, pages 361–367, Nov 2016. doi: 10.1109/CANDAR.2016.0070.

[79] S. g. Kim, C. Choi, H. Eom, H. Y. Yeom, and H. Byun. Energy-centric dvfs controling method for multi-core platforms. In *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, pages 685–690, Nov 2012. doi: 10.1109/SC.Companion.2012.94.

[80] S. Ramos Garea and T. Hoefler. Modeling communication in cache-coherent SMP systems - a case-study with xeon phi. Jun. 2013. Accepted at HPDC'13.

[81] R. Ge and K. W. Cameron. Power-aware speedup. In *2007 IEEE International Parallel*

and Distributed Processing Symposium, pages 1–10, March 2007. doi: 10.1109/IPDPS.2007.370246.

[82] R. Ge, X. Feng, W. c. Feng, and K. W. Cameron. Cpu miser: A performance-directed, run-time system for power-aware clusters. In *2007 International Conference on Parallel Processing (ICPP 2007)*, pages 18–18, Sept 2007. doi: 10.1109/ICPP.2007.29.

[83] R. Ge, X. Feng, Y. He, and P. Zou. The case for cross-component power coordination on power bounded systems. In *2016 45th International Conference on Parallel Processing (ICPP)*, pages 516–525, Aug 2016. doi: 10.1109/ICPP.2016.66.

[84] R. Ge, P. Zou, and X. Feng. Application-aware power coordination on power bounded numa multicore systems. In *2017 46th International Conference on Parallel Processing (ICPP)*, pages 591–600, Aug 2017. doi: 10.1109/ICPP.2017.68.

[85] Rong Ge, Xizhou Feng, Shuaiwen Song, Hung-Ching Chang, Dong Li, and Kirk W Cameron. Powerpack: Energy profiling and analysis of high-performance systems and applications. *Parallel and Distributed Systems, IEEE Transactions on*, 21(5):658–671, 2010.

[86] Rong Ge, Xizhou Feng, Shuaiwen Song, Hung-Ching Chang, Dong Li, and Kirk W. Cameron. Powerpack: Energy profiling and analysis of high-performance systems and applications. *IEEE Transactions on Parallel and Distributed Systems*, 21(5):658–671, 2010. ISSN 1045-9219. doi: http://doi.ieeecomputersociety.org/10.1109/TPDS.2009.76.

[87] Rong Ge, Ryan Vogt, Jahangir Majumder, Arif Alam, Martin Burtscher, and Ziliang Zong. Effects of dynamic voltage and frequency scaling on a k20 gpu. In *Proceedings of the 2013 42Nd International Conference on Parallel Processing*, ICPP '13, pages 826–

833, Washington, DC, USA, 2013. IEEE Computer Society. ISBN 978-0-7695-5117-3. doi: 10.1109/ICPP.2013.98. URL http://dx.doi.org/10.1109/ICPP.2013.98.

[88] Paul J. Gemperline, James R. Long, and Vasilis G. Gregoriou. Nonlinear multivariate calibration using principal components regression and artificial neural networks. *Analytical Chemistry*, 63(20):2313–2323, 1991. doi: 10.1021/ac00020a022. URL https://doi.org/10.1021/ac00020a022.

[89] Marco E. T. Gerards and Jan Kuper. Optimal dpm and dvfs for frame-based real-time systems. *ACM Trans. Archit. Code Optim.*, 9(4):41:1–41:23, January 2013. ISSN 1544-3566. doi: 10.1145/2400682.2400700. URL http://doi.acm.org/10.1145/2400682.2400700.

[90] Neha Gholkar, Frank Mueller, Barry Rountree, and Aniruddha Marathe. Pshifter: Feedback-based dynamic power shifting within hpc jobs for performance. In *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '18, pages 106–117, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5785-2. doi: 10.1145/3208040.3208047. URL http://doi.acm.org/10.1145/3208040.3208047.

[91] Chris Gregg, Jonathan Dorn, Kim M Hazelwood, and Kevin Skadron. Fine-grained resource sharing for concurrent gpgpu kernels. In *HotPar*, 2012.

[92] Marisabel Guevara, Chris Gregg, Kim Hazelwood, and Kevin Skadron. Enabling task parallelism in the cuda scheduler. In *Workshop on Programming Models for Emerging Architectures*, volume 9, 2009.

[93] Carla Guillen, Carmen Navarrete, David Brayford, Wolfram Hesse, and Matthias Brehm. Energy model derivation for the dvfs automatic tuning plugin: tuning energy and power related tuning objectives. *Computing*, 99(8):747–764, Aug 2017.

ISSN 1436-5057. doi: 10.1007/s00607-016-0536-3. URL https://doi.org/10.1007/s00607-016-0536-3.

[94] Nagendra Gulur, Mahesh Mehendale, Raman Manikantan, and Ramaswamy Govindarajan. Anatomy: An analytical model of memory system performance. In *The 2014 ACM International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '14, pages 505–517, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2789-3. doi: 10.1145/2591971.2591995. URL http://doi.acm.org/10.1145/2591971.2591995.

[95] Sudhanva Gurumurthi, Anand Sivasubramaniam, Mahmut Kandemir, and Hubertus Franke. Drpm: Dynamic speed control for power management in server class disks. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, ISCA '03, pages 169–181, New York, NY, USA, 2003. ACM. ISBN 0-7695-1945-8. doi: 10.1145/859618.859638. URL http://doi.acm.org/10.1145/859618.859638.

[96] Heather Hanson and Karthick Rajamani. What computer architects need to know about memory throttling. In *Proceedings of the 2010 International Conference on Computer Architecture*, ISCA'10, pages 233–242, Berlin, Heidelberg, 2012. Springer-Verlag. ISBN 978-3-642-24321-9. doi: 10.1007/978-3-642-24322-6_20. URL http://dx.doi.org/10.1007/978-3-642-24322-6_20.

[97] Van Emden Henson and Ulrike Meier Yang. Boomeramg: a parallel algebraic multigrid solver and preconditioner. *Applied Numerical Mathematics*, 41:155–177, 2000.

[98] Torsten Hoefler, Torsten Mehlan, Frank Mietke, and Wolfgang Rehm. Logfp-a model for small messages in infiniband. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, pages 6–pp. IEEE, 2006.

[99] Sunpyo Hong and Hyesoon Kim. An integrated GPU power and performance model. In *Proceedings of the 37th annual international symposium on Computer architecture*, ISCA '10, pages 280–289, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0053-7. doi: 10.1145/1815961.1815998. URL http://doi.acm.org/10.1145/1815961.1815998.

[100] J.C. Hoskins and D.M. Himmelblau. Process control via artificial neural networks and reinforcement learning. *Computers and Chemical Engineering*, 16(4):241 – 251, 1992. ISSN 0098-1354. doi: https://doi.org/10.1016/0098-1354(92)80045-B. URL http://www.sciencedirect.com/science/article/pii/009813549280045B. Neutral network applications in chemical engineering.

[101] K. Hou, H. Wang, and W. c. Feng. Delivering Parallel Programmability to the Masses via the Intel MIC Ecosystem: A Case Study. In *2014 43rd International Conference on Parallel Processing Workshops*, pages 273–282, Sept 2014.

[102] Zhigang Hu, Alper Buyuktosunoglu, Viji Srinivasan, Victor Zyuban, Hans Jacobson, and Pradip Bose. Microarchitectural techniques for power gating of execution units. In *Proceedings of the 2004 International Symposium on Low Power Electronics and Design*, ISLPED '04, pages 32–37, New York, NY, USA, 2004. ACM. ISBN 1-58113-929-2. doi: 10.1145/1013235.1013249. URL http://doi.acm.org/10.1145/1013235.1013249.

[103] Hai Huang, Kang G. Shin, Charles Lefurgy, and Tom Keller. Improving energy efficiency by making dram less randomly accessed. In *Proceedings of the 2005 International Symposium on Low Power Electronics and Design*, ISLPED '05, pages 393–398, New York, NY, USA, 2005. ACM. ISBN 1-59593-137-6. doi: 10.1145/1077603.1077696. URL http://doi.acm.org/10.1145/1077603.1077696.

[104] I. Hur and C. Lin. A comprehensive approach to dram power management. In *2008 IEEE 14th International Symposium on High Performance Computer Architecture*, pages 305–316, Feb 2008. doi: 10.1109/HPCA.2008.4658648.

[105] Shadi Ibrahim, Tien-Dat Phan, Alexandra Carpen-Amarie, Houssem-Eddine Chihoub, Diana Moise, and Gabriel Antoniu. Governing energy consumption in hadoop through cpu frequency scaling: An analysis. *Future Generation Computer Systems*, 54(Supplement C):219 – 232, 2016. ISSN 0167-739X. doi: https://doi.org/10.1016/j.future.2015.01.005. URL http://www.sciencedirect.com/science/article/pii/S0167739X15000060.

[106] Shadi Ibrahim, Tien-Dat Phan, Alexandra Carpen-Amarie, Houssem-Eddine Chihoub, Diana Moise, and Gabriel Antoniu. Governing energy consumption in hadoop through {CPU} frequency scaling: An analysis. *Future Generation Computer Systems*, 54:219 – 232, 2016. ISSN 0167-739X. doi: http://dx.doi.org/10.1016/j.future.2015.01.005. URL http://www.sciencedirect.com/science/article/pii/S0167739X15000060.

[107] S. Imamura, H. Sasaki, K. Inoue, and D. S. Nikolopoulos. Power-capped dvfs and thread allocation with ann models on modern numa systems. In *2014 IEEE 32nd International Conference on Computer Design (ICCD)*, pages 324–331, Oct 2014. doi: 10.1109/ICCD.2014.6974701.

[108] Canturk Isci, Alper Buyuktosunoglu, Chen-Yong Cher, Pradip Bose, and Margaret Martonosi. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 39, pages 347–358, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2732-9. doi: 10.1109/MICRO.2006.8. URL https://doi.org/10.1109/MICRO.2006.8.

[109] H. Jacobson, P. Bose, Zhigang Hu, A. Buyuktosunoglu, V. Zyuban, R. Eickemeyer, L. Eisen, J. Griswell, D. Logan, Balaram Sinharoy, and J. Tendler. Stretching the limits of clock-gating efficiency in server-class processors. In *11th International Symposium on High-Performance Computer Architecture*, pages 238–242, Feb 2005. doi: 10.1109/HPCA.2005.33.

[110] J. Jang and M. Park. Dram frequency scaling for energy efficiency based on memory usage. In *2017 IEEE International Conference on Consumer Electronics (ICCE)*, pages 308–309, Jan 2017. doi: 10.1109/ICCE.2017.7889331.

[111] Q. Jiao, M. Lu, H. P. Huynh, and T. Mitra. Improving gpgpu energy-efficiency through concurrent kernel execution and dvfs. In *2015 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pages 1–11, Feb 2015. doi: 10.1109/CGO.2015.7054182.

[112] Qing Jiao, Mian Lu, Huynh Phung Huynh, and Tulika Mitra. Improving gpgpu energy-efficiency through concurrent kernel execution and dvfs. In *Code Generation and Optimization (CGO), 2015 IEEE/ACM International Symposium on*, pages 1–11. IEEE, 2015.

[113] Y. Jiao, H. Lin, P. Balaji, and W. Feng. Power and performance characterization of computational kernels on the gpu. In *Green Computing and Communications (GreenCom), 2010 IEEE/ACM Int'l Conference on Int'l Conference on Cyber, Physical and Social Computing (CPSCom)*, pages 221–228, Dec 2010. doi: 10.1109/GreenCom-CPSCom.2010.143.

[114] R. Colin Johnson. Department of Energy Chooses Xeon Phi. http://goparallel.sourceforge.net/department-of-energy-chooses-xeon-phi/, 2013.

[115] Ian Karlin and Mike Collette. Strong scaling bottleneck identification and mitigation in Ares. In *Nuclear Explosives Code Development Conference*, NECDC'14, Los Alamos, NM, October 2014.

[116] Ian Karlin, Jim McGraw, Jeff Keasler, and Bert Still. Tuning the LULESH mini-app for current and future hardware. In *Nuclear Explosive Code Development Conference*, NECDC'12, Livermore, CA, October 2012.

[117] Georgios Keramidas, Vasileios Spiliopoulos, and Stefanos Kaxiras. Interval-based models for run-time dvfs orchestration in superscalar processors. In *Proceedings of the 7th ACM International Conference on Computing Frontiers*, CF '10, pages 287–296, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0044-5. doi: 10.1145/1787275.1787338. URL http://doi.acm.org/10.1145/1787275.1787338.

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

[119] Y. G. Kim, M. Kim, and S. W. Chung. Enhancing energy efficiency of multimedia applications in heterogeneous mobile multi-core processors. *IEEE Transactions on Computers*, 66(11):1878–1889, Nov 2017. ISSN 0018-9340. doi: 10.1109/TC.2017.2710317.

[120] T. Komoda, S. Hayashi, T. Nakada, S. Miwa, and H. Nakamura. Power capping of cpu-gpu heterogeneous systems through coordinating dvfs and task mapping. In *2013 IEEE 31st International Conference on Computer Design (ICCD)*, pages 349–356, Oct 2013. doi: 10.1109/ICCD.2013.6657064.

[121] F. Kong, Y. Wang, Q. Deng, and W. Yi. Minimizing multi-resource energy for real-time systems with discrete operation modes. In *2010 22nd Euromicro Conference on Real-Time Systems*, pages 113–122, July 2010. doi: 10.1109/ECRTS.2010.18.

[122] S. H. Langer, A. Bhatele, and C. H. Still. pf3d simulations of laser-plasma interactions in national ignition facility experiments. *Computing in Science Engineering*, 16(6): 42–50, Nov 2014. ISSN 1521-9615. doi: 10.1109/MCSE.2014.79.

[123] Steven H. Langer, Abhinav Bhatele, and Charles H. Still. pF3D simulations of laser-plasma interactions in National Ignition Facility experiments. *Computing in Science & Engineering*, 16(6):42–50, Nov 2014.

[124] J. H. Laros, P. Pokorny, and D. DeBonis. PowerInsight – a commodity power measurement capability. In *International Green Computing Conference*, June 2013. doi: 10.1109/IGCC.2013.6604485.

[125] Jungseob Lee and Nam Sung Kim. Optimizing throughput of power- and thermal-constrained multicore processors using dvfs and per-core power-gating. In *2009 46th ACM/IEEE Design Automation Conference*, pages 47–50, July 2009.

[126] Jungseob Lee, Vijay Sathisha, Michael Schulte, Katherine Compton, and Nam Sung Kim. Improving throughput of power-constrained gpus using dynamic voltage/frequency and core scaling. In *Proceedings of the 2011 International Conference on Parallel Architectures and Compilation Techniques*, PACT '11, pages 111–120, Washington, DC, USA, 2011. IEEE Computer Society. ISBN 978-0-7695-4566-0. doi: 10.1109/PACT.2011.17. URL http://dx.doi.org/10.1109/PACT.2011.17.

[127] Edgar A. León and Ian Karlin. Characterizing the impact of program optimizations on power and energy for explicit hydrodynamics. In *International Parallel & Distributed*

Processing Symposium; Workshop on High-Performance, Power-Aware Computing, HPPAC'14, Phoenix, AZ, May 2014. IEEE.

[128] Edgar A. León, Ian Karlin, and Ryan E. Grant. Optimizing explicit hydrodynamics for power, energy, and performance. In *International Conference on Cluster Computing*, Cluster'15, Chicago, IL, September 2015. IEEE.

[129] Edgar A. Leon, Ian Karlin, Abhinav Bhatele, Steven H. Langer, Chris Chambreau, Louis H. Howell, Trent D'Hooge, and Matthew L. Leininger. Characterizing parallel scientific applications on commodity clusters: An empirical study of a tapered fat-tree. In *International Conference for High Performance Computing, Networking, Storage and Analysis*, SC'16, Salt Lake City, UT, November 2016. IEEE/ACM.

[130] Edgar A. León, Ian Karlin, Ryan E. Grant, and Matthew Dosanjh. Program optimizations: The interplay between power, performance, and energy. *Parallel Computing*, 58:56–75, October 2016. URL http://dx.doi.org/10.1016/j.parco.2016.05.004.

[131] E. A. León, I. Karlin, and R. E. Grant. Optimizing explicit hydrodynamics for power, energy, and performance. In *2015 IEEE International Conference on Cluster Computing*, pages 11–21, Sept 2015. doi: 10.1109/CLUSTER.2015.12.

[132] B. Li, H. Chang, S. Song, C. Su, T. Meyer, J. Mooring, and K. W. Cameron. The power-performance tradeoffs of the intel xeon phi on hpc applications. In *2014 IEEE International Parallel Distributed Processing Symposium Workshops*, pages 1448–1456, May 2014. doi: 10.1109/IPDPSW.2014.162.

[133] Bo Li and Edgar A. León. Memory throttling on bg/q: A case study with explicit hydrodynamics. In *6th Workshop on Power-Aware Computing and Systems (HotPower 14)*, Broomfield, CO, October 2014. USENIX Association. URL https://www.usenix.org/conference/hotpower14/workshop-program/presentation/li.

[134] D. Li, B. R. de Supinski, M. Schulz, D. S. Nikolopoulos, and K. W. Cameron. Strategies for energy-efficient resource management of hybrid programming models. *IEEE Transactions on Parallel and Distributed Systems*, 24(1):144–157, Jan 2013. ISSN 1045-9219. doi: 10.1109/TPDS.2012.95.

[135] Dong Li, B.R. de Supinski, M. Schulz, K. Cameron, and D.S. Nikolopoulos. Hybrid mpi/openmp power-aware computing. In *Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, pages 1–12, April 2010. doi: 10.1109/IPDPS.2010.5470463.

[136] Yun Liang, Huynh Phung Huynh, Kyle Rupnow, Rick Siow Mong Goh, and Deming Chen. Efficient gpu spatial-temporal multitasking. *IEEE Transactions on Parallel and Distributed Systems*, 26(3):748–760, 2015.

[137] Y. Liu, G. Cox, Q. Deng, S. C. Draper, and R. Bianchini. Fastcap: An efficient and fair algorithm for power capping in many-core systems. In *2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 57–68, April 2016. doi: 10.1109/ISPASS.2016.7482074.

[138] Yanpei Liu, Stark C. Draper, and Nam Sung Kim. Sleepscale: Runtime joint speed scaling and sleep states management for power efficient data centers. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, ISCA '14, pages 313–324, Piscataway, NJ, USA, 2014. IEEE Press. ISBN 978-1-4799-4394-4. URL http://dl.acm.org/citation.cfm?id=2665671.2665719.

[139] Charles Lively, Xingfu Wu, Valerie Taylor, Shirley Moore, Hung-Ching Chang, Chun-Yi Su, and Kirk Cameron. Power-aware predictive models of hybrid (mpi/openmp) scientific applications on multicore systems. *Computer Science - Research and Devel-*

opment, 27(4):245–253, Nov 2012. ISSN 1865-2042. doi: 10.1007/s00450-011-0190-0. URL https://doi.org/10.1007/s00450-011-0190-0.

[140] Teng Lu, Partha Pratim Pande, and Behrooz Shirazi. A dynamic, compiler guided dvfs mechanism to achieve energy-efficiency in multi-core processors. *Sustainable Computing: Informatics and Systems*, 12(Supplement C):1 – 9, 2016. ISSN 2210-5379. doi: https://doi.org/10.1016/j.suscom.2016.04.003. URL http://www.sciencedirect.com/science/article/pii/S2210537916300622.

[141] Y. Lu, D. Wu, B. He, X. Tang, J. Xu, and M. Guo. Rank-aware dynamic migrations and adaptive demotions for dram power management. *IEEE Transactions on Computers*, 65(1):187–202, Jan 2016. ISSN 0018-9340. doi: 10.1109/TC.2015.2409847.

[142] Yanchao Lu, Bingsheng He, Xueyan Tang, and Minyi Guo. Synergy of dynamic frequency scaling and demotion on dram power management: Models and optimizations. *IEEE Transactions on Computers*, 64(8):2367–2381, 2015.

[143] Anita Lungu, Pradip Bose, Alper Buyuktosunoglu, and Daniel J. Sorin. Dynamic power gating with quality guarantees. In *Proceedings of the 2009 ACM/IEEE International Symposium on Low Power Electronics and Design*, ISLPED '09, pages 377–382, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-684-7. doi: 10.1145/1594233.1594331. URL http://doi.acm.org/10.1145/1594233.1594331.

[144] Kai Ma, Yunhao Bai, Xiaorui Wang, Wei Chen, and Xue Li. Energy conservation for gpu–cpu architectures with dynamic workload division and frequency scaling. *Sustainable Computing: Informatics and Systems*, 12(Supplement C):21 – 33, 2016. ISSN 2210-5379. doi: https://doi.org/10.1016/j.suscom.2016.05.002. URL http://www.sciencedirect.com/science/article/pii/S2210537916300683.

[145] Aniruddha Marathe, Peter E. Bailey, David K. Lowenthal, Barry Rountree, Martin Schulz, and Bronis R. de Supinski. *A Run-Time System for Power-Constrained HPC Applications*, pages 394–408. Springer International Publishing, Cham, 2015. ISBN 978-3-319-20119-1. doi: 10.1007/978-3-319-20119-1_28. URL https://doi.org/10.1007/978-3-319-20119-1_28.

[146] José Luis March, Salvador Petit, Julio Sahuquillo, Houcine Hassan, and José Duato. Dynamic wcet estimation for real-time multicore embedded systems supporting dvfs. In *High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC, CSS, ICESS), 2014 IEEE Intl Conf on*, pages 27–33. IEEE, 2014.

[147] Xinxin Mei, Ling Sing Yung, Kaiyong Zhao, and Xiaowen Chu. A measurement study of gpu dvfs on energy conservation. In *Proceedings of the Workshop on Power-Aware Computing and Systems*, HotPower '13, pages 10:1–10:5, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2458-8. doi: 10.1145/2525526.2525852. URL http://doi.acm.org/10.1145/2525526.2525852.

[148] R. Miftakhutdinov, E. Ebrahimi, and Y.N. Patt. Predicting performance impact of dvfs for realistic memory systems. In *Microarchitecture (MICRO), 2012 45th Annual IEEE/ACM International Symposium on*, pages 155–165, Dec 2012. doi: 10.1109/MICRO.2012.23.

[149] Goldi Misra, Nisha Kurkure, Abhishek Das, Manjunatha Valmiki, Shweta Das, and Abhinav Gupta. Evaluation of rodinia codes on intel xeon phi. In *Intelligent Systems Modelling & Simulation (ISMS), 2013 4th International Conference on*, pages 415–419. IEEE, 2013.

[150] Csaba Andras Moritz and Matthew I Frank. Logpc: Modeling network contention in

message-passing programs. In *ACM SIGMETRICS Performance Evaluation Review*, volume 26, pages 254–263. ACM, 1998.

[151] Rajib Nath and Dean Tullsen. The crisp performance model for dynamic voltage and frequency scaling in a gpgpu. In *Proceedings of the 48th International Symposium on Microarchitecture*, MICRO-48, pages 281–293, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-4034-2. doi: 10.1145/2830772.2830826. URL http://doi.acm.org/10.1145/2830772.2830826.

[152] Rajiv Nishtala, Marc Gonzalez Tallada, and Xavier Martorell. A methodology to build models and predict performance-power in cmps. In *Parallel Processing Workshops (ICPPW), 2015 44th International Conference on*, pages 193–202. IEEE, 2015.

[153] Akira Nukada, Yutaka Maruyama, and Satoshi Matsuoka. High performance 3-D FFT using multiple CUDA GPUs. In *Proceedings of the 5th Annual Workshop on General Purpose Processing with Graphics Processing Units*, GPGPU-5, pages 57–63, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1233-2. doi: 10.1145/2159430.2159437. URL http://doi.acm.org/10.1145/2159430.2159437.

[154] ORNL. The Scalable Heterogeneous Computing Benchmark Suite. https://github.com/spaffy/shoc, 2013.

[155] Sreepathi Pai, Matthew J. Thazhuthaveetil, and R. Govindarajan. Improving gpgpu concurrency with elastic kernels. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '13, pages 407–418, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1870-9. doi: 10.1145/2451116.2451160. URL http://doi.acm.org/10.1145/2451116.2451160.

[156] J. Park and H. Cha. Aggressive voltage and temperature control for power saving in mobile application processors. *IEEE Transactions on Mobile Computing*, PP(99):1–1, 2017. ISSN 1536-1233. doi: 10.1109/TMC.2017.2762670.

[157] Jongsoo Park, Ping Tak Peter Tang, Mikhail Smelyanskiy, Daehyun Kim, and Thomas Benson. Efficient backprojection-based synthetic aperture radar computation with many-core processors. In *Proceedings of the 2012 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 1–11, Salt Lake City, Utah, USA, 2012. IEEE Computer Society. ISBN 978-1-4673-0806-9. doi: 10.1109/SC.2012.53. URL http://dx.doi.org/10.1109/SC.2012.53.

[158] Indrani Paul, Wei Huang, Manish Arora, and Sudhakar Yalamanchili. Harmonia: Balancing compute and memory power in high-performance gpus. In *Proceedings of the 42Nd Annual International Symposium on Computer Architecture*, ISCA '15, pages 54–65, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3402-0. doi: 10.1145/2749469.2750404. URL http://doi.acm.org/10.1145/2749469.2750404.

[159] K. Pedretti, R. E. Grant, J. H. Laros III, M. Levenhagen, S. L. Olivier, L. Ward, and A. J. Younge. A comparison of power management mechanisms: P-states vs. node-level power cap control. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 725–729, May 2018. doi: 10.1109/IPDPSW.2018.00117.

[160] A.J. Pena and S.R. Alam. Evaluation of inter- and intra-node data transfer efficiencies between GPU devices and their impact on scalable applications. In *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, pages 144–151, 2013. doi: 10.1109/CCGrid.2013.15.

[161] SJ Pennycook, CJ Hughes, M Smelyanskiy, and SA Jarvis. Exploring SIMD for molec-

ular dynamics, using Intel Xeon processors and Intel Xeon Phi coprocessors. In *IEEE International Parallel & Distributed Processing Symposium*, IPDPS '13, Boston, MA, USA, 2013. IEEE Computer Society.

[162] Joshua Peraza, Ananta Tiwari, Michael Laurenzano, Laura Carrington, and Allan Snavely. Pmac's green queue: a framework for selecting energy optimal dvfs configurations in large scale mpi applications. *Concurrency and Computation: Practice and Experience*, 28(2):211–231, 2016. ISSN 1532-0634. doi: 10.1002/cpe.3184. URL http://dx.doi.org/10.1002/cpe.3184.

[163] A. Prakash, H. Amrouch, M. Shafique, T. Mitra, and J. Henkel. Improving mobile gaming performance through cooperative cpu-gpu thermal management. In *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2016. doi: 10.1145/2897937.2898031.

[164] K. Rao, J. Wang, S. Yalamanchili, Y. Wardi, and Y. Handong. Application-specific performance-aware energy optimization on android mobile devices. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 169–180, Feb 2017. doi: 10.1109/HPCA.2017.32.

[165] Pier Giorgio Raponi, Fabrizio Petrini, Robert Walkup, and Fabio Checconi. Characterization of the communication patterns of scientific applications on Blue Gene/P. In *International Workshop on System Management Techniques, Processes, and Services*, SMTPS'11, Anchorage, AK, May 2011.

[166] F. Diniz Rossi, M. Storch, I. de Oliveira, and C. A. F. De Rose. Modeling power consumption for dvfs policies. In *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1879–1882, May 2015. doi: 10.1109/ISCAS.2015.7169024.

[167] Fábio D Rossi, Miguel G Xavier, César AF De Rose, Rodrigo N Calheiros, and Rajkumar Buyya. E-eco: Performance-aware energy-efficient cloud data center orchestration. *Journal of Network and Computer Applications*, 78:83–96, 2017.

[168] B. Rountree, D.K. Lowenthal, M. Schulz, and B.R. de Supinski. Practical performance prediction under dynamic voltage frequency scaling. In *Green Computing Conference and Workshops (IGCC), 2011 International*, pages 1–8, July 2011. doi: 10.1109/IGCC.2011.6008553.

[169] Barry Rountree, David K. Lownenthal, Bronis R. de Supinski, Martin Schulz, Vincent W. Freeh, and Tyler Bletsch. Adagio: Making dvs practical for complex hpc applications. In *Proceedings of the 23rd International Conference on Supercomputing*, ICS '09, pages 460–469, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-498-0. doi: 10.1145/1542275.1542340. URL http://doi.acm.org/10.1145/1542275.1542340.

[170] O. Sarood, A. Langer, L. Kalé, B. Rountree, and B. de Supinski. Optimizing power allocation to cpu and memory subsystems in overprovisioned hpc systems. In *2013 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 1–8, Sept 2013. doi: 10.1109/CLUSTER.2013.6702684.

[171] Erik Saule and Umit V Catalyurek. An early evaluation of the scalability of graph algorithms on the Intel MIC architecture. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International*, pages 1629–1639. IEEE, 2012.

[172] Erik Saule, Kamer Kaya, and Ümit V. Çatalyürek. Performance evaluation of sparse matrix multiplication kernels on Intel Xeon Phi. *CoRR*, abs/1302.1078, 2013.

[173] Robert Schöne, Thomas Ilsche, Mario Bielert, Daniel Molka, and Daniel Hackenberg. Software controlled clock modulation for energy efficiency optimization on intel pro-

cessors. In *Proceedings of the 4th International Workshop on Energy Efficient Supercomputing*, E2SC '16, pages 69–76, Piscataway, NJ, USA, 2016. IEEE Press. ISBN 978-1-5090-3856-5. doi: 10.1109/E2SC.2016.15. URL https://doi.org/10.1109/E2SC.2016.15.

[174] Greg Semeraro, Grigorios Magklis, Rajeev Balasubramonian, David H. Albonesi, Sandhya Dwarkadas, and Michael L. Scott. Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling. In *Proceedings of the 8th International Symposium on High-Performance Computer Architecture*, HPCA '02, pages 29–, Washington, DC, USA, 2002. IEEE Computer Society. URL http://dl.acm.org/citation.cfm?id=874076.876477.

[175] R. A. Shafik, S. Yang, A. Das, L. A. Maeda-Nunez, G. V. Merrett, and B. M. Al-Hashimi. Learning transfer-based adaptive energy minimization in embedded systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35 (6):877–890, June 2016. ISSN 0278-0070. doi: 10.1109/TCAD.2015.2481867.

[176] Yakun Shao and David Brooks. Energy characterization and instruction-level energy model of Intel's Xeon Phi processor. In *International Symposium on Low Power Electronics and Design (ISLPED), 2013*, 2013.

[177] Jaewoong Sim, Aniruddha Dasgupta, Hyesoon Kim, and Richard Vuduc. A performance analysis framework for identifying potential benefits in GPGPU applications. In *Proceedings of the 17th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming*, PPoPP '12, pages 11–22, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1160-1. doi: 10.1145/2145816.2145819. URL http://doi.acm.org/10.1145/2145816.2145819.

[178] S. P. T. Srinivasan and U. Bellur. Watttime: Novel system power model and completion

time model for dvfs-enabled servers. In *2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS)*, pages 448–455, Dec 2015. doi: 10.1109/ICPADS.2015.63.

[179] B. Su, J. Gu, L. Shen, W. Huang, J. L. Greathouse, and Z. Wang. Ppep: Online performance, power, and energy prediction framework and dvfs space exploration. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 445–457, Dec 2014. doi: 10.1109/MICRO.2014.17.

[180] Bo Su, Joseph L. Greathouse, Junli Gu, Michael Boyer, Li Shen, and Zhiying Wang. Implementing a leading loads performance predictor on commodity processors. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, Philadelphia, PA, June 2014. USENIX Association. URL https://www.usenix.org/conference/atc14/technical-sessions/presentation/su.

[181] K. Sudan, K. Rajamani, W. Huang, and J. B. Carter. Tiered memory: An iso-power memory architecture to address the memory power wall. *IEEE Transactions on Computers*, 61(12):1697–1710, Dec 2012. ISSN 0018-9340. doi: 10.1109/TC.2012.119.

[182] M. Aater Suleman, Moinuddin K. Qureshi, and Yale N. Patt. Feedback-driven threading: Power-efficient and high-performance execution of multi-threaded workloads on cmps. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XIII, pages 277–286, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-958-6. doi: 10.1145/1346281.1346317. URL http://doi.acm.org/10.1145/1346281.1346317.

[183] Vaibhav Sundriyal and Masha Sosonkina. Joint frequency scaling of processor and dram. *The Journal of Supercomputing*, 72(4):1549–1569, 2016. ISSN

1573-0484. doi: 10.1007/s11227-016-1680-4. URL http://dx.doi.org/10.1007/s11227-016-1680-4.

[184] Vaibhav Sundriyal, Ellie Fought, Masha Sosonkina, and Theresa L. Windus. Power profiling and evaluating the effect of frequency scaling on nwchem. In *Proceedings of the 24th High Performance Computing Symposium*, HPC '16, pages 19:1–19:8, San Diego, CA, USA, 2016. Society for Computer Simulation International. ISBN 978-1-5108-2318-1. doi: 10.22360/SpringSim.2016.HPC.044. URL https://doi.org/10.22360/SpringSim.2016.HPC.044.

[185] Dan Terpstra, Heike Jagode, Haihang You, and Jack Dongarra. Collecting performance data with papi-c. In Matthias S. Müller, Michael M. Resch, Alexander Schulz, and Wolfgang E. Nagel, editors, *Tools for High Performance Computing 2009*, pages 157–173, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-11261-4.

[186] K. Tsuzuku and T. Endo. Power capping of cpu-gpu heterogeneous systems using power and performance models. In *2015 International Conference on Smart Cities and Green ICT Systems (SMARTGREENS)*, pages 1–8, May 2015.

[187] Leslie G Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990.

[188] Augusto Vega, Alper Buyuktosunoglu, Heather Hanson, Pradip Bose, and Srinivasan Ramani. Crank it up or dial it down: Coordinated multiprocessor frequency and folding control. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-46, pages 210–221, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2638-4. doi: 10.1145/2540708.2540727. URL http://doi.acm.org/10.1145/2540708.2540727.

[189] Akshay Venkatesh, Abhinav Vishnu, Khaled Hamidouche, Nathan Tallent, Dhabaleswar (DK) Panda, Darren Kerbyson, and Adolfy Hoisie. A case for application-oblivious energy-efficient mpi runtime. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '15, pages 29:1–29:12, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3723-6. doi: 10.1145/2807591.2807658. URL http://doi.acm.org/10.1145/2807591.2807658.

[190] Jeffrey S. Vetter and Frank Mueller. Communication characteristics of large-scale scientific applications for contemporary cluster architectures. In *International Parallel and Distributed Processing Symposium*, IPDPS'02, Fort Lauderdale, FL, April 2002. IEEE.

[191] Guibin Wang, YiSong Lin, and Wei Yi. Kernel fusion: An effective method for better power efficiency on multithreaded gpu. In *Proceedings of the 2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing*, pages 344–350. IEEE Computer Society, 2010.

[192] W. Wang, A. Porterfield, J. Cavazos, and S. Bhalachandra. Using per-loop cpu clock modulation for energy efficiency in openmp applications. In *2015 44th International Conference on Parallel Processing*, pages 629–638, Sept 2015. doi: 10.1109/ICPP.2015.72.

[193] Samuel Williams, Dhiraj D. Kalamkar, Amik Singh, Anand M. Deshpande, Brian Van Straalen, Mikhail Smelyanskiy, Ann Almgren, Pradeep Dubey, John Shalf, and Leonid Oliker. Optimization of geometric multigrid for emerging multi- and many-core processors. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 96:1–96:11, Salt Lake

City, Utah, USA, 2012. IEEE Computer Society Press. ISBN 978-1-4673-0804-5. URL http://dl.acm.org/citation.cfm?id=2388996.2389126.

[194] T. Wirtz and R. Ge. Improving mapreduce energy efficiency for computation intensive workloads. In *2011 International Green Computing Conference and Workshops*, pages 1–8, July 2011. doi: 10.1109/IGCC.2011.6008564.

[195] H. Wong, M.-M. Papadopoulou, M. Sadooghi-Alvandi, and A. Moshovos. Demystifying GPU microarchitecture through microbenchmarking. In *Performance Analysis of Systems Software (ISPASS), 2010 IEEE International Symposium on*, pages 235–246, 2010. doi: 10.1109/ISPASS.2010.5452013.

[196] Chia-Ming Wu, Ruay-Shiung Chang, and Hsin-Yu Chan. A green energy-efficient scheduling algorithm using the dvfs technique for cloud datacenters. *Future Generation Computer Systems*, 37(Supplement C):141 – 147, 2014. ISSN 0167-739X. doi: https://doi.org/10.1016/j.future.2013.06.009. URL http://www.sciencedirect.com/science/article/pii/S0167739X13001234. Special Section: Innovative Methods and Algorithms for Advanced Data-Intensive Computing Special Section: Semantics, Intelligent processing and services for big data Special Section: Advances in Data-Intensive Modelling and Simulation Special Section: Hybrid Intelligence for Growing Internet and its Applications.

[197] D. Wu, B. He, X. Tang, J. Xu, and M. Guo. Ramzzz: Rank-aware dram power management with dynamic migrations and demotions. In *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, pages 1–11, Nov 2012. doi: 10.1109/SC.2012.99.

[198] Qiang Wu, Philo Juang, Margaret Martonosi, and Douglas W. Clark. Formal online methods for voltage/frequency control in multiple clock domain microprocessors. In

Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS XI, pages 248–259, New York, NY, USA, 2004. ACM. ISBN 1-58113-804-0. doi: 10.1145/1024393.1024423. URL http://doi.acm.org/10.1145/1024393.1024423.

[199] Qiang Wu, Philo Juang, Margaret Martonosi, and Douglas W. Clark. Voltage and frequency control with adaptive reaction time in multiple-clock-domain processors. In *Proceedings of the 11th International Symposium on High-Performance Computer Architecture*, HPCA '05, pages 178–189, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2275-0. doi: 10.1109/HPCA.2005.43. URL https://doi.org/10.1109/HPCA.2005.43.

[200] Y. Wu, J. Nunez-Yanez, R. Woods, and D. S. Nikolopoulos. Power modelling and capping for heterogeneous arm/fpga socs. In *2014 International Conference on Field-Programmable Technology (FPT)*, pages 231–234, Dec 2014. doi: 10.1109/FPT.2014.7082782.

[201] Y. Wu, D. S. Nikolopoulos, and R. Woods. Runtime support for adaptive power capping on heterogeneous socs. In *2016 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS)*, pages 71–78, July 2016. doi: 10.1109/SAMOS.2016.7818333.

[202] D. You and K. . Chung. Dynamic voltage and frequency scaling framework for low-power embedded gpus. *Electronics Letters*, 48(21):1333–1334, October 2012. ISSN 0013-5194. doi: 10.1049/el.2012.2624.

[203] Da Zhang, Hao Wang, Kaixi Hou, Jing Zhang, and Wu chun Feng. pDindel: Accelerating InDel Detection on a Multicore CPU Architecture with SIMD. In *2015 IEEE*

5th International Conference on Computational Advances in Bio and Medical Sciences (ICCABS), pages 1–6, Oct 2015.

[204] Huazhe Zhang and H Hoffman. A quantitative evaluation of the rapl power control system. *Feedback Computing*, 2015.

[205] Xiao Zhang, Sandhya Dwarkadas, and Kai Shen. Hardware execution throttling for multi-core resource management. In *Proceedings of the 2009 Conference on USENIX Annual Technical Conference*, USENIX'09, pages 23–23, Berkeley, CA, USA, 2009. USENIX Association. URL http://dl.acm.org/citation.cfm?id=1855807.1855830.

[206] H. Zheng and Z. Zhu. Power and performance trade-offs in contemporary dram system designs for multicore processors. *IEEE Transactions on Computers*, 59(8):1033–1046, Aug 2010. ISSN 0018-9340. doi: 10.1109/TC.2010.108.

[207] Jianlong Zhong and Bingsheng He. Kernelet: High-throughput gpu kernel executions with dynamic slicing and scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 25(6):1522–1532, 2014.

[208] P. Zou, T. Allen, C. H. D. IV, X. Feng, and R. Ge. Clip: Cluster-level intelligent power coordination for power-bounded systems. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 541–551, Sept 2017. doi: 10.1109/CLUSTER.2017.98.