Models and Techniques for Green High-Performance Computing

Vignesh Adhinarayanan

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 and Application

May 7, 2019
Blacksburg, Virginia

Keywords: Green Supercomputing, Power Modeling, Power Management, DVFS, Phase Prediction, Heterogeneity, GPUs, Data Movement, On-chip Interconnects, In-situ Techniques

Models and Techniques for Green High-Performance Computing

ABSTRACT

High-performance computing (HPC) systems have become power limited. For instance, the U.S. Department of Energy set a power envelope of 20 MW in 2008 for the first exascale supercomputer now expected to arrive in 2021–22. Toward this end, we seek to improve the *greenness* of HPC systems by improving their performance per watt at the allocated power budget.

In this dissertation, we develop a series of models and techniques to manage power at micro-, meso-, and macro-levels of the system hierarchy, specifically addressing data movement and heterogeneity. We target the chip interconnect at the micro-level, heterogeneous nodes at the meso-level, and a supercomputing cluster at the macro-level. Overall, our goal is to improve the greenness of HPC systems by intelligently managing power.

The first part of this dissertation focuses on measurement and modeling problems for power. First, we study how to infer chip-interconnect power by observing the system-wide power consumption. Our proposal is to design a novel micro-benchmarking methodology based on data-movement distance by which we can properly isolate the chip interconnect and measure its power. Next, we study how to develop software power meters to monitor a GPU's power consumption at runtime. Our proposal is to adapt performance counter-based models for their use at runtime via a combination of heuristics, statistical techniques, and application-specific knowledge.

In the second part of this dissertation, we focus on managing power. First, we propose to reduce the chip-interconnect power by proactively managing its dynamic voltage and frequency (DVFS) state. Toward this end, we develop a novel phase predictor that uses approximate pattern matching to forecast future requirements and in turn, proactively manage power. Second, we study the problem of applying a power cap to a heterogeneous node. Our proposal proactively manages the GPU power using phase prediction and a DVFS power model but reactively manages the CPU. The resulting hybrid approach can take advantage of the differences in the capabilities of the two devices. Third, we study how in-situ techniques can be applied to improve the greenness of HPC clusters.

Overall, in our dissertation, we demonstrate that it is possible to infer power consumption of real hardware components without directly measuring them, using the chip interconnect and GPU as examples. We also demonstrate that it is possible to build models of sufficient accuracy and apply them for intelligently managing power at many levels of the system hierarchy.

Models and Techniques for Green High-Performance Computing

Vignesh Adhinarayanan

(GENERAL AUDIENCE ABSTRACT)

Past research in green high-performance computing (HPC) mostly focused on managing the power consumed by general-purpose processors, known as central processing units (CPUs) and to a lesser extent, memory. In this dissertation, we study two increasingly important components: interconnects (predominantly focused on those inside a chip, but not limited to them) and graphics processing units (GPUs). Our contributions in this dissertation include a set of innovative measurement techniques to estimate the power consumed by the target components, statistical and analytical approaches to develop power models and their optimizations, and algorithms to manage power statically and at runtime. Experimental results show that it is possible to build models of sufficient accuracy and apply them for intelligently managing power on multiple levels of the system hierarchy: chip interconnect at the micro-level, heterogeneous nodes at the meso-level, and a supercomputing cluster at the macro-level.

To

my parents, Maheswari and Adhinarayanan,

and

my sister, Kavipriya.

Acknowledgments

Many people have contributed to the completion of this dissertation. I would like to express my gratitude to them in this page.

First and foremost, I would like to thank my advisor, Prof. Wu-chun Feng, for his academic and professional guidance, financial support, and patience while I completed this dissertation. Prof. Feng gave me the opportunity to work on many interesting problems and provided the latitude to make mistakes and learn from them. While this hands-off approach aided in my development as an independent researcher, Prof. Feng also provided the gentle guidance necessary to hone my research skills and timely impetus to ensure my forward progress. His constructive criticism during the weekly meetings pushed me to be meticulous with my research and his spot-on feedback on my writing and presentation helped me to communicate my research effectively. I must say, under his supervision, I learned to do research and not merely get a degree.

I would also like to extend my sincere thanks to the other members of my committee — Prof. Ali Butt, Dr. Mark Gardner, Dr. Scott Pakin, and Prof. Eli Tilevich — for their time, feedback, and insights which helped improve the contents of this dissertation.

I was fortunate to have interned at Los Alamos National Laboratory (LANL) and Advanced Micro Devices (AMD) during my graduate studies. My internships at each of these places resulted in productive collaborations and contributed to the topics covered in this dissertation. At LANL, I got an opportunity to work with Dr. James Ahrens and Mr. David Rogers on in-situ frameworks which shaped the last chapter. During my internship there, I also had the pleasure of working with Dr. Scott Pakin who shared his expertise on large-scale power-measurement infrastructure and was very helpful with the measurement studies "behind the fences" even after my internship ended. Many thanks to all three of them. I would also like to thank Dr. Indrani Paul for mentoring me at AMD Research and helping shape the chip-interconnect power-measurement work in this dissertation. I am also indebted to Dr. Joseph Greathouse for sharing his deep technical expertise on low-level details of GPU architecture and microbenchmarking techniques without which I could not have completed the chip-interconnect measurement studies.

Next, I would like thank my labmates for their constructive feedback, brainstorming, camaraderie, and commiseration. At various points, the following people have been helpful in one way or the other: Nabeel, Lokendra, Carlo, Konstantinos, Umar, Anshuman, Sarunya, Ahmed, Tom, Ashwin, and Hao. Special thanks must go out to Balaji for his guidance and feedback during my early years at Synergy Lab. I would also like to acknowledge the members of the Synergy Ops team — especially, Tom, Mark, and Paul — for keeping the lab's infrastructure running smoothly.

Outside of the lab, several people contributed towards my well-being. Of those, I wish to thank Krishnaraj, Nabeel, and Sriram for their friendship while at Virginia Tech, Leonardo Piga for being incredibly helpful during my internship at AMD, and the entire 2015 intern cohort at LANL for their welcoming nature. Thanks also to my friends and family from back home for being my well wishers. My mentors, managers, and colleagues at AMD have been patient while I completed my dissertation. I must also extend my sincere thanks to them.

I would be remiss if I did not acknowledge the role of my undergraduate research advisor, Prof. Venkateswaran (Waran) Nagarajan. He introduced me to research as an undergraduate and played an instrumental role in me getting interested in pursuing a Ph.D. in the first place. I sincerely thank him for the exposure he provided.

Lastly, but most importantly, I acknowledge my family for being a bedrock of support. My mother was available for me any time. She would stay awake in the middle of the night (in India) to send reminders just so that I do not miss an important presentation. My father stayed on top of bureaucratic matters when I got busy with research and made sure that my paperwork was always up to date. My sister, a techie herself, was of great assistance in completing the "last mile" when I got stuck and I must specially thank her for that. Thank you to the three of you! Without you I could not have completed this journey :-)

Funding Acknowledgment My research was funded by National Science Foundation (I/UCRC-IIP0804155 and IIP-1266245), U.S. Department of Energy Office of Advanced Scientific Computing Research (DE-SC0012637), Los Alamos National Laboratory, Advanced Micro Devices, Pratt Fellowship and the Department of Computer Science at Virginia Tech.

Declaration of Collaboration. Apart from my advisor Wu-chun Feng, this dissertation has benefited from several collaborators:

- Indrani Paul (AMD), Joseph L. Greathouse (AMD), Wei N. Huang (AMD), and Ashutosh Pattnaik (Penn State) contributed to the work included in Chapter 3 which resulted in [6]. Specifically, Indrani and Joseph helped refine the ideas presented in this chapter and Joseph helped with a particularly hard implementation detail.

- Balaji Subramaniam (VT) contributed to the work and helped refine the idea presented in Chapter 4 which resulted in [7].

- Joseph L. Greathouse (AMD) and Indrani Paul (AMD) contributed to the work in-

cluded in Chapter 5. Specifically, Joseph provided the traffic traces used in our experiments.

- Scott Pakin (LANL), David Rogers (LANL), and James Ahrens (LANL) contributed to the work included in Chapter 7 which resulted in [5].

Contents

List of Figures

List of Tables

List of Abbreviations and Acronyms

ACF	Auto-Correlation Function
APU	Accelerated Processing Unit
CDF	Cumulative Distribution Function
CUPTI	CUDA Profiling Tool Interface
CU	Compute Unit
DVFS	Dynamic Voltage and Frequency Scaling
EDP	Energy-Delay Product
GCN	Graphics Core Next
GPU	Graphics Processing Unit
HBM	High-Bandwidth Memory
HMC	Hybrid Memory Cube
HPC	High-Performance Computing
HTP	History Table-Based Predictor

ISA	Instruction Set Architecture
LLC	Last-Level Cache
LRU	Least-Recently Used
MAE	Mean Absolute Error
MC	Memory Controller
MDS	Metadata Servers
MLRI	Multiple Linear Regression with Interaction
MLR	Multiple Linear Regression
MPAS	Modeling for Prediction Across Scales
MPC	Model Predictive Control
MSR	Model-Specific Register
NetCDF	Network Common Data Form
NoC	Network on a Chip
NUMA	Non-Uniform Memory Access
NVML	NVIDIA Management Library
OSS	Object Storage Servers
PCU	Package Control Unit
PDU	Power Distribution Unit
PIO	Parallel I/O

PMC	Performance Monitoring Counter
QMLRI	Quadratic Multiple Regression with Interaction
QMLR	Quadratic Multiple Regression
RAPL	Running Average Power Limit
RBF	Radial Basis Function
SE	Shader Engine
SLR	Simple Linear Regression
SMM	Statistical Metric Modeling
SVR	Support Vector Regression
WIO	Wide I/O

Chapter 1

Introduction

In this chapter, we explain the motivation behind our work on *green* high-performance computing (HPC), a subdiscipline of HPC that prioritizes power, energy, and energy efficiency in the design and usage of HPC systems. Then, we present the research problems that we seek to solve and explain the contributions of this dissertation.

1.1 Motivation

The supercomputing community has historically considered performance to be the primary design criterion [68]. Even the notion of efficient supercomputing was viewed as a controversial topic when it was first proposed in 2003 [51]. By 2008, power and energy were identified as the most pervasive challenges on the path towards exascale computing in a report titled *Exascale Computing Study: Technology Challenges in Achieving Exascale Systems* [22]. Subsequently, the U.S. Department of Energy (DOE) targeted a power envelope of 20 MW for the exascale supercomputer now expected to arrive in 2021–22 [88, 36], thereby making power and energy first-order design criteria alongside performance.[1] Achieving the exascale goal under a 20 MW power budget by 2015[2] would have required the fastest supercomputer at that time (i.e., Tianhe-2A) to be 29.5× faster while incurring only a 1.1× power cost. While the situation has improved since then, today's fastest supercomputer (i.e., Summit) still needs to be 7× faster while consuming less than 2× its current power. This means that the energy efficiency of the fastest supercomputer needs to improve by 2500% to meet the exascale target. Therefore, in this dissertation, **we seek to improve the greenness of HPC systems**, where improving greenness can mean reducing power (or energy) consumption, increasing power (or energy) efficiency (i.e., performance per watt or performance per joule), or improving the performance of a power-constrained system (i.e., performance per watt at peak power consumption).

The **overarching goal** of this dissertation is (i) to understand greenness-related issues in modern HPC systems via characterization studies on real hardware and (ii) to improve the greenness of these systems via model-driven management techniques. Specifically, we study measurement, modeling, and management problems for green HPC in the context of the evolving HPC landscape, which is marked by increasing heterogeneity and data movement, targeting the chip interconnect, heterogeneous processor, and cluster representing micro-, meso-, and macro-levels of the system hierarchy.

While our focus is on HPC systems, the techniques introduced in this dissertation should be broadly applicable to almost all areas of modern computing where power is an issue—for instance, mobile computing devices such as phones, tablets, and laptops, traditional desktops, and enterprise servers. In the next sections, we will briefly describe the research problems we study in this dissertation.

[1] Since then the power budget has been informally revised to 30 MW and subsequently 40 MW, but power and energy still remain first-order constraints.

[2] The preliminary target for exascale computing was 2015 as inferred from [22].

1.2 Measurement and Modeling

We first describe the problems related to measuring and modeling power that we study in this dissertation.

1.2.1 Chip-Interconnect Power Measurement and Modeling

On-chip data movement is considered to be a major source of power consumption in modern processors. Properly understanding the power that applications expend in moving data is vital for inventing mitigation strategies. Rather than rely on power estimates from simulators and models, we seek to measure for ourselves the power consumed by the chip interconnect. Developing an approach for measuring chip interconnect power on real hardware would help us study it for the systems we need and along the dimensions we need.

Toward solving this problem, we develop a new microbenchmarking methodology that isolates the chip interconnect better than past work [23, 41, 87, 103, 115, 124, 162]. Using this methodology, we characterize chip interconnect power under different conditions, construct a detailed power model, and validate it against vendor-provided data.

1.2.2 Runtime GPU Power Modeling

Accurate power measurement at runtime is essential for the efficient functioning of a power management system. In this work, we study how to develop an accurate software power meter that can provide power readings at a better granularity than conventional power meters. Similar power meters exist for CPUs, which rely on performance counter-based models. The challenge in this work is to develop software meters of similar accuracy, but for GPUs and without having access to a wider and richer set of performance counters.

Our work explores applying *stepwise regression* and *statistics-driven heuristics* to increase the useful information during model development. We also explore using feedback from temperature sensors and introducing application-specific knowledge during the run time to improve model accuracy.

1.3 Model-Driven Management

In this section, we present three different problems and solutions to improve the greenness of a computing system as explained next.

1.3.1 Reducing Chip-Interconnect Power via Proactive DVFS Management

Our analysis of chip interconnect power consumption reveals that emerging trends such as building larger heterogeneous processors and 3-D stacked high-bandwidth memory will make the interconnect a major consumer of power. In this work, we study how to effectively reduce the power consumption of the interconnect with dynamic voltage and frequency scaling (DVFS). Our proposal is to proactively manage the DVFS state (also known as P-state) of the interconnect. The key challenge to proactively manage the interconnect power comes from the nature of traffic seen on the interconnect. Our research in this area results in a low-overhead, rule-based phase predictor that relies on approximate pattern matching to accurately determine the interconnect's future traffic requirements and set its P-state appropriately.

1.3.2 Power Capping for Heterogeneous Nodes

Building over-provisioned systems is being explored to reach the exascale target under 20 MW, but for them to safely operate, they rely on *power capping*—a power-management mechanism that ensures that a processor operates under its allocated power budget. In this dissertation, we study the problem of capping the power consumption of a heterogeneous node.

Our proposal includes building a hybrid power manager for node-level power management. Due to the differences in the capabilities of the CPU and GPU, such as having different response times before a frequency change is enforced, we propose to use different mechanisms to manage their respective power. In our approach, the GPU's power is proactively managed and the CPU quickly adjusts itself to maintain the node-level power budget.

1.3.3 Reducing Cluster Energy with In-situ Methods

Finally, we explore the role played by off-chip data movement in affecting the greenness of scientific visualization. Our experimental study on a 450-node cluster demonstrated that while traditional post-hoc visualization consumed more energy than in-situ visualization, the off-chip data movement itself is not a significant consumer of energy. However, large off-chip data movement causes the system to idle more often resulting in energy wastage. We also present an analytical model that can be used to tune the scientific visualization parameters to meet a target energy budget.

1.4 Research Contributions

In the context of today's HPC landscape, which is marked by increasing data movement and heterogeneity, we develop approaches for inferring the power consumed by a HPC system at different levels of the system hierarchy, and demonstrate that the power models, prediction techniques, and management techniques introduced in this dissertation can be used to improve the greenness of HPC systems. Specifically, we make the following contributions.

Interconnect Power Measurement. We present a new microbenchmarking approach that targets a processor's interconnect and allows us to study its power consumption in isolation. We demonstrate this idea on real hardware, characterize interconnect power under different conditions, construct a detailed power model, and validate the power model against vendor-provided data. We analyze several applications and find that the interconnect is indeed likely to be a major power consumer in future processors with up to 22% of the dynamic power spent on the interconnect in the near future.

Instantaneous GPU Power Measurement. In this work, we develop an *instantaneous* power model for graphics processors and demonstrate its use as a power proxy that is capable of providing live runtime power estimates on *real* hardware. Our contribution is a series of methods to improve model accuracy and a demonstration of the use of performance counter-based model to estimate power at run time with high accuracy. Specifically, we overcome hardware limitations that reduce the accuracy of performance counter-based models in an online runtime environment by applying *stepwise regression* and *statistics-driven heuristics* to improve model training and use information from temperature sensors and feedback from a low-resolution power meter to help introduce application-specific knowledge during the testing phase of the model. Experimental results show that an error as low as 1% can be obtained from including application-specific information to the power model.

Interconnect Power Management. In this work, we present a model-driven analysis that quantifies the extent to which interconnect power would pose a problem in the future. Based on this analysis, we propose a proactive approach to manage the power of the chip interconnect and quantify its benefits over a reactive approach. We identify the practical challenges in proactively managing the interconnect by studying 37 applications and propose a novel phase predictor based on approximate pattern matching for interconnect power management. We then present experimental results that show how a proactive power management approach that uses our phase predictor reduces interconnect power by 25.5% with little impact on performance.

Power Capping of Heterogeneous Nodes. We propose a new hybrid approach that allows for proactive power-management techniques without being significantly affected by model errors. In this approach, the GPU is first managed proactively with phase prediction and DVFS power models. The CPU responds reactively to the change in GPU power consumption in order to maintain the node-level power budget. In support of this hybrid approach, we also explore building DVFS power models for the GPU. Using data collected from real hardware, we show that the dynamic redistribution of the power budget using our hybrid power-management technique can result in effective power capping compared to a proactive power management approach.

In-situ Methods. We study the role played by off-chip data movement in affecting the greenness of a cluster. We perform characterization experiments and construct analytical models that help provide insights to improve the greenness of scientific visualization. Our experiments show that in-situ methods can improve greenness by reducing off-chip data movement and idle resources.

1.5 Dissertation Organization

The rest of the dissertation is organized as follows. Chapter 2 presents related work. Chapter 3 presents our research on interconnect power measurement, modeling, and analysis. Chapter 4 describes our research on modeling GPU power for runtime power monitoring. Chapter 5 discusses our efforts in reducing interconnect power via proactive power management. Chapter 6 describes our research on improving the performance of a heterogeneous node operating under a power cap. Chapter 7 discusses our investigation into the role played by off-chip data movement in affecting greenness and shows that in-situ methods can help improve greenness by reducing off-chip data movement. We summarize our dissertation in Chapter 8.

Overall, in this dissertation, we study greenness-related problems on measurement, modeling and optimizations under two broad themes—data movement and heterogeneity—targeting the micro-, meso-, and macro-levels of the system hierarchy. A visual mapping of the chapters to the problems, themes, and targeted system components is presented in Fig. 1.1.

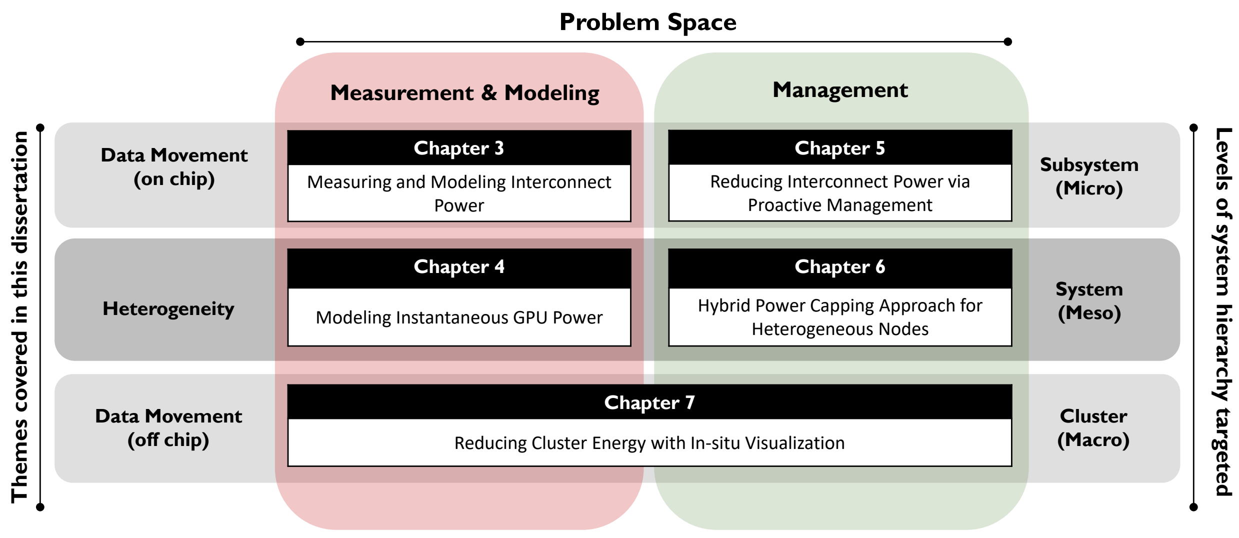

Problem Space
Measurement & Modeling
Management
Themes covered in this dissertation
Data Movement (on chip)
Heterogeneity
Data Movement (off chip)
Chapter 3
Measuring and Modeling Interconnect Power
Chapter 5
Reducing Interconnect Power via Proactive Management
Chapter 4
Modeling Instantaneous GPU Power
Chapter 6
Hybrid Power Capping Approach for Heterogeneous Nodes
Chapter 7
Reducing Cluster Energy with In-situ Visualization
Subsystem (Micro)
System (Meso)
Cluster (Macro)
Levels of system hierarchy targeted

Figure 1.1: Organization of the chapters in this dissertation

Chapter 2

Background and Related Work

This chapters provides background on related work and explains how our contributions differ from these work.

2.1 Techniques for Power Measurement

Understanding where power is being spent in today's high-performance computing systems is essential to improve their efficiency. This section summarizes some popular methods to measure power consumption.

2.1.1 Direct Measurement

The most common way to measure power is to use *external* power meters or *internal* sensors to measure wattage or derive it from related metrics such as current draw and voltage drop. Measurements can be made at different levels of the HPC system. The power consumption of the entire machine can be obtained by measuring at the switchboard level or aggregating the rack-level power measurements as in the case of the Cielo, Roadrunner, and Luna supercomputers [114]. At the rack level, metered power distribution units (PDUs) are becoming common in newer machines. For example, the HP Apollo 8000 System Manager [64] for the Hikari supercomputer at TACC can provide power measurements at one-second granularity. Similarly, it is possible to obtain power measurements at similar granularity at the cage level or the shelf level as in the case of Caddy [15]. In contrast, a Watts Up? PRO can be used to measure the power consumption of a node at a one-second granularity while a Yokogawa WT210 can perform node-level measurements at sub one-second granularity. Several other power monitoring tools are capable of providing a component-wise breakdown of power on a real system. Examples of such tools include Powerpack [54], Powermon [20], and PowerInsight [91]. All these tools operate by directly measuring current and voltage from power rails. However, they cannot measure the power consumption of components on a shared rail (e.g., chip interconnects) in today's processors. An extended discussion of hardware-based measurement approaches can be found in the survey paper by Hsu and Poole [69]. Note that while the component-level measurement devices operate at a higher resolution than node-level or cluster-level devices, such approaches are intrusive in nature. The devices operating at the node or higher levels provide power measurements at a low resolution, which results in fewer opportunities to optimize power [86].

2.1.2 Indirect Measurement

We are primarily interested in measuring chip interconnect power. So we discuss them separately.

Measuring Data-Movement Power

Kestor et al. [87] present a methodology for measuring the energy cost of moving data across the memory hierarchy for scientific workloads. Pandiyan et al. [115] present a similar approach for mobile workloads. They develop microbenchmarks that move data from different levels of the memory hierarchy to the registers. By measuring the difference in energy consumption between these microbenchmarks, they estimate the energy spent towards data movement. Unfortunately, this technique does not separate the energy cost of data movement from that of data access. For example, by subtracting the energy cost of their L1-\$ microbenchmark from the L2-\$ microbenchmark, the resultant energy is not just the cost to move data from L2 to L1, but also includes the energy expended within the L2 cache.

Manousakis et al. [103] also adopt a microbenchmark-based approach where they vary the operational intensity of the microbenchmarks and study power consumption. Their study also measures data accesses rather than data *movement*. The works of Shao et al. [142] and Molka et al. [107] provide estimates for various types of compute and memory access instruction. Of particular interest are the load/store instructions across various levels in the memory hierarchy. However, both studies are limited in their inability to separate data access and data movement power.

Other Indirect Measurements

Techniques that indirectly measure power are largely based on observing some characteristics of the system (e.g., via performance monitoring counters) and using a pretrained model to indirectly infer power. Since the technique largely relies on the power models, we describe such approaches in detail in Section 2.2.1.

2.2 Techniques for Power Modeling

2.2.1 Power Estimation Models

In this section, we describe the related work in power estimation which tells the power consumed by a processor for a certain task.

CPU Power Modeling Bellosa was among the first to show the existence of a relationship between power consumption and performance events [21]. This discovery resulted in several works on instantaneous power prediction [41, 144, 24] and management [76, 72] for CPUs. In addition to adapting these techniques for GPUs appropriately, we also had to overcome limitations in GPU hardware profiling and software tools. Our proposals to address these limitations include application-specific and online modeling. We also systematically study several models similar to the work done by Rivoire et al. [129] and Davis et al. [46] for CPUs.

GPU Power Modeling Ma et al. were among the first to model the power consumption of a graphics processing unit. They use support vector machines to achieve modest accuracy in predicting power [99]. However, this model is no longer applicable to general-purpose GPUs. Nagasaka et al. developed a linear regression-based model to accurately estimate the average power consumed by a GPU kernel on a GeForce 285 GTX GPU [109]. Abe et al. developed an architecture-agnostic power model, but the accuracy was greatly diminished compared to architecture-specific power model [1]. Song et al. present another model using artificial neural networks to estimate average power consumption [145]. Ghosh et al. have explored statistical techniques that encapsulate the non-linear relationship between power and performance events and have reported higher accuracy than the purely linear models [57]. Kasichayanula et al. estimate the power of micro-architectural components on GPU using an empirical activity-based model [84]. However, all these works [57, 84, 109, 145, 1] use several counters requiring multiple application runs and can only predict the power consumption offline.

2.2.2 Component-level Power Models

Regression-based power models constructed using performance counters have the potential to estimate the power consumption of several components within a processor. Past work [124, 23, 41, 86, 162] have provided a breakdown for many components within a processor; however, these power models were only validated for overall power consumption and cannot be relied upon for component-level estimation.

2.2.3 Analytical Models and Simulations

In this section, we describe the related work on analytical models for GPUs and chip interconnects.

GPU Models

Hong and Kim present a detailed analytical model to estimate the power consumed by GPU [65]. Constructing such models requires in-depth knowledge of the low-level microarchitecture, making it difficult to use in practice. GPUWattch [94] and PowerRed [127] are other GPU power models, both of which require extensive architectural knowledge and low-level simulation. While these models are detailed and provide high accuracy, they cannot be directly ported to real hardware for instantaneous power measurement due to the limitations of hardware counters.

Models for Chip Interconnects

At the lowest level, it is possible to model chip interconnect power with circuit simulators such as SPICE. These tools provide excellent low-level details but require a great deal of design information and are extremely slow. It is unlikely that hardware designers would release SPICE-level models of large microprocessors. GPUWattch [94] and McPAT [95] model various components including chip interconnects, but are constructed using some reference numbers from the industry. However, compared to SPICE models, they can have higher errors and some times the exact details of a target architecture may not be available as well.

In this dissertation, we focus on the chip interconnects for component-level investigation. So we discuss models and tools available for this particular component specifically. High-level tools such as Orion [81] provide reasonably detailed models for the various chip-interconnect components. Orion relies on data released by the industry to validate and fine-tune its model. With the limited information that is available in the public domain, researchers improved the accuracy of earlier versions of Orion [157, 80], but the model needs constant revision as chip-interconnect technology advancements are released [154]. The sparsity of publicly available data on power breakdown for modern processors makes this revision and validation difficult. Our work, presented in Chapter 3, makes it possible to independently obtain this reference data. In addition, our methodology makes it possible to run real applications on hardware and obtain the data-movement power for an entire application run rather than rely on worst-case estimates from low-level tools. Similar limitations exist in other analytical models for chip interconnect power such as DSENT [152]. Our work enables rigorous validation of such models by making it possible to independently obtain real-world chip interconnect power measurements on much larger designs and applications.

2.2.4 DVFS Power Models

Su et al. [150] present a DVFS power model for CPUs. In their paper, the power consumed is modeled as a function of frequency of the CPU and the performance counter values at that frequency. A hardware event predictor takes the performance counter values at a baseline frequency as input and estimates the corresponding values at other frequencies. Using the hardware event predictor and the DVFS model, one could predict the power consumed by the application at any frequency. We adopt a similar approach in our dissertation to support dynamic power management. Wu et al. [161] use machine learning approach to predict the GPU power consumed by an application at any frequency, again using a few performance counter values as input. Dutta et al. [49] and Guerreiro et al. [59] also present DVFS model for NVIDIA GPUs. Compared to these works, our model needs to predict the power at different states at runtime. This is a harder problem to solve as the number of performance counters that can be monitored at runtime is fewer than what can be monitored statically. Also certain types of models such as SMO regression proposed by Dutta et al. [49] would incur a significant overhead at runtime.

2.2.5 Phase Prediction Models

In order to proactively manage a processor's power consumption, we rely on phase predictors to estimate the future requirements of the processor. The ideas explored in this dissertation include Markov predictors and history table-based methods. Sherwood et al. [143] first explored Markov models for phase prediction. While the approach used in this dissertation is similar, we apply Markov models at kernel boundaries rather than at some discrete time intervals which makes it a harder problem to achieve high prediction accuracy. Our work also covers a broader type of application for evaluation and examines previously unexplored parameters (i.e., number of states that the predictor needs to choose from). Isci et al. [73, 77, 75, 71] introduced the history table-based approach for phase prediction. Duesterwald et al. [48] also proposed online table-based approach for phase prediction. In our work, we introduce the notion of approximate pattern matching which helped improve the accuracy of the predictor considerably. Furthermore, we study parameters not previously explored for phase prediction (i.e., rule length, size of the table, and number of DVFS states). In contrast, Sarikaya et al. [134] explored Markov model-based approaches to predict workload behavior. We compare the effectiveness of our approach against Markov models and quantify the accuracy improvements.

2.3 Techniques for Power Management

2.3.1 Chip-Interconnect Power Management

Several works have explored DVFS-based techniques for the power management of a processor's chip interconnect. Shang et al. [141] were among the earliest to identify chip interconnect power to be a serious problem in modern processors and proposed DVFS for chip interconnect links. They proposed a history-based approach to set the DVFS state for chip interconnect links based on past utilization, but it only targeted long-term changes in chip-interconnect utilization and ignored short-term fluctuations in traffic. Unlike their approach, we provide sophisticated mechanisms to address short-term fluctuations. Since we have all the links on a single DVFS island and due to the emerging trends highlighted in Section 5.2.2, the potential opportunities to save power on the chip interconnect is larger than in the past, which makes our proposed sophisticated schemes possible. Other early works have targeted on/off links where power management is restricted to choosing between two states [146], control-theoretic approaches for power management of chip interconnects [113, 29] and the entire uncore[1] [38, 39, 40]. Similar to our work, their entire network on a chip (NoC) is on a single DVFS island; however, unlike our work, their power-management techniques are reactive in nature, whereas ours is proactive. Some more recent work has explored proactive management of chip interconnects, but they target coherence traffic [159, 63]. The NoC studied in this dissertation carries all traffic except coherence traffic, thus our work complement theirs [159, 63].

In the literature, researchers have also proposed a thread voting approach [163]. Like several earlier works [141, 146], this work selects DVFS states for each chip interconnect link individually, and our comparatively heavy-weight approach may not be appropriate when the potential power-saving opportunities are lower.

2.3.2 Proactive Power Management

While this dissertation proposes proactive power management of the chip interconnect for the first time, this class of power management has been previously explored for other components of a computing system. These works are acknowledged here and differences highlighted, where appropriate.

Hsu et al. [67] propose an automatic power management algorithm that detects memory boundedness to proactively managers power consumption. Lo et al. [96] use latency statistics collected in the past to set DVFS states of cores to meet some latency requirements. Isci et al. [72] demonstrate proactive power management with a history table-based approach. Chen et al. [37] predict the characteristics of an upcoming phase and adapt DVFS states dynamically. Majumdar et al. [101] explore proactive power management on GPUs. Their

[1]The uncore includes the last-level cache (LLC) and the network on a chip (NoC)

work uses model predictive control (MPC) techniques to predict the behavior of the next n kernels to plan the energy profile for the future. We require to predict only the next *one* state and consequently explore lighter-weight techniques.

2.3.3 Power Capping

Isci et al. [70] were among the earliest to explore the concept of maximizing the performance of a power-constrained processor. In their work, they adjust the per-core DVFS states in accordance with workload characteristics and demonstrate that is possible to ensure safe operation under a power budget without significant performance degradation. Lefurgy et al. [93] introduced the concept of power shifting (also known as power sloshing), which explicitly looks at the solution to the problem of maximizing the performance of a power-constrained processor by stealing power from components that do not need them to components that need them. David et al. [45] present Intel's power-capping product that ensures that a component (such as core or memory) operates under a power budget within a given time window. This enabled many cluster-level power management schemes [130, 117, 17, 104, 18, 118, 138, 55, 164, 132, 56].

On heterogeneous devices, researchers have looked at related research problems. Paul et al. [120, 121] explored the problem of balancing the power consumption between the CPU and GPU components of an accelerated processing unit (APU) and between compute units and memory of a GPU [119]. The end goal in the above work is to minimize the energy-delay product (EDP) whereas our work focuses on maximizing the performance under a power budget. The underlying solutions in these dissertations [120, 121, 119] rely on a reactive approach whereas our solution, in part, relies on proactive power management. Majumdar et al. [102] also propose proactive power management for heterogeneous nodes for the related problem of minimizing EDP, but their solution uses model-predictive control (MPC) which is a NP-hard solution and may not be optimal for real-time power management considering the overheads involved.

Chapter 3

Measuring and Modeling Chip-Interconnect Power

This chapter presents our work on measuring power consumed by the interconnect on real hardware, characterizing interconnect power under different conditions, and modeling interconnect power using the data collected.

3.1 Introduction

Two major challenges associated with new silicon technology nodes have exacerbated power issues:

1. Dennard scaling has failed, meaning that as transistor density continues to increase, the power used by each transistor no longer decreases at the same rate.
2. The power density of wires is increasing even faster than that of transistors due to poor wire-size scaling [27]. The cost of communication is thus a large and growing concern.

This chapter deals with the latter issue. While the power associated with data movement has been recognized as a problem that needs to be addressed, the extent of the problem is not yet clearly understood [35, 98]. No previous study has accurately measured the data-movement power in real, modern processors. Some of the difficulties in doing so are highlighted in the work of Leng et al. [94], which states, "It is almost impossible to isolate L2 cache power from network-on-a-chip (NOC) power because each L2 cache access involves an NOC access."

An implication of the above statement is that it is difficult to separate the cost of data-access from the cost of data-movement with conventional measurement approaches. This limitation is also observed in the work of Kestor et al. [87], who were among the first to attempt to measure the energy cost of data movement on *real* hardware. Thus, despite the perceived importance of data-movement power, no previous study has accurately measured it separately from data-access power.

In this chapter, we devise a set of novel techniques that overcome these limitations and separate the power of data movement from that of data access. Specifically, we design microbenchmarks that use distance-based metrics, instead of traditional data-volume metrics, to study the chip interconnects. Our microbenchmarks each have the same data-access rates and perform the same operations, but they differ in the physical distance that the data must travel within the chip interconnect. This allows us to separate the chip interconnect's power from data-access power.

Our microbenchmarks allow us to characterize the chip-interconnect power used by an AMD GPU built in 28 nm technology. We observe that the chip interconnect's power increases linearly with the distance of data movement, the wire toggle rate, and the bandwidth of data movement. Nonetheless, applications with the same toggle rate can consume different power based on the values sent along the wires due to the effect of crosstalk. We then use this data to develop *architecture-specific* empirical models and to study the chip-interconnect power of 22 real applications running on a GPU. To highlight the utility of our model, we analyze a power-reduction technique, namely optimizing the chip layout for lower chip-interconnect energy (rather than other metrics such as signal delay).

The contributions of this chapter can be summarized as follows:

- **We describe a novel methodology to measure the chip-interconnect power in *real* processors**. We design a series of microbenchmarks that use the same operations to access on-chip memories in different locations at the same rate. We demonstrate this on a modern AMD GPU, though our methodology is applicable to any architecture.
- **We characterize the chip interconnect power of 22 applications both in 28 nm technology and in a hypothetical 7 nm node**. We show that up to 14% of the dynamic power in these applications comes from the chip interconnect and that this may increase to 22% in a 7 nm node.
- **We demonstrate our model's utility by exploring a power-reduction technique**. Specifically, we study *layout-based optimization*, e.g., the impact of the placement of L2 and memory controllers within the chip.

The rest of the chapter is organized as follows. We present some background material in Section 3.2. Section 3.3 details our interconnect power-measurement methodology. We present the results of our characterization studies in Section 3.4 and our models in Section 3.5. We use our models to study real applications in Section 3.6 and evaluate interconnect power-mitigation techniques in Section 3.7. We present our conclusions in Section 3.8.

3.2 Background

This section details the hardware we use in our studies and describes some pertinent microarchitectural details of AMD Graphics Core Next (GCN) architecture.

3.2.1 AMD GCN Architecture

For our tests, we used an AMD FirePro™ W9100 GPU, a workstation-class discrete GPU that uses the Graphics Core Next (GCN) 1.1 instruction set architecture (ISA) [11]. A simplified block diagram of this GPU, which nonetheless roughly represents the location of many important structures, is shown in Figure 3.1. This GPU consists of four shader engines (SEs), each containing a number of compute units (CUs) that are similar to 64-wide vector processors. The AMD FirePro W9100 has 11 CUs per SE, yielding a total of 44 CUs.

Each CU has its own dedicated L1 data cache that is connected to the CU by short wires (not shown in the figure). The L2 cache is divided into several partitions (16 on our GPU), but every CU can communicate with every L2 partition via a crossbar. Each L2 partition is directly connected to an on-chip DRAM controller. As we discuss later, these controllers (and thus also the L2 cache) are address sliced, such that each controller accesses (and each L2 partition caches) a disjoint subset of the memory space.

These L2 cache partitions are located in different parts of the chip, meaning that the physical distance between any pair of CU/L1 and L2 partition can vary measurably. Each quadrant of the L2 cache has an SE "local" to it (i.e., the physical distance separating them is smaller compared to the distance between that SE and another L2 cache quadrant). For example, the CUs in SE-I in Figure 3.1 are closer to the L2 partitions at the topleft of the design than they are to L2 partitions at the bottomright. We will exploit this observation to characterize

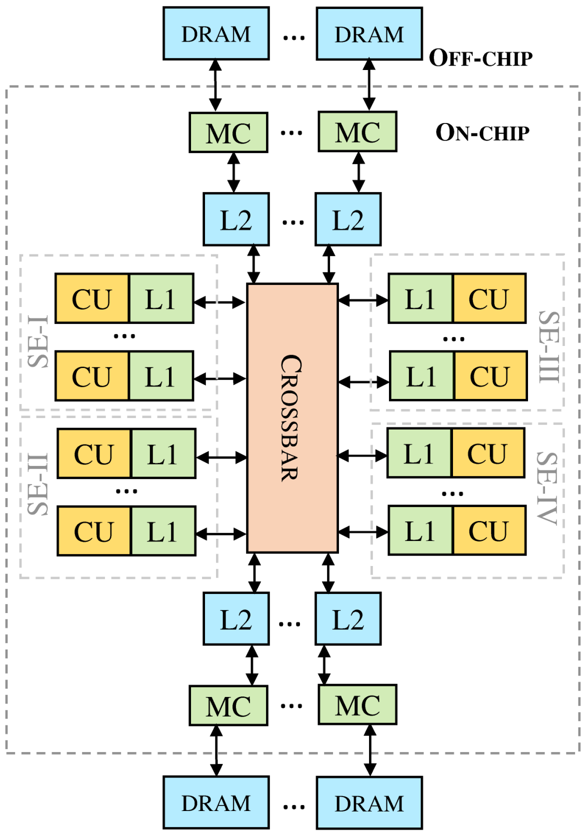


Figure 3.1: Representative block diagram of the GPU, showing 8 out of 44 compute units (CUs)

the chip interconnect's power later in this chapter.

Given the goal of this study is to estimate the power consumption of the on-chip interconnects and assess where data-movement power is spent, we focus on three major aspects of chip interconnects: (i) the wires between the CUs and L1, (ii) the crossbar connecting L1 and L2, and (iii) the wires between the L2 partitions and memory controllers.

3.2.2 Experimental Setup

As previously mentioned, we performed our experiments on an AMD FirePro™ W9100 discrete GPU. Table 3.1 lists the key parameters of this GPU.

Parameter	Value
Total Compute Units (CUs)	44
CUs per Shader Engine (SE)	11
Total SEs	4
Core Frequency	930 MHz
L1 Cache Size per CU	16 kB
Total L2 Cache Size	1024 kB
Number of L2 Partitions	16
Total DRAM Size	16 GB
Number of DRAM Channels	16
Memory Frequency	1250 MHz

Table 3.1: Description of the AMD FirePro™ W9100 GPU

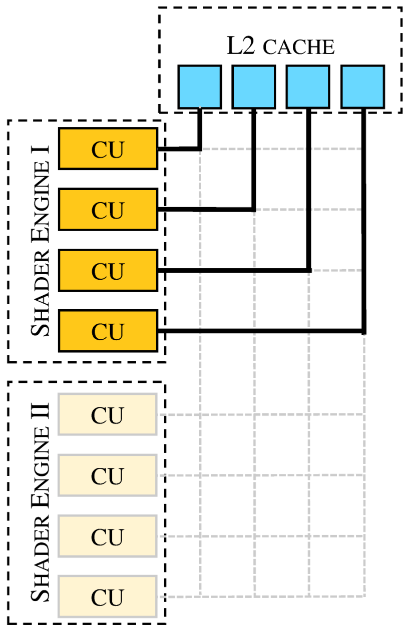
Software Setup: We ran our experiments on a host with Ubuntu 14.04, AMD FirePro driver v15.20.7, and the AMD APP SDK v2.9.1. Our microbenchmarks use OpenCL™ 1.2.

Power Monitoring: To monitor the power consumption of our GPU, we use a high-precision power meter that measures current and voltage from the voltage regulators going into the chip. This instrument can provide power measurements at 1 kHz. The instrumentation setup is capable of measuring the power consumption of only the chip as a whole, and hence the study is limited to focusing just the on-chip data movement and not the off-chip movement (e.g., to main memory).

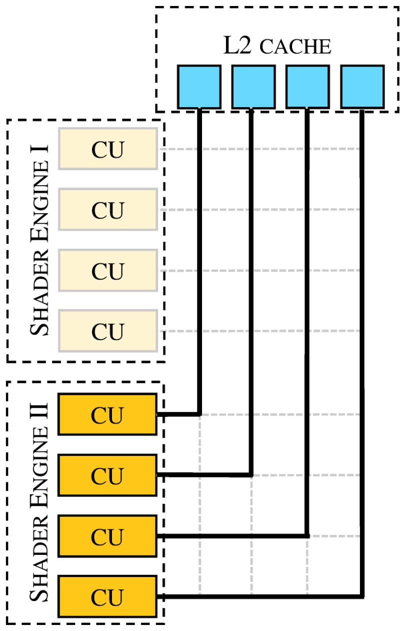
Performance Counters: To guide the design of the microbenchmarks and to validate them, we use AMD CodeXL v1.6. We later describe how AMD CodeXL performance counters can be used to estimate chip-interconnect power in larger applications in Sections 3.5 and 3.6.

3.3 Methodology

This section describes our microbenchmarking strategy for measuring the interconnect power of the processor described in Section 3.2.2. While the details are specific to the AMD GPU, the methodology itself is generalizable to other architectures.



(a) Short Path



(b) Long Path

Figure 3.2: Design of our chip interconnect microbenchmarks

3.3.1 Overview

Our microbenchmarking methodology is based on the observation that longer wires consume more energy than shorter wires while carrying the same current. Therefore, data that travels a longer physical distance within the chip consumes more energy than the same amount of data moving a shorter distance.

Our conjecture based on the above observation is that when we continuously move data from a partition of the L2 cache to the various L1 caches that are located in the different parts of the chip, we should observe a difference in power consumption. To test this conjecture, we design two microbenchmarks, illustrated in Figure 3.2. The first (referred to as **short path**) continuously moves data between all the compute units (CUs) in shader engine I and the

L2 quadrant closest to it. The second (referred to as **long path**) moves the data between shader engine II and the same L2 quadrant, thereby moving the data across a longer physical distance. While this idea is demonstrated on real hardware with a crossbar interconnect as an example, our approach can be easily extended to any interconnect topology. For instance, if we had a mesh topology, we could design a pair of microbenchmarks where one microbenchmark would move data between neighboring nodes and the other moving data from the topleft compute unit to bottomright compute unit.

Next, we explain the implementation challenges in realizing our design on real hardware and accurately measuring the power difference, as this is a non-trivial task. Then, we present our solutions to these challenges in the subsequent sections.

3.3.2 Detailed Approach

Realizing the basic idea presented in Section 3.3.1 on real hardware poses several challenges that must be mitigated:

1. We use OpenCL™ to implement our microbenchmarks, but it lacks native support to pin threads to programmer-specified locations on the chip.
2. Designing a microbenchmark where all of the data is fetched from one quarter of the L2 cache is challenging, since each L2 quadrant contains only 256 kB, whereas the total size of an SE's L1 caches is 176 kB.
3. The microbenchmarks must use as much bandwidth as possible to reliably observe and measure the chip-wide power difference between the two microbenchmarks.
4. Latency effects must be hidden from the long-path microbenchmark. Because the second shader engine is located on a physically different part of the chip from the first

SE, there is an increase in latency when it accesses the top-left L2 quadrant. Sufficient L2 requests must be generated so that the long-path microbenchmark sees the same bandwidth as the short-path microbenchmark.

5. Temperature has a major impact on the power consumption of a processor. The effects of temperature on the two microbenchmarks should be properly isolated so that only the effect of data-movement distance is measured.

Locking OpenCL™ Kernel to Specific SEs

While OpenCL™ does not directly offer support for running threads on only one shader engine (SE), it is possible to achieve the effect by editing the binary that is generated by the OpenCL runtime. An example is shown in Figure 3.3, where work is performed only on CUs 0 through 10 (i.e., SE-I). In this approach, we write an initial OpenCL snippet, shown in Figure 3.3a, in which useful work is performed only if the wavefront ID is between 0 and 10. The wavefront ID is a placeholder that we will modify to hold the value of CU IDs.

Figure 3.3b shows a portion of the equivalent GCN assembly code for this snippet. The instruction that writes the value of the wavefront ID to the variable used in the *if* conditional is boldfaced and highlighted in red. The scalar register corresponding to this variable is `s0` and the hex of the instruction that writes its value is `81000210`. Figure 3.3c shows the hex value of the instruction in little-endian format in the binary file, which can be obtained using `clGetProgramInfo()`.

We then manually replaced this instruction with the `S_GETREG_B32` GCN instruction, which loads the CU and SE IDs out of the `HWID` register and puts them into `s0` (`B9003204`, as shown in Figure 3.3d). This derivation is based on the information provided in AMD ISA manuals [10, 11]. We further verified that this process resulted in the desired effect by

```
__kernel void l2_read( __global float *data,
                              __global float *output) {
            int gid = get_global_id(0);
            int wid = get_group_id(0);
            if (wid >= 0 && wid <= 10) {
                        // Read data from L2
            }
}
```

(a) Initial OpenCL code snippet

```
s_min_u32     s0, s0, 0x0000ffff  // 000000000014: 8380FF00 0000FFFF
s_mul_i32     s0, s16, s0         // 00000000001C: 93000010
s_add_i32     s0, s0, s1          // 000000000020: 81000100
v_add_i32     v0, vcc, s0, v0     // 000000000024: 4A000000
s_add_i32     s0, s16, s2         // 000000000028: 81000210
s_cmp_gt_i32  s0, -1              // 00000000002C: BF02C100
s_cbranch_scc0  label_0011        // 000000000030: BF840004
```

(b) Equivalent assembly code

```
00 FF 80 83 FF FF 00 00
10 00 00 93 00 01 00 81
00 00 00 4A 10 02 00 81
00 C1 02 BF 04 00 84 BF
00 FF 04 BF 81 00 00 00
C1 80 01 85 01 00 82 BF
```

(c) Equivalent binary

```
00 FF 80 83 FF FF 00 00
10 00 00 93 00 01 00 81
00 00 00 4A 04 32 00 B9
00 C1 02 BF 04 00 84 BF
00 FF 04 BF 81 00 00 00
C1 80 01 85 01 00 82 BF
```

(d) Modified binary

```
__kernel void l2_read( __global float *data,
                              __global float *output) {
            int gid = get_global_id(0);
            int cu_id = get_cu_id(0);
            if (cu_id >= 0 && cu_id <= 10) {
                        // Read data from L2
            }
}
```

(e) Equivalent OpenCL code

Figure 3.3: Steps to launch wavefronts on only one shader array

analyzing the performance counters from AMD CodeXL.

After making these modifications, we use `clCreateProgramWithBinary()` to load our custom binary into the application. This achieves the same effect as writing the hypothetical OpenCL code shown in Figure 3.3e.

Accessing Data Only from L2

Our microbenchmarks seek to access data from one quadrant (four out of the 16 partitions) of the L2 cache that is located closest to SE-I. In our target architecture, there is a one-to-one mapping between the memory channels and the L2 partitions. That is, the data that resides in one memory channel can be cached in only one L2 partition. The address interleaving for the memory channels is specified in AMD's instruction-set architecture (ISA) manuals [11].

Each channel holds a contiguous 256 bytes of memory (equivalent of 64 floats) and, given an address we can identify the channel number from bits 8-11. Using the above information we obtain the L2 cache partition given an array index for any data type and thus write a microbenchmark that only targets a particular partition.

Saturating the L1-L2 Chip Interconnect Bandwidth

To obtain the best results from our microbenchmarks, we must use as much L1-L2 bandwidth as possible. Higher bandwidth means a greater difference in the sum total of the data moved for each benchmark on a per-second basis, which should translate to a greater difference in power consumption between the two microbenchmarks. This difference will help minimize error from other uncontrollable sources, such as measurement noise. In addition, saturating the chip interconnect also helps to keep the CU pipelines busy, helping to prevent the long path from stalling more often than the short path due to any difference in L2 access latency.

Unfortunately, launching a small number of wavefronts to a small number of CUs cannot saturate the L1-L2 interconnect if they only touch one cache line before stalling. We could design our microbenchmarks such that each thread accesses several cache lines. This would require extra address calculations and could potentially increase the global working set size, however, resulting in register pressure and unwanted main-memory accesses. Alternately, we could increase the number of wavefronts kept in flight. This could inadvertently increase the L1 hit rate by scheduling threads in a way that keeps all of the data accessed by one wavefront in the L1 cache. Based on the above options, we chose the latter option but with modification in order to maintain the L1 hit rate.

To prevent an increase in the L1 cache hit rate, we modified the firmware of our GPU to artificially shrink the size of L1 cache to 4 KB per CU. This allowed us to increase the number of wavefronts in flight (thereby increasing the chip interconnect bandwidth and hiding the L2 access latency for long-path), avoid cache hits in L1, and keep accesses to the main memory to an absolute minimum and focus on compulsory misses only.

Isolating Temperature Effects

Modern silicon technology consumes significant static power which is, in turn, affected by the operating temperature. However, our power measurements come from the off-chip voltage regulators. These regulators must supply all power to the chip, both static and dynamic. This means that our measurements cannot directly differentiate between the two. To this end, we developed a small set of tests to help us isolate the dynamic power in the chip interconnect from the static and other non-interconnect power.

We build a power model for idle power to capture the effect of temperature on power. We gathered the data required to build this model by fixing the frequency and voltage of the GPU and heating the chip with a computationally intensive application (e.g., the FurMark benchmark). After the GPU reaches our target temperature, we stop the benchmark and allow the chip to cool while still maintaining the frequency and voltage. As the chip cools, we continually measure the chip's temperature using the on-chip thermal sensors and the chip's power using our power monitor. Figure 3.4 shows the idle power of our target device

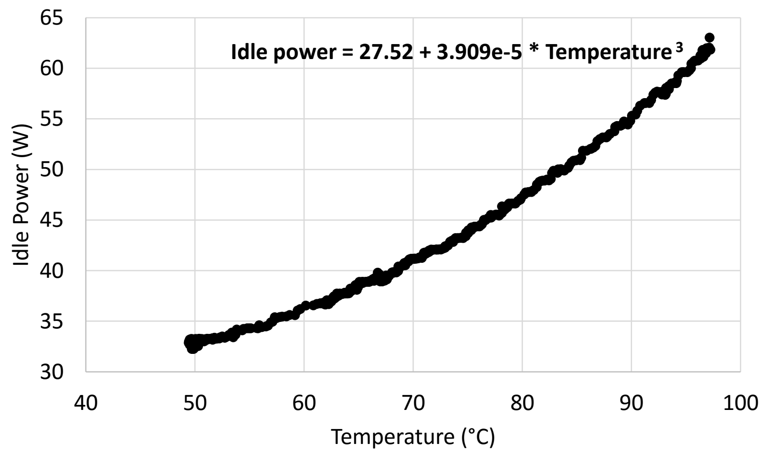


Figure 3.4: Relationship between idle power and temperature

across the range of temperatures that we studied. We can observe from this data that there is a non-linear relationship between idle power and temperature. This effect of temperature should thus be separated from our voltage regulator measurements to accurately measure the power consumption of the chip interconnect.

To achieve this separation during our microbenchmarks, we run the GPU's fan at high speed to constrain the device temperature. We then construct an idle power model for the device using a regression of the data we present in Figure 3.4, which models idle power as a cubic function of the device temperature. The model is optimized for the typical operating temperature range for our microbenchmarks in order to increase its accuracy. Using this model, we subtract out the idle power for the microbenchmark tests from our voltage regulator measurements. This allows us to separate out the effects of temperature from our tests and focus on chip-interconnect power caused by communication.

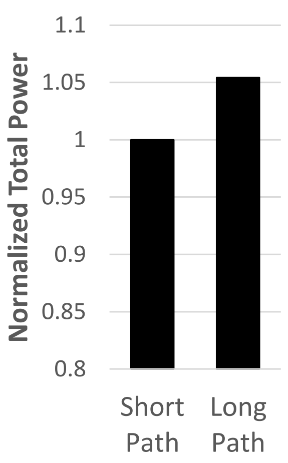
3.4 Characterization Results

In this section, we present the results of our microbenchmark studies that show the impact of the following parameters on chip-interconnect power: (i) data-movement distance, (ii) toggle rate, (iii) voltage and frequency, and (iv) chip-interconnect bandwidth.

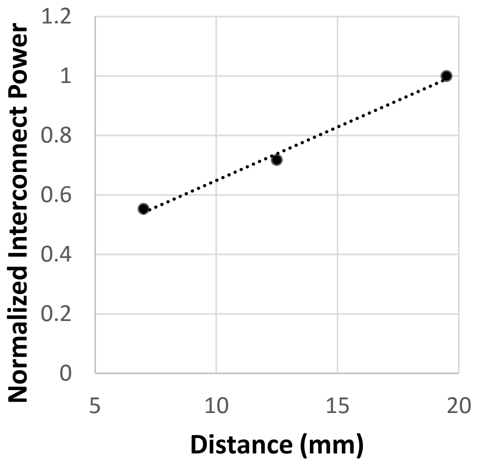
Impact of Data-Movement Distance

Figure 3.5a shows the average dynamic-power consumption for the short-path and long-path microbenchmarks. The values presented on the y-axis are normalized against the short-path microbenchmark. This figure shows that long path consumes 5% more chip-wide dynamic power than the short path. These two microbenchmarks have identical computational and data access rates as verified from hardware performance counters. Therefore, the additional power can only be attributed to the higher data movement distance for the long-path microbenchmark. This additional distance is estimated to be 10.5 mm from an analysis of a die photo of the GPU [158]. Specifically, we calculated the *average* distance between each CU in SE-I and the top-left L2 quadrant and subtracted it from the average distance between CUs in SE-II and the same L2 quadrant from the die photo. Alternatively, when the chip layout is available, the distance can be calculated from the layout diagram.

Validation efforts. We converted the observed difference in power for a distance of 10.5 mm to a metric known as *energy/bit/mm*, which is the energy cost to move one bit of data through a physical distance of 1 mm. This value (110 fJ/bit/mm) was compared against industrial estimates available for 40 nm [85] and 32 nm [27] technology nodes using appropriate scaling factors from [27]. We found that our estimate for energy/bit/mm was within 10% and 15% of these two industrial estimates. Dally [42] quotes 1 nJ of energy consumption for a 256-bit bus when traversing the length and breadth of a 20x20 mm^2 chip (i.e., when traversing 40 mm)



(a) Benchmark power



(b) Distance vs. chip interconnect power

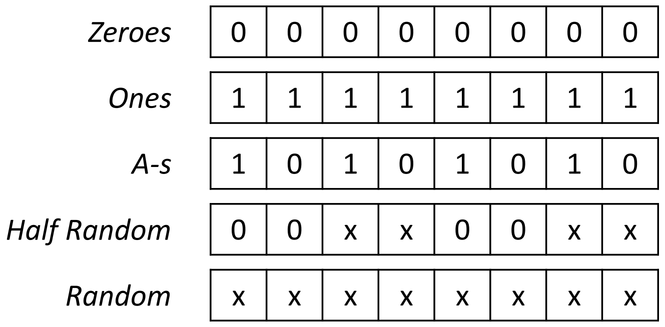
Figure 3.5: Impact of data movement distance on chip interconnect power.

which translates to an energy cost of 98 fJ/bit/mm which is within 11% of our measured value.

Next, we study the relationship between the data-movement distance and the chip-interconnect power. For this study, we developed microbenchmarks that are variants of the short-path and long-path microbenchmarks. The basic idea behind the microbenchmarks remains the same, but instead of running OpenCL™ threads on 11 CUs (i.e., an entire shader engine), we run them only on 4 CUs. This allows us to obtain the difference in power consumption for different distances. The values obtained for the chip-interconnect power from *four* such microbenchmarks are presented in Figure 3.5b. In this figure, the x-axis represents data-movement distance and the y-axis represents chip-interconnect power normalized against the highest value observed in this set of experiments. One of the four microbenchmarks is used to obtain reference power based on which the other three microbenchmarks are studied. Therefore, we have three data points in the graph. Our characterization result shows that the chip interconnect's power increases linearly with data-movement distance.

Impact of Toggle Rate

Next, we studied the impact of toggle rate on chip-interconnect power. For this study, we moved different data patterns across the chip interconnect and observed the power difference for the short path and the long path. Figure 3.6 shows the patterns studied. Of these, *zeros*, *ones*, and *A*-s show no toggling. *Zeros* and *ones* are self-explanatory; for *A*-s, we send a pattern of alternate *1*s and *0*s, which when represented in hexadecimal looks like a string of *A*-s. For the random data, each bit can take any value and the probability of bit toggling (i.e., a transition from *1 to 0* or *0 to 1*) is 0.5. For the half-random dataset, a few bits of random data and a few bits of zeros alternate. The overall toggle rate for this dataset is 0.25.

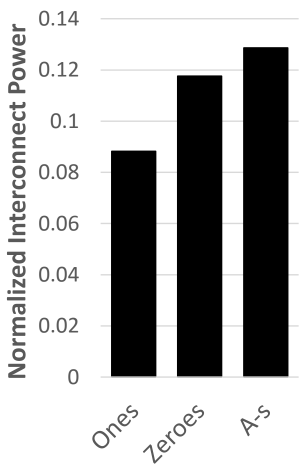


Zeroes	0	0	0	0	0	0	0	0
Ones	1	1	1	1	1	1	1	1
A-s	1	0	1	0	1	0	1	0
Half Random	0	0	x	x	0	0	x	x
Random	x	x	x	x	x	x	x	x

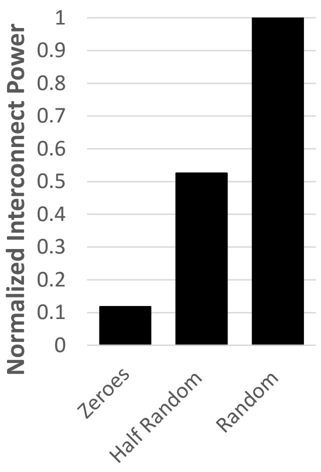
Figure 3.6: Data patterns explored in this study

Figure 3.7a shows the normalized chip-interconnect power for data patterns showing 0% toggle rate. Figure 3.7b shows the same for data patterns exhibiting toggle rates from 0% to 50%. The normalization is performed against the random dataset. Note that the figures are drawn to different scales. We make the following observations from this study:

1. Sending only zeros or ones consumes a small amount of power in the chip interconnect (about 10% of the power seen for random data). This power can be attributed to the arbiters present within the chip interconnect which is independent of toggle rate.



(a) 0% toggle rate



(b) 0-50% toggle rate

Figure 3.7: Toggle rate and data pattern impact on chip-interconnect power

2. Transmitting zeros consumes more power than ones.

3. Interference from neighboring bit lines has a small, but noticeable impact on the chip-interconnect power. This can be seen from the fact that A-'s consume more power than zeros despite showing 0% toggle and transmitting fewer power-hungry 0 bits.

4. Toggle rate has a significant impact on the chip-interconnect power as seen from zeros (0% toggle), half-random (25%), and random data (50%). The relationship between toggle rate and chip-interconnect power is linear.

Impact of Voltage and Frequency

We repeat our experiments while setting the GPU to different dynamic voltage and frequency scaling (DVFS) states in order to study the impact of voltage and frequency on the chip-interconnect power. Figure 3.8 shows the normalized chip interconnect power for these DVFS states. In this figure, the chip interconnect power is plotted against V^2f which is the expected relationship between voltage, frequency, and power. As expected, the relationship between them is linear. Note that the chip-interconnect bandwidth (or the amount of data) differs at the various points in the graph as the frequency changes.

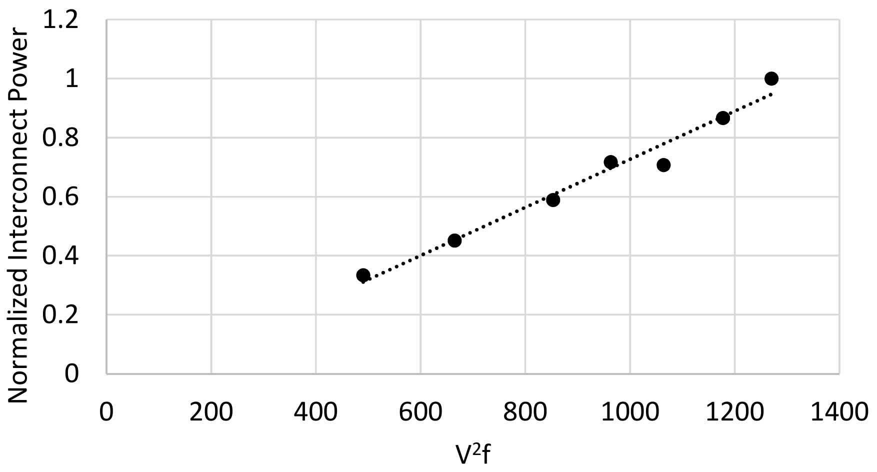


Figure 3.8: Normalized chip interconnect power for moving data at different DVFS states

Impact of Chip-Interconnect Bandwidth

Next, we study changes in chip-interconnect power when the amount of data that moves through it changes. To perform this experiment, we inserted NOPs in our code to reduce the frequency of data access from the L2 cache, which also reduces the chip-interconnect bandwidth. A lower bandwidth means fewer bit transitions per second and consequently lower power. Figure 3.9 shows this for two different bandwidths, where we observe that the chip-interconnect power is roughly half when the chip-interconnect bandwidth is reduced to half its original value.

3.5 Modeling Chip-Interconnect Power

The characterization results presented in Section 3.4 can be combined into a parameterized equation, which naturally lends itself to model interconnect power of larger applications, dif-

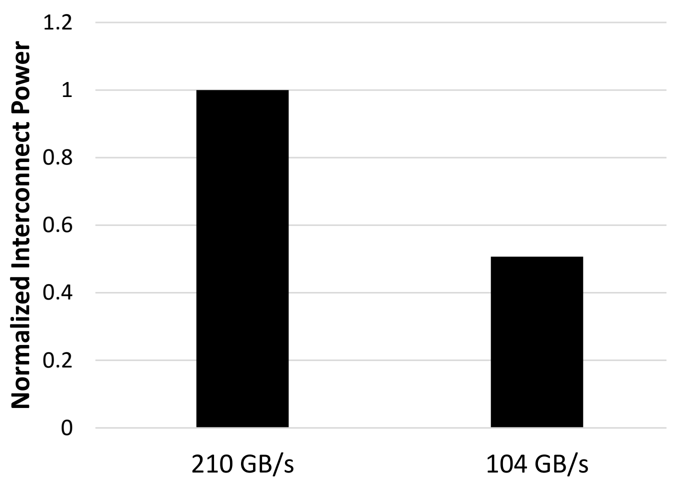


Figure 3.9: Impact of bandwidth on chip interconnect power.

ferent chips, and different technology nodes. The general form of the parameterized equation can be expressed as follows:

Chip Interconnect Power = Constant × % Peak Bandwidth × Toggle Rate × Distance × Scaled Frequency × Scaled Voltage^2

Constant refers to the maximum power consumed by the chip interconnect for a given chip and a reference DVFS state. This value is calculated from the difference in power consumption between the short path and long path (shown in Figure 3.5a), which is then scaled for peak bandwidth, 100% toggle rate, and unit wire distance. The *constant* value is architecture-specific and can be derived for existing GPUs using the microbenchmarks described in Section 3.2.2 and extrapolated to future technology nodes using process scaling information [27].

Next, we describe how to estimate chip-interconnect power for real applications at different interconnect segments of the memory hierarchy, as shown in Figure 3.1, using hardware performance counters (PCs). First, the obtained bandwidth (BW) is calculated for each interconnect segment from the PCs for *L1 accesses*, *L2 hits*, and *L2 misses*. This gives a measure of the actual data volume for an application at different segments.

$$\begin{aligned}
Register\ to\ L1\ BW &= \frac{L1\ accesses}{Time} \times L1\ width \\
L1\ to\ L2\ BW &= \frac{L2\ hits + L2\ misses}{Time} \times L2\ width \\
L2\ to\ MC\ BW &= \frac{L2\ misses}{Time} \times MC\ width
\end{aligned} \tag{3.1}$$

The bus width of L1 cache and L2 cache is 64 bytes, and the width of the memory controller (MC) is 32 bytes for our target architecture. The obtained BW is then expressed as a percentage of the peak interconnect BW. The calculation for the peak L1-L2 BW is shown as an example below:

$$Peak\ L1\ to\ L2\ BW = \#\ L2\ banks \times 64\ bytes\ per\ bank \times clock\ rate \tag{3.2}$$

Next, *toggle rate* is the probability of bit toggling for a given program. For completely random data, the expected probability of toggling is 0.5. The typical average toggle rate observed for the chip interconnects is 0.34 [14].

Distance is an estimate of the average distance the data has to move through the chip interconnect. For existing GPUs where application threads are not pinned to any particular CU and accesses are evenly distributed across all L2 slices, using average distance for calculations is a reasonable assumption. For the AMD FirePro™ W9100 GPU, we calculated the average distance for each part of the chip interconnect by using layout information from the design, though public die photos could also be used [158]. Table 3.2 presents the distances we measured.

The chip-interconnect power for any voltage and frequency pair can be calculated by scaling

Chip Interconnect	**Estimated distance**
Register to L1	3.5 mm
L1 to L2	10.5 mm
L2 to memory controller	11.5 mm

Table 3.2: Average distance estimates for the different parts of the chip interconnect

these parameters with respect to the reference voltage and frequency pair. Alternatively, the constant factor may be recalculated from the microbenchmarks for the required voltage and frequency.

3.6 Estimation of Chip-Interconnect Power for Real Applications

In this section, we estimate the interconnect power of 22 OpenCL™ applications obtained from various sources shown in Table 3.3. We chose these applications considering that the maximum frequency for our power meter is 1 kHz and the chosen applications all have OpenCL kernels that run long enough (over 2 ms) to get meaningful power measurements. The total GPU power for each application at runtime is measured using the power meters that measure voltage and current from the voltage regulators. We also measure the average temperature of the GPU chip across all its thermal sensors while running the applications. To extract dynamic power from these measurements, the idle power is subtracted using the temperature-idle power relationship described in Section 3.3.2.

In our evaluation, using our performance counter-driven model, we estimate the interconnect power spent by the application at the various parts of the interconnects: (i) register to L1, (ii) L1 to L2, and (iii) L2 to memory controller (MC). The results are presented for the 28 nm AMD FirePro™ W9100 GPU architecture and a hypothetical 7 nm shrink of the same

Source	**Applications**
AMD APP SDK	eigen, fwt, histo, montecarlo, nbody, scan
DOE proxy apps	CoMD and CoMD-LJ [106], XSBench [153], LULESH [82], and miniFE [61]
Graph500 [108]	graph500
OpenDwarfs [52]	crc, gemnoui, swat
Pannotia [31]	color
Phoronix [90]	mandelbulb, smallpt
Rodinia [32]	kmeans, streamcluster, srad
SHOC [44]	stencil, spmv

Table 3.3: Applications used for evaluation

die. For the hypothetical chip, we use Borkar's scaling factors for wires and transistors [27] to scale the total dynamic power and interconnect power from 28nm to 7nm.

Figure 3.10 shows the power spent on the different parts of the interconnect, expressed as a percentage of overall dynamic power, for the various applications for the 28 nm and the hypothetical 7 nm GPUs, respectively. Due to the lack of toggle-rate monitors in hardware for these results, we assume an average toggle rate of 0.34 for all applications which is based on past studies [14]. Across applications, the interconnect consumes 5.6% of the total dynamic power on our GPU on an average. Within the interconnect, *register to L1* consumes the most power, using over 45% of the total interconnect power. The crossbar consumes 30% of the total interconnect power, and the rest is consumed by *MC to L2*.

Among all applications, *color* shows the highest percentage of 14.3% for interconnect power. This is due to the fact that *color* is an irregular application with many branch and memory divergences, causing a large amount of data accesses at different levels of the memory hierarchy. *Comd-lj*, *kmeans*, *lulesh*, and *scan* also consume significant amounts of interconnect power with over 10% of the overall dynamic power going towards the interconnect. Of these, *kmeans*, *lulesh*, and *scan* are memory-bound and understandably consume a greater amount of interconnect power as data has to be frequently fetched from the distant memory.

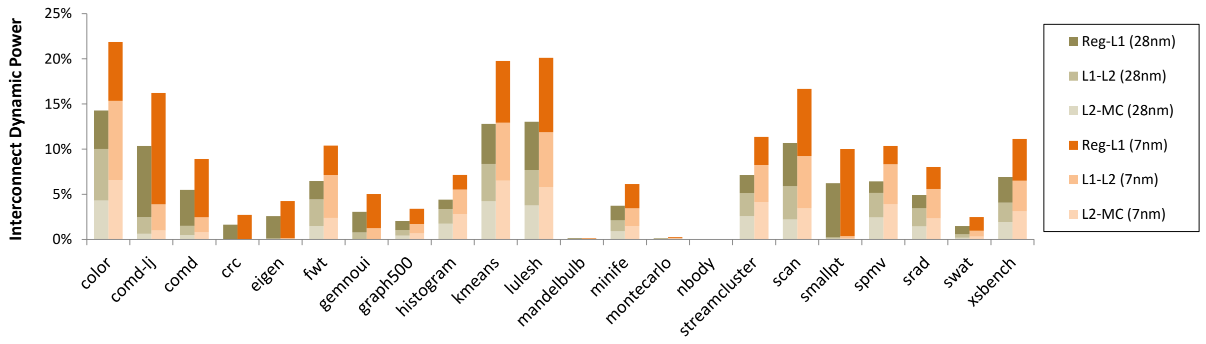


Figure 3.10: Percentage of the total dynamic power spent by the interconnect on the 28 nm FirePro™ W9100 GPU and a hypothetical 7 nm die shrink. The assumed toggle rate is 0.34 for all the applications.

Comd-lj is largely compute-bound with most data accesses either going to the register file or L1. Although the distance between the SIMD units and L1 is relatively small, it still has a significant amount of power spent in data movement because of the high data-access counts to L1.

At the other extreme, applications such as *mandelbulb*, *montecarlo*, and *nbody* all consume nearly *zero* interconnect power. These are all compute-bound, but unlike *comd*, the working set for these applications fits within the register files and therefore does not access L1 much. Therefore, they avoid short-distance accesses as well and see lower data-movement power.

On the 7 nm architecture, the trends remain the same, but, the interconnect consumes 8.9% of the total dynamic power across applications. Individually, we see up to 21.9% for interconnect power as in the case of *color*. These values correspond to nearly a 59% increase in the interconnect power for real applications. This highlights that data movement is going to be an even more significant problem in future GPUs.

3.7 Interconnect Power Optimization

The interconnect power model presented in this chapter can be used to evaluate and guide several optimization techniques in a variety of scenarios ranging from design-time optimization to runtime management of interconnect power. In this section, we present one such example—layout-based optimization.

Layout-based Optimization

Here we use our model to quickly evaluate different layouts in order to find the one that minimizes data-movement power. Intuitively, by reducing the physical distance for the part of the interconnect that is being used the most, one can save data-movement power. In this section, we quantify the savings possible using two sample layouts that optimize different parts of the interconnect.

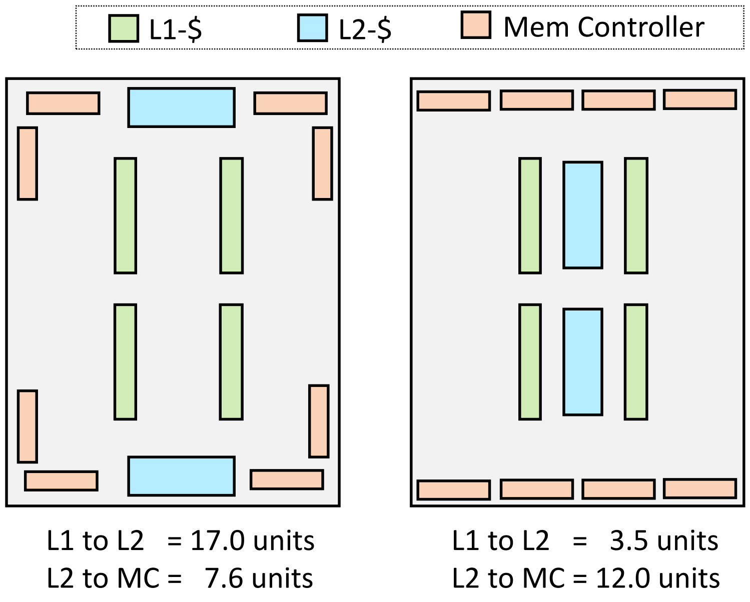


Figure 3.11: Two sample layouts that are designed to reduce the distance between L2 cache and memory controller (left) and the distance between L1 cache and L2 cache (right).

Figure 3.11 shows the two sample layouts. In the baseline case, the average Manhattan distance between L1 and L2 is 17.0 units, and the average distance between L2 and MC is 7.6 units. The layout on the right tries to reduce the L1-to-L2 distance at the cost of a significant increase in L2-to-MC distance. The distances for this layout are 3.5 units for the L1-L2 interconnect and 12.0 units for the L2-MC interconnect.

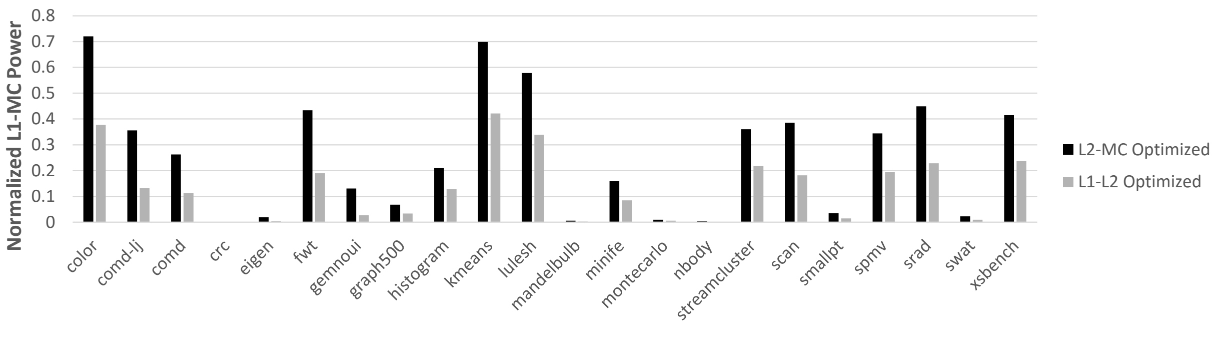


Figure 3.12: Normalized interconnect power for L2-MC optimized layout and L1-L2 optimized layout

We use our model to calculate the power consumed by these interconnects for the layouts presented in Figure 3.12. We assume that the conditions are similar to our experimental platform: (i) 28 nm technology node, (ii) 1.1687 V, (iii) 930 MHz, and (iv) the same constant factor in our equation, owing to equivalent wire capacitance. The normalized power for the interconnects between L1 and MC is presented in Figure 3.12 for our testing applications.

We observe that the L1-L2 optimized layout consistently consumed less power for all the applications. On an average, the L1-L2 optimized layout consumed 48% lower power for the interconnects between L1 and MC. A maximum of 79% reduction in power was observed for *eigen* as there are far fewer references to memory than to L2 for this application. Our results thus show the importance of prioritizing L1-L2 interconnects over L2-MC interconnects.

3.8 Summary

In this chapter, we devised a novel methodology to measure interconnect power using carefully developed distance-based microbenchmarks. We then developed an empirical model using hardware performance counters to obtain the interconnect power for any large application. We evaluated 22 applications and showed that up to 22% of the dynamic power of a GPU can be consumed by the interconnect in the 7 nm node. Finally, we explored layout-based optimization to reduce interconnect power.

Chapter 4

Modeling Instantaneous GPU Power

4.1 Introduction

Achieving the exascale goal under a 20 MW power budget requires both software and hardware innovation. Traditionally, on the software side, a runtime system manages the system power consumption [67, 131]. Power models play a vital role in the efficient functioning of power management by estimating the instantaneous power consumption of the system at runtime. While external power *meters* enable accurate power measurement at high resolution, power *models* or software meters offer other advantages such as enabling power measurements at even higher resolution and in a cost- and infrastructure-effective manner with a minor loss in accuracy. While emerging internal meters offer similar advantages in theory, internally they measure current from voltage regulators and the measurements tend to be noisy and need to be smoothed out over a time window. Further, power models offer another advantage over software meters — power models can be naturally extended to model the impact of dynamic voltage and frequency scaling (DVFS) for effective power management, a problem investigated later as a part of Chapter 6.

In this chapter, we focus on modeling the power consumption of a processor for the purpose of online power measurement. Since we observe a proliferation of graphics processing units (GPUs) in the Green500 list [58] as shown in Figure 4.1, our efforts are targeted at GPUs.

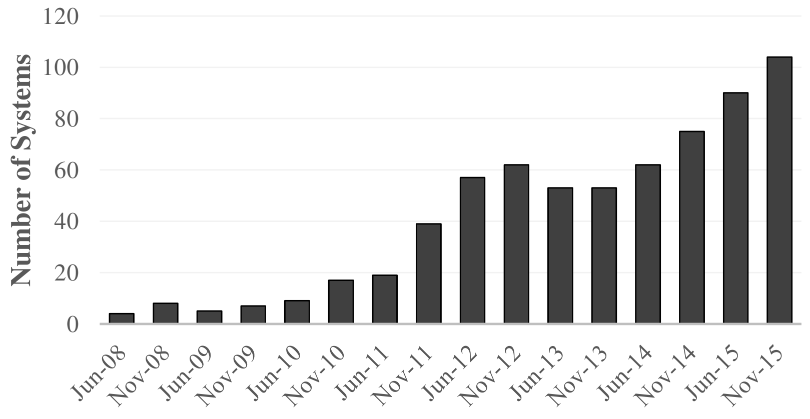


Figure 4.1: Systems using accelerators in the Green500 lists

While power models for online power measurement have been extensively studied in the past, those studies targeted CPUs. Here, we study power modeling for online measurement in the context of GPUs. This topic merits further investigation as extensive CPU studies have led to some convergence on the power models used for CPU systems, but not for GPUs. For example, linear models are widely accepted to be suitable for estimating CPU power consumption. For GPUs, some researchers have shown that linear regression-based techniques are suitable [109] while others argue that such models may be insufficient to capture the complexities of a modern GPU architecture [145]. Moreover, the conclusions drawn from these studies are based on experiences gained in estimating the *average power* consumption. However, for these models to be useful in a runtime system, they should predict *instantaneous power* and should possess the following properties:

- **Accuracy:** The models have to estimate instantaneous power consumption accurately (less than 5% error rate) and track power-phase changes so that a runtime system can make correct decisions for power management.

- **Overhead:** Given the premium on performance in an HPC system, estimating the power consumption should not adversely affect the performance of the system under consideration. Therefore, monitoring system activity and predicting power should incur minimal overhead (e.g., not more than 1%).

Limitations of existing GPU power models: While a number of GPU power models have been explored recently, they all suffer from one of the following limitations.

1. They require the applications to be run multiple times to collect all the necessary input to the model [57, 84, 109, 145, 1]. While the model may be constructed offline by running the application multiple times, the model cannot be used at runtime because all the input values need to be measured simultaneously which is not possible at runtime.
2. They work only in a simulator as the parameters used in these models cannot be measured on a real system [94, 127].

Our work overcomes the above limitations. The contributions made in this chapter can be summarized as follows:

- We present the *first* realization of an *instantaneous* power model for GPUs that is capable of providing live, runtime power estimates on real hardware. Towards achieving the above, we perform a rigorous comparison of five types of statistical models.
- To improve the model's accuracy, we introduce temperature-awareness to the model. While common in low-level models, modeling the temperature effects is generally lacking even in the well-studied higher-level CPU power models that are based on performance counters.

- To improve the accuracy of the instantaneous GPU power model, we introduce the notion of application-dependent models.

- To make this practical, we propose to construct these power models *online* at runtime using the GPU's built-in power sensors for feedback. Our evaluation shows promising results for this approach with a mean absolute error rate of 1% and negligible overhead.

- Finally, we build and evaluate architecture-independent, portable models for estimating power across platforms.

We highlight some of the observations from our experimental results which can help with making the right modeling decisions for other GPU architectures.

- In the case of application-independent models, multiple linear regression produces the highest accuracy with a mean absolute error rate less than 6% for both microarchitecture generations of NVIDIA GPUs under consideration.

- Temperature plays a significant role in determining the power consumed by the GPU. For all models evaluated, introducing temperature awareness improved the prediction accuracy.

- Application-dependent models deliver significantly higher accuracy with a mean absolute error rate of nearly 1% for both GPUs using quadratic models. Our results also reveal that the penalty of using a quadratic model is higher when sufficient information is not available.

- Only a minimal number of samples (100, as determined by our experiments) is required to construct an accurate application-dependent model at runtime. This indicates that the model can be constructed at runtime with a low overhead and high accuracy.

The rest of the chapter is organized as follows. Section 4.2 discuses background related to the statistical techniques used in this work. Section 4.3 describes the hardware platform and application used for GPU power modeling. Section 4.4 describes our methodology. We present our evaluation in Section 4.5, and Section 4.6 concludes this chapter.

4.2 Background

In this section, we describe the statistical techniques used to model GPU power.

Statistical Methods

Regression techniques are normally used to establish the relationship between two variables, a dependent variable y (also known as the *response*) and an independent variable x (also known as the *predictor*), through an unknown parameter β. In our experiments, the response variable is the power consumed and the predictors are the performance counters. Regression modeling involves finding the best value for β, given a modeling function f. Traditionally, the method of least squares is used to find the value of β.

Linear Models: In statistics, if the modeling function f is linear in β, then the model is considered linear. Thus, even if the relationship between the dependent variable y and the independent variable x is non-linear, a model falls under the category of linear models as long as β is linear. The linear models explored in this chapter are described below.

Simple Linear Model (SLR): In this model, the response variable y, depends on a single predictor x. The basis function can be written as $y_i = \beta_0 + \beta_1 x_i + \epsilon$, where ϵ is the error term that cannot be modeled empirically.

Multiple Linear Model (MLR): In a linear model, if the dependent variable is related to more than one independent variable, then it is called an MLR model.

Interaction Effects: Interaction terms must be included in a model if two independent variables play a combined role on the dependent variable and their effects cannot be separated. Interaction effects are expressed through a third variable which is the product of the two independent variables. In this chapter, we consider only second-order interaction effects (i.e., interaction between two variables only).

Quadratic Relationships: The MLR model may also include higher-order variables which indicates a non-linear relationship between the predictors and the response. In this chapter, we evaluate quadratic models in which the response depends on the square of the predictors.

The mathematical basis functions of the MLR models evaluated in this chapter are presented below:

Basic MLR model without interaction (MLR)

$y_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \epsilon$

Basic MLR model with interaction (MLR+I)

$y_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \beta_{12} x_{1i} x_{2i} + \epsilon$

Quadratic model without interaction (QMLR)

$y_i = \beta_0 + \beta_1 x_{1i} + \beta_{11} x_{1i}^2 + \beta_2 x_{2i} + \beta_{22} x_{2i}^2 + \epsilon$

Quadratic model with interaction (QMLR+I)

$y_i = \beta_0 + \beta_1 x_{1i} + \beta_{11} x_{1i}^2 + \beta_2 x_{2i} + \beta_{22} x_{2i}^2 + \beta_{12} x_{1i} x_{2i} + \epsilon$

Stepwise Regression: Sometimes, too many independent variables are considered for modeling. To eliminate the unnecessary ones, we use a statistical technique known as *stepwise regression*. This technique alternates between two steps: (i) *forward selection* and (ii) *back-*

ward elimination. During the forward selection step, the variable with the smallest *p-value*[1] is added to the model if this value is below a certain threshold. Intuitively, this variable explains the most variation in the modeling data. In the backward elimination step, among all variables added to the model, the one with the largest *p-value* is dropped if the value is above a certain threshold. This indicates that the variable dropped does not significantly affect the accuracy of the model. The algorithm terminates when there are no more variables to be added or dropped.

4.3 Experimental Setup

In this section, we described the hardware platform and the applications used in this study.

Hardware Platforms

Our hardware platforms include two high-end NVIDIA GPUs from different generations: (i) Fermi C2075 and (ii) Kepler K20c. Table 4.1 presents the relevant details of these platforms. Both these platforms are equipped with built-in sensors to measure power and temperature and have hardware counters to profile performance events (i.e., system activity).

Measuring Power and Temperature: Power and temperature values reported by the built-in sensors can be accessed through the NVIDIA management library (NVML) interface [112]. This interface provides a C-based thread-safe API to monitor and manage the GPU. The corresponding methods to measure instantaneous power consumption and temperature are `nvmlDeviceGetPowerUsage` and `nvmlDeviceGetTemperature`, respectively.

Profiling Performance Events: CUDA profiling tools interface (CUPTI) is used to profile

[1] *p-value* is a statistical metric that indicates whether a variable truly has an effect on the response.

Table 4.1: Hardware details

Parameters	**Fermi C2075**	**Kepler K20c**
# CUDA cores	448	2496
# SMs	14	13
Core frequency	1150 MHz	706 MHz
Memory size	6GB	5GB
Memory type	GDDR5	GDDR5
Memory frequency	1.5 GHz	2.6 GHz
Memory bandwidth	144 GB/s	208 GB/s
Peak DP performance	515 GFlops	1170 GFlops
Total board power	215W	225W

performance events by configuring and querying the hardware performance counters available in the NVIDIA GPUs [111]. There are 74 and 140 native performance events in C2075 and K20c, respectively. However, only a small fraction of these events (between one and eight, depending on the chosen events) can be profiled simultaneously.

Applications

Statistical modeling involves a *training phase* in which the data is collected and the model is constructed and a *testing phase* in which the prediction accuracy of the model is evaluated. For our training phase, we chose workloads that exhibit a variety of computational and communication patterns representing a spectrum of application behavior. The *Level 1* applications (basic computational primitives) from the SHOC benchmark suite [44] were appropriate for this task. For the testing phase, we use applications from various benchmark suites including *Level 2* applications (full-fledged applications) from SHOC, selected CUDA SDK samples, LLNL ASC proxy apps [83], and CUDA-equivalent SPEC ACCEL benchmarks [147] from Rodinia [32] and Parboil [148]. Tables 4.2 and 4.3 present a brief description of the training and the testing applications, respectively.

Table 4.2: Training applications

Benchmark	**Description**	**Problem Size**
Stencil2D	Standard two-dimensional, nine-point stencil	4096x4096; 1000 *iter*; 100 *pass*
SpMV	Sparse matrix-vector multiplication	12288x12288; 250 *pass*
FFT	Multiple two-dimensional fast Fourier transform	512-pt; 256MB; 1000 *pass*
GEMM	Single- and double-precision matrix multiplication	16KB sq. matrix; 50 *pass*
MD	Pairwise calculation of Lennard-Jones potential	36864 atoms; 20000 *pass*
Reduction	Sum-reduction operation	64MB vector; 1000 *pass*
Triad	Streaming vector dot product computation	16MB data; 1500 *pass*
Scan	Parallel prefix-sum algorithm	64MB data; 350 *pass*
Sort	Radix sort algorithm	96MB data; 20000 *pass*
BFS	Breadth-first search on an undirected graph	1000000 vertices; 1000 *pass*

4.4 Methodology

In this section, we describe our data collection methodology, event selection technique, and model construction.

4.4.1 Data Collection

We modified all the applications to include a *profiling CPU thread* that periodically measures instantaneous power and board temperature via NVML and system activities via CUPTI. This thread periodically looks up dedicated GPU hardware registers for measurement and does not interfere with normal GPU execution. The profiling interval was set to 20 ms for the C2075 GPU and 1000 ms for the K20c GPU considering the capabilities of the power

Table 4.3: Testing applications

Benchmark	**Description**	**Problem Size**
LULESH	Unstructured explicit shock hydrodynamics problem using Lagrangian methods	90 edges
S3D	Chemical reaction computations across regular 3D grid	48/40 edges; 2500*pass*
QTC	Quality threshold clustering algorithm	26x1024
FWT	Product of a square data set and matrix of basis vectors	128 (Data), 32M (Kernel)
Eigen	Eigen-value calculation for a given matrix	32768x32768
NW	Needleman-Wunsch algorithm for DNA-sequence alignment	16384 *seq*; 20 *pass*
Hotspot	Processor-temperature estimation based on architectural floorplan and power measurements	2048x2048, 4(ht) ; 60000 *iter*
Histo	Histogram calculations for number of occurrences of each value	256Wx8192H
MRI-Q	MRI image reconstruction from sampled radio responses	128x128x128
TPACF	Two-point angular correlation function to measure distribution of massive bodies in space	48589 pts; 100 *pass*

measurement infrastructure of these systems.[2] The problem sizes for the different applications were chosen to ensure that we collect enough data points for model construction and testing. The relevant details for each application is shown in Tables 4.2 and 4.3.

4.4.2 Event Selection

Selecting the right predictors in a regression model plays an important role in determining the accuracy of the model. While all relevant events could be included to maximize the accuracy, current GPUs have only a limited number of hardware counters to profile events. Furthermore, several pairs of events cannot be simultaneously profiled. This severely limits

[2]We chose 20ms for C2075 versus 1000ms for K20c as the latter has issues with fine-grained power measurements as reported in [28].

the number of events that can be included in the model.

From several tens of available events, we select a concise set to model power consumption. The selection is done in two phases. In the first phase, we consider the events in isolation to identify those exhibiting high correlation with power. In the second phase, we consider the events in concert, identifying non-redundant events that can be simultaneously profiled to address device limitations.

The steps involved in the first phase are listed below:

1. We collect performance counters and power drawn for each {application, event} pair by running the applications multiple times.

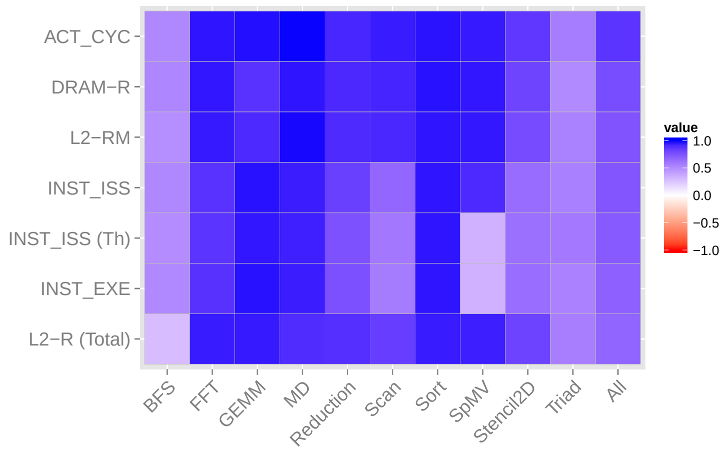
2. We compute the Pearson's correlation coefficient between performance counters and power for each application individually, and for all the applications collectively.

3. We eliminate the events showing a low overall correlation less than δ. The value for δ was set as 0.65 for C2075 and 0.55 for K20c by manually performing sensitivity analysis to maximize accuracy.

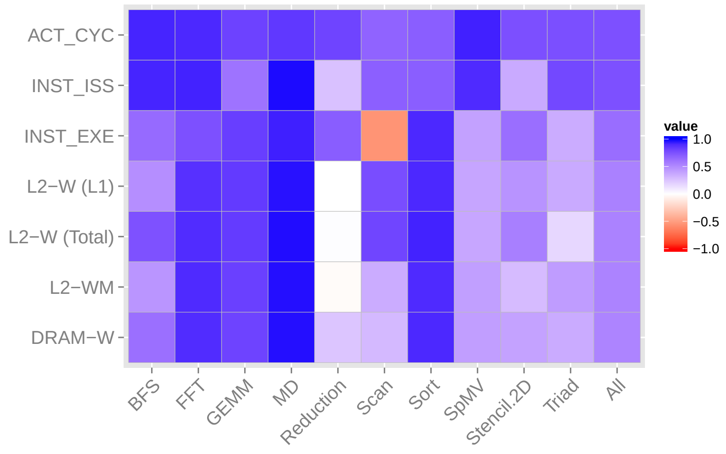
4. From the remaining set of highly correlating events, we eliminate those events that do not consistently show high correlation across applications.

We show the events identified at the end of this phase in the form of a correlation heatmap for the two GPUs under consideration in Figure 4.2a and Figure 4.2b.

Figure 4.3 shows the algorithm used in phase two to determine events that are ultimately included in the power model. In this algorithm, the events are considered in decreasing order of overall correlation, with the highest correlating event selected first. For subsequent events, we check if they can be profiled simultaneously with the events already selected. If so, such



(a) Correlation heatmap for C2075



(b) Correlation heatmap for K20c

Figure 4.2: Correlation heatmap. Performance counters showing the highest overall correlation with power (in decreasing order). Higher correlations are represented by darker shades and lower correlations are represented by lighter shades.

events are included in the model only if they do not correlate with any of the events selected prior. As a result, we eliminate events that do not provide any new information.

The performance counters identified at end of this phase for modeling power are described below:

- **ACT_CYC:** Number of cycles in which the GPU has at least one active warp.

```
Input: E (Set of events showing high correlation)
Output: S (Set of events to be included in the model)
Algorithm
S ← Ø
for each event $E_i$ (in decreasing order of correlation) in set E
    if $E_i$ can be simultaneously profiled with events in Set S, then
        Calculate Pearson's correlation coefficient $\rho_{ij}$ between
        $E_i$ and all events $S_j$ in Set S
        if $\rho_{ij} < \rho_{min}$ for all j, then
            $S \leftarrow S \cup E_i$
        end if
    end if
end for
```

Figure 4.3: Algorithm for event selection

- **DRAM-R:** Number of read requests sent to DRAM.
- **INST_ISS:** Number of instructions issued.
- **INST_EXE:** Number of instructions executed.
- **L2-R:** Number of read requests sent to L2 cache.
- **L2-W(L1):** Number of write requests sent to L2 cache from L1 cache.

Among these, only `ACT_CYC`, `INST_ISS` and `INST_EXE` were selected for both the GPUs. The difference between the two GPUs is that reads (`DRAM-R`, `L2-R`) correlated with power on the C2075, whereas writes (`L2-W(L1)`) correlated with power on the K20. Table 4.4 shows the predictors used for the two GPUs. We also consider portable models constructed using only those counters that show high correlation for both the GPUs under consideration. We consider these models to evaluate the scope of architecture-independent models. That is, we study whether it is possible to construct a single power model that works for all GPUs.

Table 4.4: Predictors used in the models

Counter	C2075	K20c	Portable
ACT_CYC	+	+	+
DRAM-R	+		
INST_ISS	+	+	+
INST_EXE	+	+	+
L2-R	+		
L2-W (L1)		+	

4.4.3 Modeling

We construct five different models: simple linear regression (SLR), basic multiple linear regression (MLR), basic multiple linear regression with interaction (MLR+I), quadratic multiple linear regression (QMLR), and quadratic multiple linear regression with interaction (QMLR+I). We also explore the following approaches to construct the above five types of models.

Application-Independent Models: Two distinct sets of workloads are used for the training and testing phases as described in Section 4.3. From each workload used in the training phase, we collect 150 power and performance counter values to construct the model. This ensures adequate representation of each application in the model construction and avoids biasing the model towards longer-running applications. We use the `lm` function available in the statistical software `R` to construct the model. The intercept values are fixed at 83 W and 42 W for the C2075 and K20c GPUs respectively. These values are the power consumed by the GPUs in their *active idle state*. The predictors of these models are the events identified in the *event selection* step. For the MLR models, we further refine the model by using stepwise regression to eliminate predictors that are not useful. To achieve this, we use the `stepAIC` function in `R`.

Application-Dependent Models: To increase the accuracy of the models, we explore application-dependent modeling. However, constructing such a model involves a one-time cost for each application. To reduce the cost, these models can be constructed using small-sized problems that run only for a few iterations.

Online Models: Our proposed technique for improving the accuracy while addressing the drawbacks of application-dependent modeling involves the construction of a power model using the GPU's power sensors during the first few seconds of an application's execution. The power consumed by the rest of the application is predicted from the model thus constructed. The idea is to use a low-resolution power meter to construct a high-resolution power model for fine-grained power management. To test the feasibility of the approach, we study the sensitivity (i.e., accuracy) of the model with respect to the number of samples (or data points), thereby determining the minimum number of samples required to construct these application-dependent models at runtime. A low number of samples indicates that the model can be constructed quickly within the first few seconds of an application's execution.

Temperature-Aware Models: An initial analysis of the predicted values on the workloads used in the training phase revealed that the accuracy steadily decreased with an application's execution time. This is because the long-running applications increased the GPU's temperature which in turn affected the leakage power. To study the relationship between temperature and power, we operated the GPU under different temperatures. We performed this study by measuring the *active idle* power *after* the execution of a stress workload that increased the GPU's temperature. As also seen previously in Figure 3.4, we find that idle power is affected by the temperature of the operating device. Therefore, in order to accommodate the effect of the GPU temperature on power consumption, we add temperature as a predictor to the models under consideration.

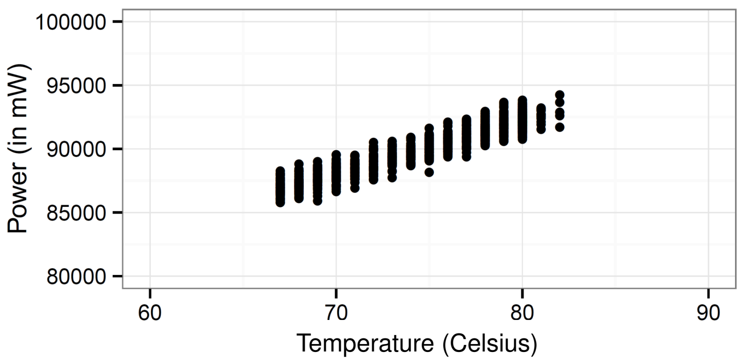


Figure 4.4: Effect of temperature on idle power

4.5 Results

In this section, we present the accuracy of the various models in terms of mean absolute error percentage, which is calculated as follows: for every time slice (20 ms for C2075 and 1000 ms for K20c), the absolute error percentage is calculated using the following equation:

$$\text{Error }\% = \frac{|\text{Estimated Power} - \text{Measured Power}|}{\text{Measured Power}} * 100 \tag{4.1}$$

The mean absolute error for an application is then calculated by averaging the values obtained across time slices.

Application-Independent Models

Table 4.5 summarizes the results obtained for the various models on the target GPUs. The values presented in this table are the geometric mean error across all the test applications. We make two observations that hold true for both the GPUs: (i) temperature-aware models consistently produce significantly higher accuracy compared to the basic models, and (ii) linear models produce the highest overall accuracy with a mean error percentage of 4.49%

on C2075 and 6.14% on K20c.

Table 4.5: Mean error % for application-independent models

	C2075		**K20c**	
Models	**Basic**	**Temp-aware**	**Basic**	**Temp-aware**
SLR	17.96	8.59	21.67	9.44
MLR	11.59	**4.49**	18.66	8.29
MLR+I	14.02	6.83	14.74	**6.14**
QMLR	14.83	6.42	15.46	7.82
QMLR+I	19.05	10.31	19.56	8.86

Table 4.6 shows the coefficient terms for the best application-independent models (i.e., MLR for C2075 and MLR+I for K20c). We observe that while modeling the interaction terms helped in improving the accuracy for K20c, their contribution towards overall power is small, as indicated by their disproportionately smaller coefficients. According to these models, one degree Celsius increase in device temperature increased the power consumption by 0.4 W on the C2075 GPU and 0.58 W on the K20c GPU. This can be attributed to the difference in transistor sizes: 40 nm for C2075 and 28 nm for K20c. We note that the linear form of the equations and the parameters used are similar to the CPU power models explored in the past. This indicates the possibility of a generic power model for heterogeneous systems worth exploring in the future.

Next, we present the mean error percentage for the applications individually in Fig 4.5. We observe that even for the more accurate temperature-aware linear models, certain applications (e.g., Eigen) exhibited high error. To understand the nature of this high error, we present the estimated and measured power profiles for QTC on C2075 and Eigen on K20c in Figure 4.6 and Figure 4.7, respectively. We chose these applications to highlight both the positives and the negatives of the models simultaneously. In both cases, we observe that the MLR model accurately estimates the phase shifts in power, but not the exact power values. If we subtract the estimated values by some constant offset, the error percentage

Table 4.6: Coefficient values for the best application-independent models

Device	CYC	II	IE	DRAM-R	L2-R	L2-W	L2-W*IE
C2075	-3.62E-04	4.91E-04	1.54E-03	6.03E-03	1.22E-03	–	–
K20c	5.73E-05	-8.42E-05	2.70E-05	–	–	2.50E-05	9.54E-14

L2-W*II	L2-W*CYC	CYC*IE	CYC*II	II*IE	Constant	Temperature
–	–	–	–	–	8.30E+04	4.01E+02
8.23E-14	-1.41E-13	-3.99E-14	1.25E-13	-2.19E-15	4.20E+04	5.80E+02

Key:

CYC: Number of cycles in which the GPU has at least one active warp
II: Number of instructions issued
IE: Number of instructions executed
DRAM-R: Number of read requests sent to DRAM
L2-R: Number of read requests sent to L2
L2-W: Number of write requests sent to L2
L2-W*IE: Parameter capturing the interaction between L2 writes and instructions executed
L2-W*II: Parameter capturing the interaction between L2 writes and instructions issued
L2-W*CYC: Parameter capturing the interaction between L2 writes and active GPU cycles
CYC*IE: Parameter capturing the interaction between active cycles and instructions executed
CYC*II: Parameter capturing the interaction between active cycles and instructions issued
II*IE: Parameter capturing the interaction between instructions issued and instructions executed

drops dramatically, for example from 32% to 3% for Eigen on K20c. Our results indicate that the application-independent models are robust predictors of power-phase shifts for the workloads under consideration.

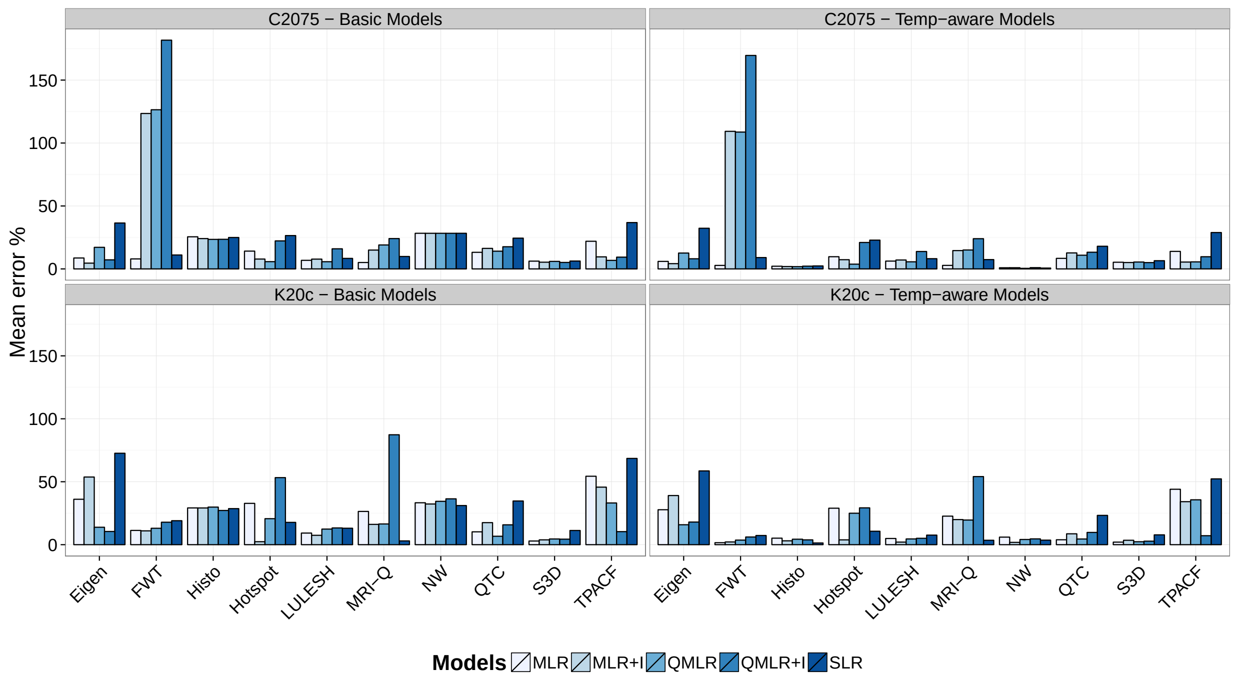


Figure 4.5: Application-independent models. Mean error % for applications shown individually for all evaluated models.

Cost of Portability: We evaluate portable models by restricting the models to include only those events that have a high correlation with power consumption on both the GPUs under consideration. Table 4.7 shows the error percentage achieved for these models. We observe that the cost of portability is quite high. For instance, on C2075, the mean error % increased from 4.49% for the best non-portable power model to 8.22% for the best portable power model on C2075 as shown in Tables 4.5 and 4.7. Likewise, on K20c, the mean error % increased from 6.14% for the best non-portable power model to 9.40% for the best portable power model as summarized in Tables 4.5 and 4.7. This shows that even for successive generations of GPUs, portable models cannot be constructed without sacrificing accuracy.

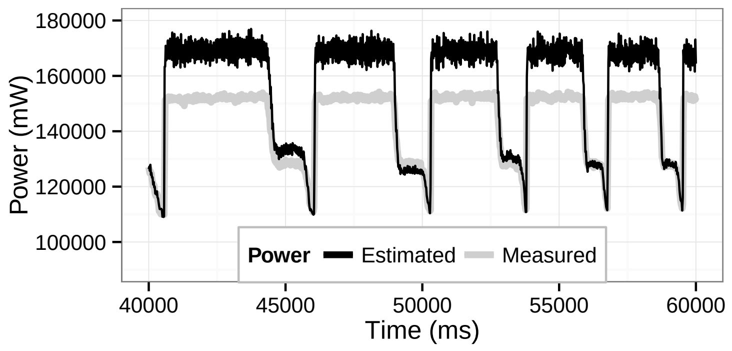


Figure 4.6: Measured power and estimated power from application-independent models for QTC on C2075.

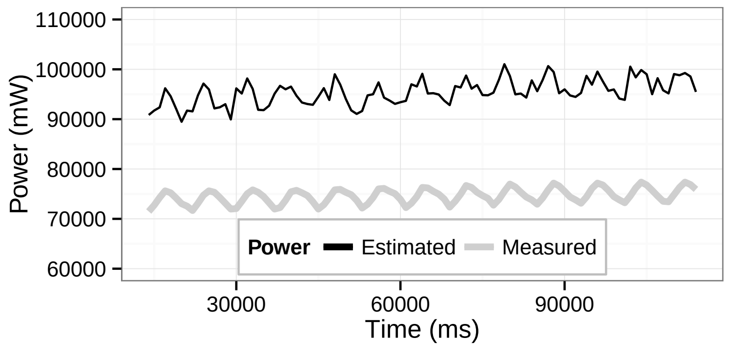


Figure 4.7: Measured power and estimated power from application-independent models for Eigen on K20c.

Overhead: We observe an overhead of less than 0.1% when we profile up to five performance counters (the maximum used by our model) at a sampling frequency of 50 Hz. Our experiments reveal that profiling the performance counters does not induce additional overheads on the GPU.

Application-Dependent Models

In this section, we evaluate if there are benefits to using application-dependent models. We observe that the application-dependent models exhibit higher accuracy than the application-

Table 4.7: Mean error % for portable models

	C2075		K20c	
Models	**Basic**	**Temp-aware**	**Basic**	**Temp-aware**
SLR	17.96	8.59	21.67	9.44
MLR	18.55	8.22	23.36	8.48
MLR+I	18.26	11.15	22.84	9.40
QMLR	16.76	11.24	22.27	9.23
QMLR+I	18.36	11.63	20.87	8.79

independent model. Specifically, the mean error % decreases from 4.49% to 1.02% for the C2075 and from 6.14% to 0.88% for the K20c as shown in Table 4.5 and Table 4.8. Compared to application-independent models, these models are provided with only the most relevant information as their training data. This results in our model accurately estimating the power consumption, as shown in Figure 4.9 and Figure 4.10 and not merely phase shifts as seen previously with Figure 4.6 and Figure 4.7.

Table 4.8: Mean error % for application-dependent models

	C2075		K20c	
Models	**Basic**	**Temp-aware**	**Basic**	**Temp-aware**
SLR	7.32	2.26	3.39	1.49
MLR	4.73	1.62	2.64	1.22
MLR+I	2.94	1.07	2.22	0.92
QMLR	3.04	1.08	2.24	0.96
QMLR+I	2.79	**1.02**	2.17	**0.88**

Figure 4.8 shows the mean error of the various power models for each application. In most cases, the application-dependent models exhibit significantly better accuracy compared to the application-independent models. Notable exceptions include LULESH and S3D, which are composed of several computational kernels with distinct characteristics whereas the other applications are more homogeneous in nature. Such applications may benefit by modeling each computational kernel separately.

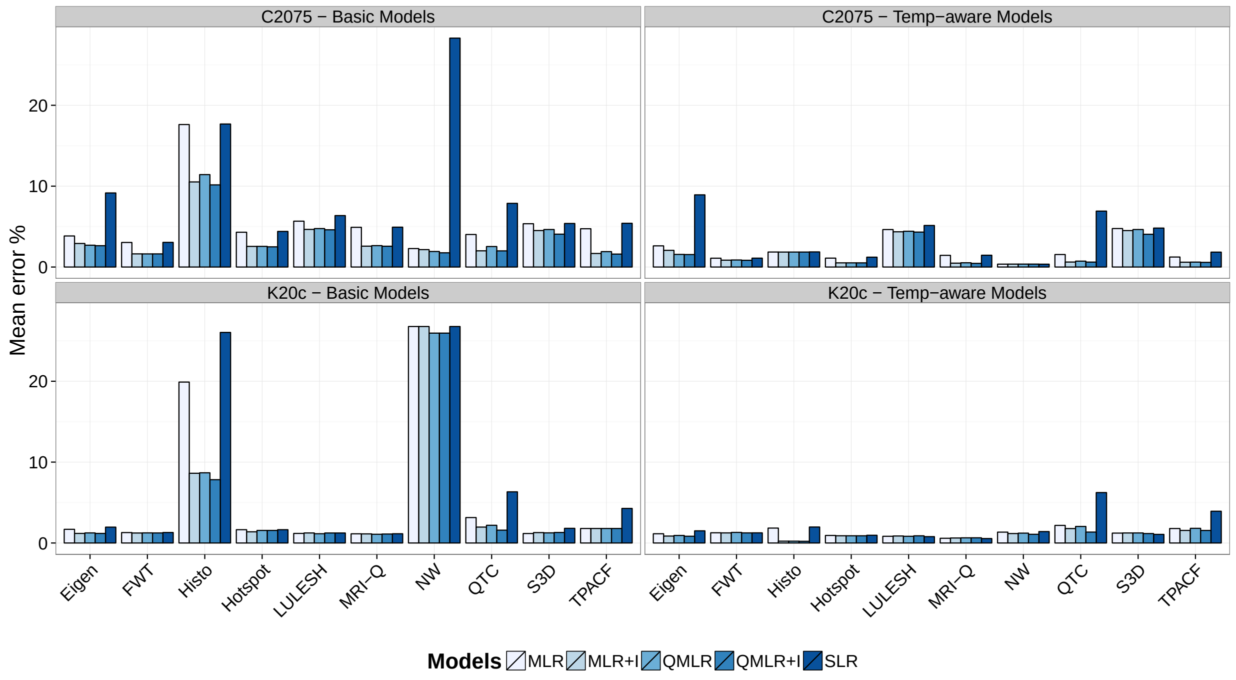


Figure 4.8: Application-dependent models. Mean error % for applications shown individually for all evaluated models.

Constructing Power Models at Runtime

To achieve the high accuracy offered by the application-dependent models, we first construct the model offline once for each application separately. This step can be avoided if we could construct the application-dependent models at runtime. However, for such models to be useful in a runtime system, model construction should not introduce any significant overhead. This means only a few samples may be used to construct the model using simple techniques.

Sample-Size Sensitivity: We measured the sensitivity of MLR to the number of samples used for training. Figure 4.11 shows the cumulative distribution function (CDF) plot for QTC on C2075. The x-axis represents error percentage and the y-axis represents the percentage of estimated values that falls below a given error percent during the testing phase. Models were constructed using the first 50, 100, 200, 400, and 800 samples and tested for the entire

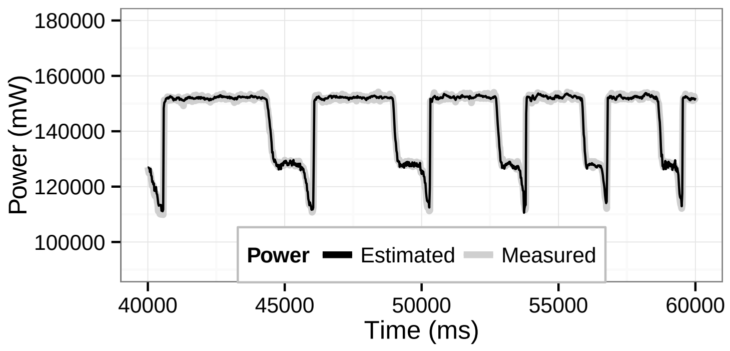


Figure 4.9: Measured and estimated power for application-dependent models for QTC on C2075.

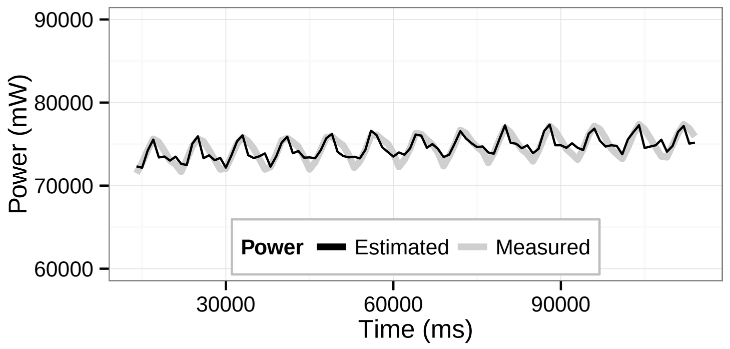


Figure 4.10: Measured and estimated power for application-dependent models for Eigen on K20c.

duration of the application (which consists of few thousand additional sample points). We observe that the accuracy of the MLR model improves only marginally when we use more than 100 samples.

Figure 4.12 shows the sample-size sensitivity for the remaining applications. We observe that the different applications require different number of samples to accurately capture the characteristics of the application. In general, about 100 sample points are sufficient to construct an accurate model for scientific applications, which produces no noticeable overhead. Therefore, model construction is feasible at runtime as the overhead is minimal and the accuracy obtained is high. However, for applications having heterogeneous characteristics,

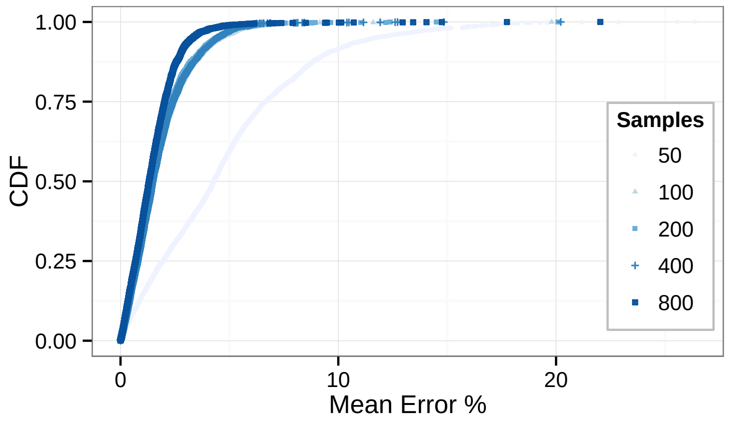


Figure 4.11: Sensitivity of power model to input data set size for QTC

such models need to be adapted dynamically depending on the kernel being executed.

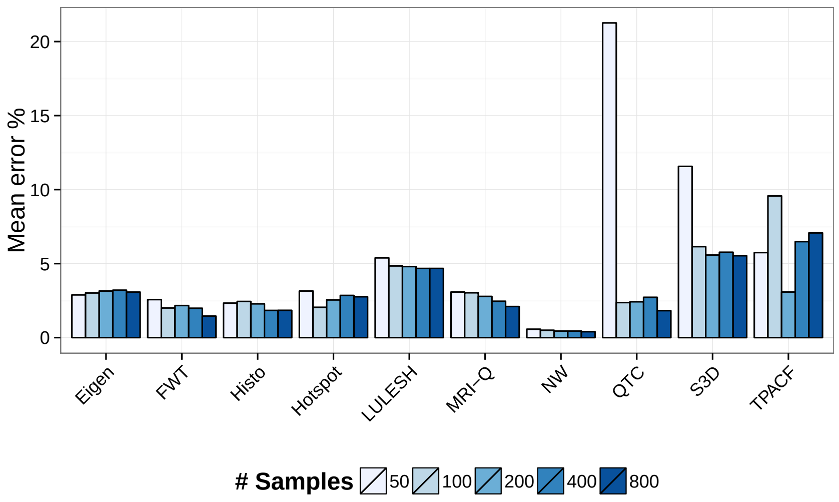


Figure 4.12: Sensitivity of power model to input data set size for all test applications

4.6 Summary

In this chapter, we narrowed the knowledge gap between GPU power models and the existing literature on CPU power models. We presented insights and techniques to address unresolved questions regarding GPU power modeling, including the best modeling functions for GPU power consumption, a maximal set of observable modeling parameters for run-time power estimation, and a set of practical approaches to achieve the desired target error of 5%. We identified system activities that correlate with power consumption of GPU systems, primarily active GPU cycles, instruction-issue rate, instruction-execution rate, DRAM read-request rate, L2-read request rate, L2-write request rate. We found that apart from system activities, the device temperature plays a major role in determining the GPU's power consumption. We found that linear functions involving a few simple parameters are sufficient to model a GPU's power consumption. Specifically, we showed that application-dependent models are highly accurate with a mean error of 1% and a worst-case error of 5%. Finally, we showed that such application-dependent models can be constructed at runtime from data collected during the first two seconds of an application's execution resulting in an average error under the desired target of 5%.

Chapter 5

Reducing Chip-Interconnect Power via Proactive Management

5.1 Introduction

Under initiatives such as the U.S. Department of Energy's DesignForward [47], FastForward [50], and PathForward [116], vendors have proposed exascale system design with larger heterogeneous processors, 3-D stacked high-bandwidth memory, and improved technology nodes to help achieve DOE's exascale goals [139, 155, 156]. Our analysis in this chapter shows such designs will exacerbate a known problem: *interconnect power consumption*. That is, the cost of moving data over the processor's chip interconnect will make it difficult to build an exascale supercomputer that operates under the desired power budget.

Figure 5.1 presents the results from our analysis where we can see that the data movement between the main memory and last-level cache (LLC) is estimated to consume over 34 W for a *futuristic* processor targeting exascale computing. The same application (`FDTD`) is

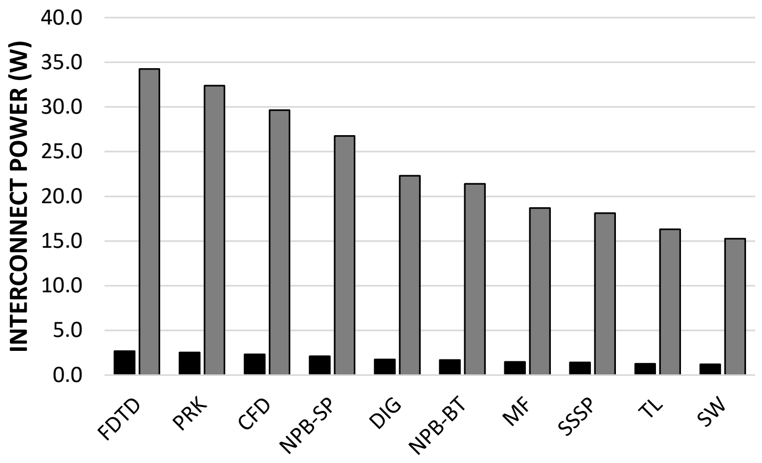


Figure 5.1: Chip interconnect power for transfers between main memory and last level cache for a 2014 GPU and a futuristic GPU

estimated to consume only 2.7 W on an AMD FirePro™ W9100 GPU released circa 2014 [9]. While past research has also identified chip-interconnect power to be a problem in chip design [105, 97, 26, 66, 133], our analysis, presented in Section 5.2, suggests that the power consumed by the chip interconnect could be much larger for exascale processors.

To address this problem, we propose to proactively manage the dynamic voltage and frequency scaling (DVFS) state (also referred to as P-state) of the chip interconnect. Our proposal seeks to use light-weight phase predictors to learn future chip-interconnect traffic requirements from past observations and set the chip interconnect's P-state appropriately. We identify the problems in designing such a phase predictor via detailed characterization studies and design a general predictor that delivers noise-robust forecasting of power consumption. Specifically, our major contributions in this chapter include the following:

- **We present a taxonomy of chip-interconnect traffic by studying 37 different applications.** We identify the challenges in forecasting chip-interconnect traffic: (i) global phase-change behavior, (ii) naturally occurring noise in chip-interconnect traffic, (iii) quantization effects arising from mapping raw chip-interconnect traffic (a continuous value) to discrete P-states, and (iv) the diversity of applications, which makes the building of a one-size-fits-all predictor difficult.

- **We present a novel application of approximate pattern matching to history-table predictors.** The application of approximate matching algorithms to erstwhile history-table predictors helps to address issues related to noise in the observed traffic.

- **We present an extensive analysis of our predictor and tune its various parameters**. Our analysis indicates that treating the history-table predictor as a small cache of rules and outcomes with the least-recently used (LRU) policy for replacing rules is a viable approach for building low-overhead predictors. In addition to reducing the memory footprint and computational overheads, limiting the size of the history table also helps the predictor adapt itself quickly to *global* phase changes.

In addition to the above contributions, we present a quantitative analysis that compares our approach with Markov predictors, another popular approach for proactive management of architectural resources. We also present results from a trace-based simulator that compares a proactive power-management technique using our predictor versus reactive management. Our experimental data shows that we can save over 25.5% (6.0 W) of the chip interconnect's power while reducing performance by only 1.5% and adding less than 0.5 W of power overhead for our proactive technique. A reactive technique, on the other hand, would save slightly more power on the chip interconnect (31.7%), but it does so by underestimating the chip-interconnect traffic which, in turn, degrades performance by 8.0%.

The rest of the chapter is organized as follows. Section 5.2 explains why chip-interconnect power poses a major problem for exascale designs. Section 5.3 presents an overview of our proposed proactive power-management approach. Section 5.4 presents a characterization of chip-interconnect traffic from 37 applications and highlights the challenges in the proactive management of the chip interconnect. In Sections 5.5 and 5.6, we present a Markov predictor and history-table predictor for chip-interconnect traffic management, respectively. Section 5.6 also describes extensions to the history table-based approach to improve the prediction accuracy. For a baseline processor described in Section 5.7, we present simulation results comparing various power management approaches with and without our predictor in Section 5.8. We present our conclusion in Section 5.9.

5.2 Background

In this section, we explain the factors governing chip-interconnect power and highlight the emerging trends affecting these factors. We quantify the impact of these trends by comparing the chip-interconnect power for two general-purpose graphics processors (GPUs) — AMD FirePro™ W9100 GPU from 2014 and another being a *futuristic* GPU targeting exascale computing.

5.2.1 Basics of Chip-Interconnect Power

This section reinforces our findings from Chapter 3. Chip interconnects are wires that connect various modules of a processor (e.g., cores and memory) and are responsible for data movement and transmission of control signals across these modules. The power consumed by the chip interconnect is given by the following formula [100]:

$$P = AF \cdot C \cdot V^2 \cdot F \tag{5.1}$$

Here, AF is the activity factor, C is the effective capacitance, V and F are the operating voltage and frequency of the chip interconnect. Equation (5.1) can also be rewritten as:

$$P = k \cdot D \cdot T \cdot L \cdot V^2 \cdot F \tag{5.2}$$

According to this equation, the various factors governing chip-interconnect power are as follows:

- **Data Volume (D).** As the chip interconnect sends and receives more data, its activity increases and more power is spent on the chip interconnect. This term relates to the AF in Equation (5.1).
- **Chip-Interconnect Length (L).** Data that travels over a longer distance in the chip burns more power in the wires.
- **Toggle Rate (T).** Different patterns sent over the wires produce different toggle rates, which in turn affects active capacitance and thus chip-interconnect power.
- **Voltage (V) and Frequency (F).** The same amount of data sent over the wires consumes different amounts of power depending on the operating voltage and frequency.
- **Constant (k).** The constant factor depends on the material used for the chip interconnect and the process technology in which it was built.

5.2.2 Emerging Trends

While past studies have shown chip interconnects to be a major source of power consumption, recent trends exacerbate this issue, as articulated below.

Bigger Chips. In the past several years, performance improvements in high-end server processors have largely come from transistor scaling, which helped increase the core count in modern processors. However, with the diminishing returns from Moore's Law, it has become harder to increase core count just from transistor scaling. Nevertheless, the number of cores in high-end processors have continued to increase by building larger chips. Previously, building larger chips was cost prohibitive due to yield issues. However, the recent shift from designing monolithic chips to modular chips via technologies such as multi-chip module (MCM) [16] and chiplet designs [151] have made building larger chips possible. A side effect of the above trend is that, data has to move through a longer distance on larger chips on an average. Consequently, as implied by Equation (5.2), the chip interconnect would consume more power on a per-bit basis as compared to previous generation processors.

Higher Memory Bandwidth. Historically, DRAM bandwidth has lagged behind compute performance improvements, which has kept the proportion of power spent moving data between memory and the last-level cache (LLC) low. However, with the recent introduction of stacked memory architectures, such as high-bandwidth memory (HBM), hybrid memory cube (HMC), and wide I/O (WIO), the available memory bandwidth has seen a sharp increase [43]. For instance, data in [13] shows that a single stack of HBM offers *four times* the bandwidth of GDDR5. While the off-chip data movement energy is significantly lower for HBM compared to GDDR5 on a per-bit basis, the on-chip data movement cost (e.g., between the memory controller and last-level caches) remains the same. As a result, higher memory bandwidth will result in a larger proportion of power spent on the wires.

Failure of Wire Scaling. While the end of Dennard scaling has resulted in a slowdown of the rate at which transistor's power scales, Borkar showed that wires are scaling at an even slower rate [26]. Therefore, the amount of energy (and thus, power) spent in on-chip data movement will increase as a proportion of total energy (and power) consumption of the chip.

5.2.3 Analysis of Chip-Interconnect Power on Emerging Hardware

Here we present the power consumed on on-chip wires while moving data from the GPU's memory to the L2 cache for the ten-most data-intensive applications from our study. The data is presented for two generations of GPUs — AMD FirePro™ W9100 GPU that debuted in late 2014 and a futuristic GPU targeted at exascale computing nodes. Table 5.1 summarizes the key parameters that affect the chip interconnect power.

Table 5.1: Reference GPU vs. futuristic GPU

Chip Interconnect Parameter	**Ref. GPU (c. 2014)**	**Futuristic “Exascale” GPU**
Technology Node	28 nm	14 nm
Voltage	1.125 V	1 V
Frequency	0.93 GHz	1.00 GHz
Bandwidth	320 GB/s	3810 GB/s
Avg. distance (up to L2)	11.5 mm	28.5 mm

As Figure 5.1 shows, these applications spent only 1.2-2.7 W of power in moving data and delivering it to the L2 cache on the reference GPU from 2014. However, in the futuristic GPU, the same applications would consume 15.3-34.3 W of power, an order-of-magnitude increase. Clearly, chip-interconnect power is a major concern that needs to be immediately addressed.

5.3 Proactive Chip-Interconnect Power Management

With chip interconnect consuming 34 W out of a typical 200 W TDP for real applications, active power management of the chip interconnect is necessary in order to achieve the exascale goal under 20 MW. Like past work [105], our proposal seeks to operate the chip interconnect in a separate DVFS domain. Since the cost of dynamic V-f scaling is high (particularly for larger chips), we limit the periods of V-f scaling in this work to the time between GPU kernel launches. This approach has the added advantage of being able to overlap V-f scaling overheads with kernel launches. Because the characteristics of an application changes drastically across GPU kernels, we also propose to predict the chip-interconnect traffic required for a given kernel ahead of time. Thus, the central contribution of this paper is in building a light-weight predictor that is capable of forecasting the traffic requirements accurately. While past research in phase prediction has studied the problem of forecasting characteristics of an application ahead of time [73, 77, 75, 71, 143, 134], our contribution differs in three aspects. First, we make predictions at kernel boundaries while past work focused on predicting at regular intervals of time. This means we make predictions when an application phase changes, an arguably harder problem than making a majority of the predictions when an application is in a stable state. Second, the characteristics of applications have changed over time, making it necessary to revisit phase prediction. Third, our work concerns the use of predictors to manage chip-interconnect power, a less-explored component within the processor. Upon predicting the chip-interconnect traffic for an upcoming GPU kernel launch, we set the DVFS state of the chip interconnect appropriately to reduce its power usage. In the next section, we present challenges and opportunities in predicting the chip-interconnect traffic a priori.

5.4 Characterizing Chip-Interconnect Traffic

In this section, we characterize the chip-interconnect traffic of our target applications along four dimensions—pattern type, noisiness, pattern length, and dynamic traffic range—in order to identify challenges in forecasting traffic ahead of time. We select 37 applications from

Rodinia [33], Parboil [148], PolyBench [123], OpenDwarfs [52], Mantevo [62], Phoronix [90], DOE Exascale Proxy Apps [125], and AMDAPP SDK [12] for this study. Table 3.3 summarizes these applications while Figure 5.2 presents the raw traffic traces.

Table 5.2: Characterization of chip-interconnect traffic for target applications

App.	**Source**	**Pattern Type**	**Length**	**Noise**	**Range**
ADI	PolyBench	Global	73	0.4	0.4
AMG	ProxyApps	Global	74	0.1	0.9
BC	Pannotia	Seasonal	8	0.1	0.4
BFS	Rodinia	Trend	3	0.0	0.1
BLAS	AMD SDK	Irregular	792	0.4	0.7
BWA	OpenDwarfs	Global	2	0.2	0.1
CFD	Rodinia	Global	7	0.0	0.8
CLF	Mantevo	Seasonal	130	0.1	0.9
CL3D	Mantevo	Irregular	341	0.1	0.5
DBLAS	AMD SDK	Hybrid	3	0.4	1.0
DIG	PolyBench	Irregular	1	0.1	0.2
FDTD	PolyBench	Seasonal	2	0.1	0.7
FWT	AMD SDK	Irregular	3	0.7	0.1
GAU	Rodinia	Trend	8	0.2	0.1
GS	Polybench	Seasonal	3	0.0	0.2
GRO	Other	Seasonal	1	0.0	0.1
HMM	Other	Global	5	0.1	0.7
JCB	PolyBench	Uniform	1	0.9	0.3
JUL	Phoronix	Irregular	3	0.8	0.0
LU	PolyBench	Seasonal	2	0.0	0.9
MNDL	Phoronix	Irregular	87	0.9	0.0
MF	AMD SDK	Seasonal	8	0.0	0.7
NMF	Other	Irregular	34	0.7	0.9

Pattern Type. Our proposal involves exploiting repeating traffic patterns in applications for proactive power management. These patterns can be of different types. First, the least complex is the *uniform* pattern, where traffic remains largely constant, as seen in `JCB` in Figure 5.2. Next, in *seasonal* type, we see a regular repeating pattern as in the case of `MF`. The *seasonal* pattern may also be seen in conjunction with a *global* pattern where different seasonal patterns are seen over different periods of time as in the case of `HMM`. Sometimes,

Table 5.3: Characterization of chip interconnect traffic for target applications (contd.)

App.	Source	Pattern Type	Length	Noise	Range
NPB-BT	NPB	Seasonal	19	0.0	0.9
NPB-CG	NPB	Seasonal	4	0.1	0.9
NPB-LU	NPB	Irregular	558	0.8	0.5
NPB-MG	NPB	Seasonal	106	0.2	1.0
NPB-SP	NPB	Seasonal	14	0.0	0.9
PRK	Pannotia	Seasonal	2	0.7	0.1
PB-BFS	Parboil	Irregular	1	0.0	0.0
SCN	OpenDwarfs	Global	3	0.0	0.5
SSSP	Pannotia	Global	6	0.0	0.7
SC	Parboil	Seasonal	3	0.0	0.7
SW	Other	Seasonal	5	0.0	0.7
TL	Mantevo	Seasonal	62	0.0	0.8
TL3D	Mantevo	Irregular	60	0.7	0.2
TRD	OpenDwarfs	Trend	1	0.0	0.9

seasonal patterns appear in conjunction with a long-term *trend* as in the case of `GAU`. In this example, we see a long-term decrease in traffic. Finally, we have the *irregular* patterns as in the case of `PB-BFS` where there are no repeating patterns.

Noisiness. A perfectly repeating pattern is rarely seen in the real world for various reasons. The state of the memory hierarchy, for instance, could differ with each GPU kernel launch resulting in differing amounts of data fetched from the main memory over the chip interconnect. We term these perturbations in pattern as noise. To quantify the degree of noise in the traffic patterns, we borrow concepts from the academic field of signal processing. In signal processing applications, a signal is correlated with itself (i.e., *autocorrelation* (ACF)) with a delay factor (known as *lag*) to find the presence of repeating patterns. A high value for ACF indicates the presence of patterns. Signals obscured by noise tend to have lower ACF values. In a similar manner, we find the ACF for chip-interconnect traffic and calculate the noise for an application as $1 - \text{ACF}$. An application exhibiting high noise on the chip interconnect is likely to be difficult to be proactively managed. Our goal in this dissertation

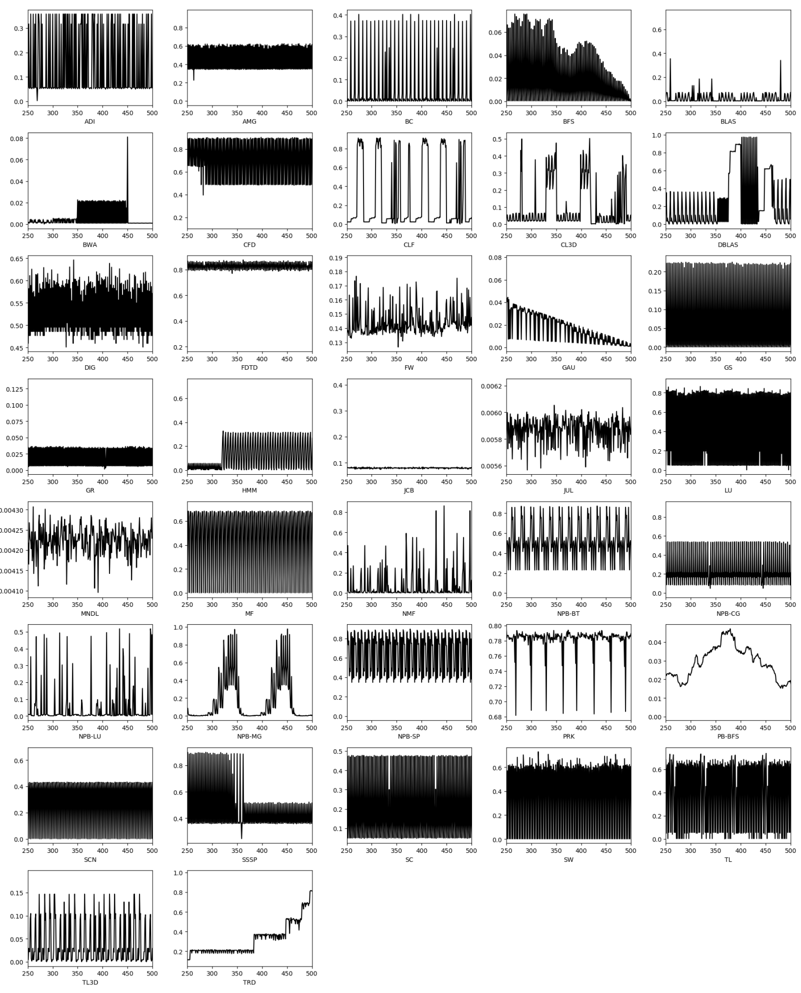


Figure 5.2: Traffic traces for applications under study

is to design a predictor that is robust to the presence of occasional noise.

Pattern Length. Patterns can vary in length. For instance, `MF` shows a simple repeating pattern of length 2, whereas `NPB-MG` shows a more complex pattern of length 53. For our characterization, we calculate the length of a pattern as the *lag* at which we observe the highest ACF value for the application.

Dynamic Traffic Range. The dynamic range for an application is the difference between the highest and lowest chip-interconnect traffic. The range is normalized to a scale of 0-1.

5.4.1 Challenges in Forecasting Chip Interconnect Traffic

We identify the following challenges in forecasting chip-interconnect traffic ahead of time:

- **Complex Pattern Types.** While simple predictors may work for *uniform* and *seasonal* patterns, the more complex patterns are harder to predict. For instance, a predictor trained during the initial phases of an application may start mispredicting the traffic upon a *global* phase change. Similarly, the *absence* of any pattern (i.e., *irregular* pattern) needs to be recognized by the predictor to avoid adversely impacting the performance of an application due to misprediction.

- **Diversity in Characteristics.** Applications differ widely in pattern lengths and dynamic range. Designing a predictor that looks for longer patterns may not work as well for applications exhibiting shorter patterns and vice versa. Designing a predictor that works well in different scenarios may be challenging.

- **Noisiness.** Another important aspect of these applications is that over one-third of the application contains noise of over 20%. This means that when we try to match a previously seen pattern with the current execution window, there oftentimes will not be a match. The accuracy of the predictor could drastically drop even in the presence of a small amount of noise.

- **Quantization Effect.** Chip-interconnect traffic is a continuous value; DVFS states are discrete. Sequence data prediction techniques that have been reported to be successful in power management of other components of a processor [72, 74, 76, 135] operate only on discrete values rather than continuous value. Therefore, we quantize (or bin) chip-interconnect traffic into non-overlapping bands (i.e., levels) before we make predictions. This quantization of continuous values into discrete bins adds noise to be pattern. For instance, when we binned interconnect traffic 0-50 GB/s to level 1 and 50-100 GB/s to level 2, we sometimes observed GPU kernels that fluctuates around 50 GB/s depending on the state of the cache before the GPU kernel begins execution. In these cases, the kernel may show up as level 1 or level 2 with a random probability.

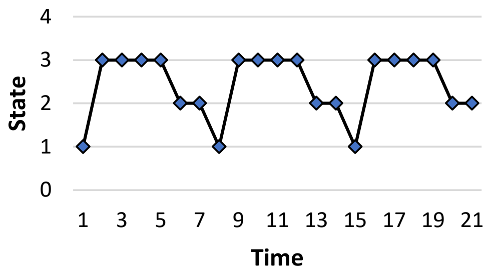
In this dissertation, we seek to design a predictor that addresses the above challenges in phase prediction.

5.5 Markov Predictor

Markov predictors can forecast future states of a stochastic system based on its current state. They can be easily implemented in hardware and are used for various speculation-based optimizations such as branch prediction [34], prefetching [79], dynamic cache sizing, and processor width adaption [143]. For power management, Markov predictors have been used by researchers for disks [53] and many-core processors [25]. For reasons described in Section 5.3, we revisit the applicability of such predictors for proactive management of chip interconnects.

Next, we briefly explain the operation of Markov models for predicting-chip interconnect traffic with a *toy* application whose traffic is shown in Fig. 5.3(a). The chip-interconnect traffic for this application can be in any one of three levels—*low* (state 1), *medium* (state 2), or *high* (state 3). This application sees a repeating pattern of length *seven*: *1-3-3-3-3-2-2*. The Markov model for this pattern can be represented as a transition matrix shown in Fig. 5.3(b).

In general, the Markov model for a system with **N** states is represented as an **NxN** matrix with the value at row **I** and column **J** representing the probability of transitioning from state



(a) Interconnect Traffic

State	1	2	3
1	0.00	0.00	1.00
2	0.50	0.50	0.00
3	0.00	0.25	0.75

(b) Transition Probability Matrix

Figure 5.3: Illustration of Markov model for interconnect traffic prediction

I+1 to state **J+1**. If the application is currently at traffic level **I**, we predict the upcoming traffic level as level **J** which has the highest transition probability starting at level **I**. When there are ties, the larger value of **J** is conservatively chosen so as to not adversely impact the performance by under-provisioning the chip interconnect. While the Markov model itself is powerful enough to forecast multiple states into the future, our implementation predicts only the next one state as it is sufficient for our purposes. In our implementation, the transition probability matrix is implemented using simple counters and is updated upon each observation of chip-interconnect traffic. While updating the counters after each observation incurs a sizable overhead, this implementation provides us with an upper bound on the prediction accuracy of Markov models.

5.5.1 Accuracy of Markov Predictor

In this subsection, we present the accuracy of Markov model for forecasting the chip interconnect's traffic. In our evaluation, predictions are made in the discrete domain and the traffic bands are assumed to be of uniform width. The predictions begin after the first 100 GPU kernel launches, which constitute the warm-up phase for the predictor.

Fig. 5.4 shows the percentage error for Markov predictor in predicting the chip-interconnect traffic of 37 different applications listed in Table 3.3. The results presented here is assume that the chip interconnect supports 10 distinct DVFS states. The metric of choice is the percentage error in prediction which is calculated as follows:

$$\text{Error \%} = \frac{100 * |\text{Predicted State} - \text{Actual State}|}{\text{Actual State}} \tag{5.3}$$

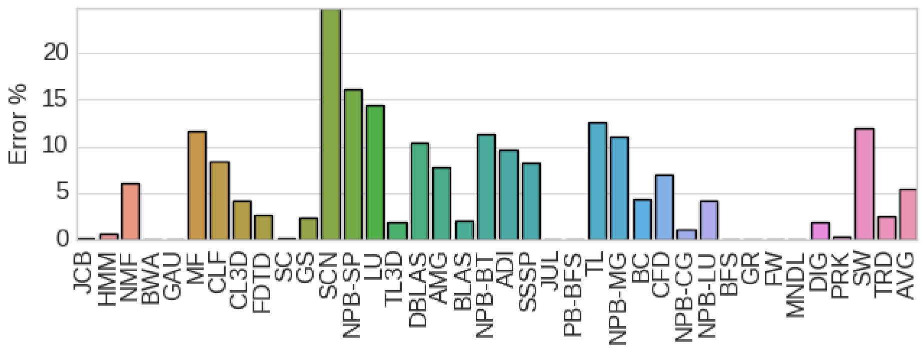


Figure 5.4: Markov model cccuracy (10 DVFS states)

The average prediction error for Markov models is 5.4% while the maximum error we observe is 24.9% as in the case of `SCN` and the minimum error observed is 0% as in the case of several applications such as `BWA`, `GAU`, `JUL`, `PB-BFS`, `BFS`, `GR`, `FW`, and `MNDL`.

`SCN` application exhibits a high error due to its *global* phase-change behavior. Fig. 5.5 depicts the challenge with `SCN` where during the initial phase of the application, a pattern length of 2 and a dynamic range of 0.4 is observed, whereas in the latter phases a pattern length of 3 and a dynamic range of 0.5 is seen. The pattern seen in the initial phase is fairly simple and the Markov model accurately predicts the traffic 100% of the time. The latter phases also show a simple pattern; however, the Markov model is strongly influenced by early observations and keeps predicting level 5 as the next state whenever level 1 is seen. The Markov predictor takes a long time to realize that a global phase change has occurred. Therefore, due to the *long-term memory* of Markov models, applications with *global* phase change behavior (e.g., ADI) tend to exhibit a high error. It is worth noting that the problem is less severe when the global phase change results in an entirely new pattern.

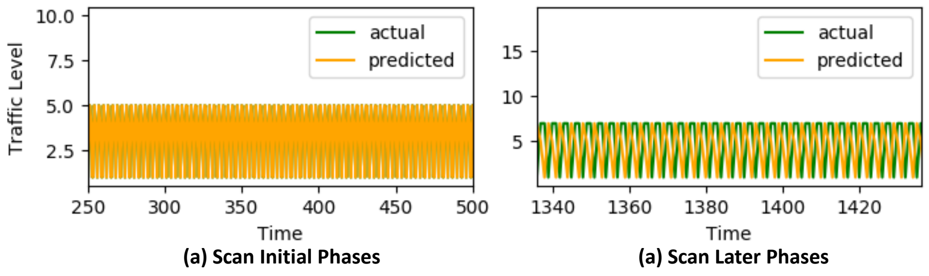


Figure 5.5: Markov predictions at the beginning and end of the Scan application

Another cross-cutting theme seen in the results is that regardless of the pattern *type*, wherever the dynamic traffic *range* is very low (e.g., less than 0.1), the Markov model shows high accuracy. This is because when raw traffic is binned into discrete bins, even *irregular* applications behave like *uniform* applications from the predictor's point of view. Several applications with *irregular* pattern with high dynamic traffic *range* (e.g., BLAS) also show a comparatively low error. This is because of the nature of irregularity in these applications. These applications exhibit uniform traffic with occasional non-periodic deviations. Since the Markov model optimizes well for the common case, the accuracy for this type of application is high.

Another problem in forecasting chip-interconnect traffic that is not adequately solved by the Markov predictor is related to the issue of quantizing continuous traffic values into discrete bands. Suppose a commonly executed GPU kernel draws data from the chip interconnect at a certain rate. If this traffic is close to either ends of a discrete band, then the observed level of chip interconnect traffic will fall into two different bands even though the raw traffic is roughly the same. Markov model shows a low accuracy in prediction when this happens as in the case of NPB-SP.

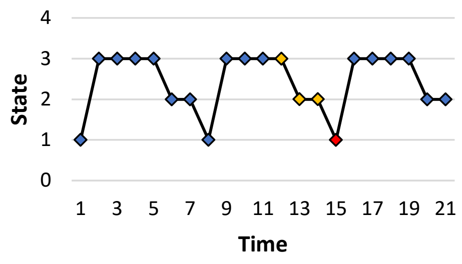
5.6 History Table-based (HT) Predictor

History table-based (HT) predictors are some of the most commonly used predictors in branch prediction [30]. In our work, we adapt HT predictor to predict chip-interconnect traffic. Compared to branch prediction, predicting the chip-interconnect traffic is an arguably harder problem because in branch prediction, the predictor makes decisions between one of two choices: branch *taken* or *not taken*. Furthermore, when encountering program structures such as *loops*, one frequently takes the branch which makes it easier for the branch predictor to achieve a high accuracy. In contrast, in our problem, the predictor needs to predict the traffic level for the upcoming kernel which is not a binary decision. In the rest of this section, we introduce the basic operation of the table-based predictor, explain some commonly encountered problems in predicting chip-interconnect traffic, and propose optimizations that addresses these problems.

5.6.1 A Simple HT Predictor

First, we explain the basic operation of a HT predictor (HTP) with an example. Let's consider the same *toy* application and its chip-interconnect traffic (discretized into three states) seen for the Markov predictor. Recall this application has a repeating pattern of length 7 (namely, *1-3-3-3-3-2-2*). We now explain how this pattern is recorded in the history table and how it gets used to make proactive decisions.

The history table consists of two primary fields—pattern (also referred to as rule) and outcome—that are essential for its operation. HTP operates as follows. First, the HTP observes the traffic of each kernel via performance monitoring counters (PMCs). HTP determines the discrete level corresponding to this traffic and pushes this information to a FIFO queue-like structure known as the active window. Once the active window reaches a predetermined length (a configurable parameter known as pattern length or rule length), the HTP compares the active window with the pattern entries in the table. If a match is found, then the outcome column tells HTP the expected traffic level for the upcoming kernel. In the example shown in Fig. 5.6, we have a rule length of 4. After the end of the fifth kernel, we have the first entry populated in the table as rule = *1-3-3-3* and outcome of the rule as *3*. At the end 11th kernel, the active window would be at *1-3-3-3* again. The active window is compared against the entries in the table and we find a match. The outcome is estimated to be *3* and the chip interconnect is provisioned appropriately. If there is no match to be found, we overprovision the chip interconnect to the highest state possible to avoid adversely affecting the performance.



(a) Interconnect Traffic

Active Window | 3 | 2 | 2 | 1 |

Pattern	Outcome	Frequency
1 3 3 3	3	2
3 3 3 3	2	2
3 3 3 2	2	2
3 3 2 2	1	2
3 2 2 1	3	1

(b) History Table

Figure 5.6: An instantiation of history table-based (HT) predictor

Sometimes as an application executes, we see that there are different outcomes for the same patterns at different points of execution. In this case, we will have multiple matches for the same active window. The ties in this case are broken by adding a separate field to the history table known as *frequency*. Each time an outcome is observed, the *frequency* counter gets incremented by one. This way, whenever there is a tie, we optimize for the common case.

5.6.2 Prediction Accuracy of HT Predictor

Fig. 5.7 shows the percentage error for HT predictor when 10 DVFS states are used. The mean error across all applications is less than 3.5% compared to 5.4% for Markov predictors. The HT predictor performs better on applications with *global* phase change as in the case of `SCN` as it does not have problems associated with having a long-term memory.

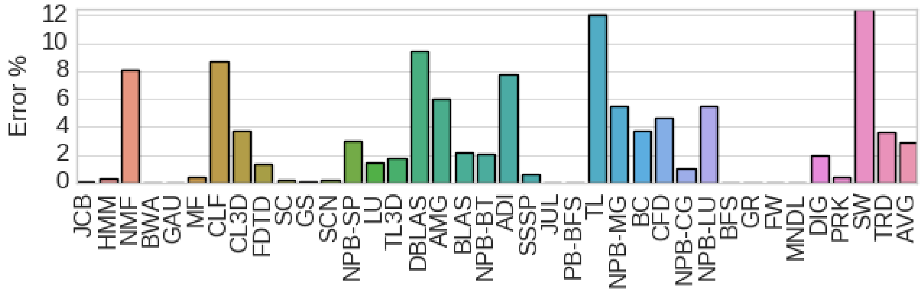


Figure 5.7: Prediction accuracy of the history table (HT) predictor when using 10 DVFS states

Finding the optimal rule length. As seen in our characterization study, applications show patterns of varying length. Our goal here is to tune the *rule length* to find a configuration of table that works for all applications. The results from the tuning experiment is presented in Fig. 5.8, which shows the prediction accuracy of the HT predictor for different rule lengths. From this experiment, we conclude that shorter rule lengths offer the best prediction results even for applications where the repeating pattern is long. This is because a long repeating pattern can be represented as a combination of several short patterns albeit separated by other patterns. For all subsequent analysis performed in this paper, we fix the rule length as 4 which provides the best prediction accuracy.

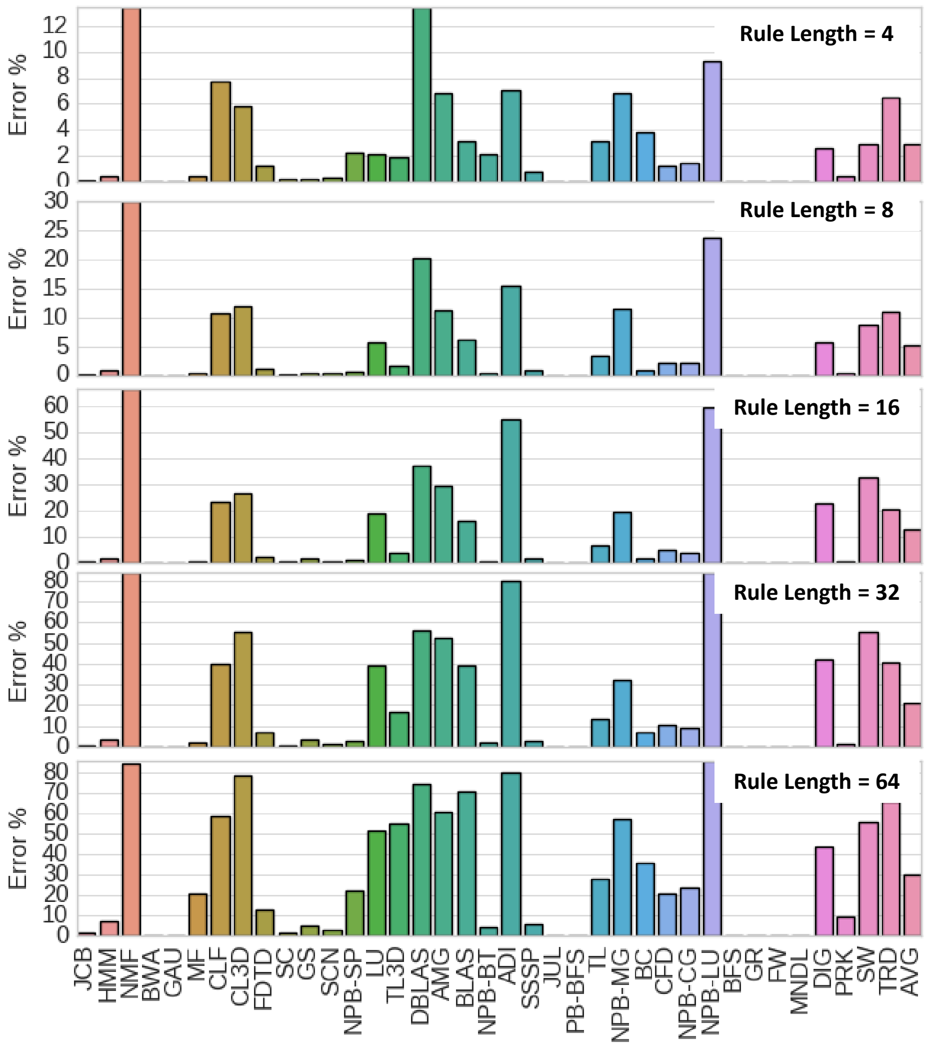


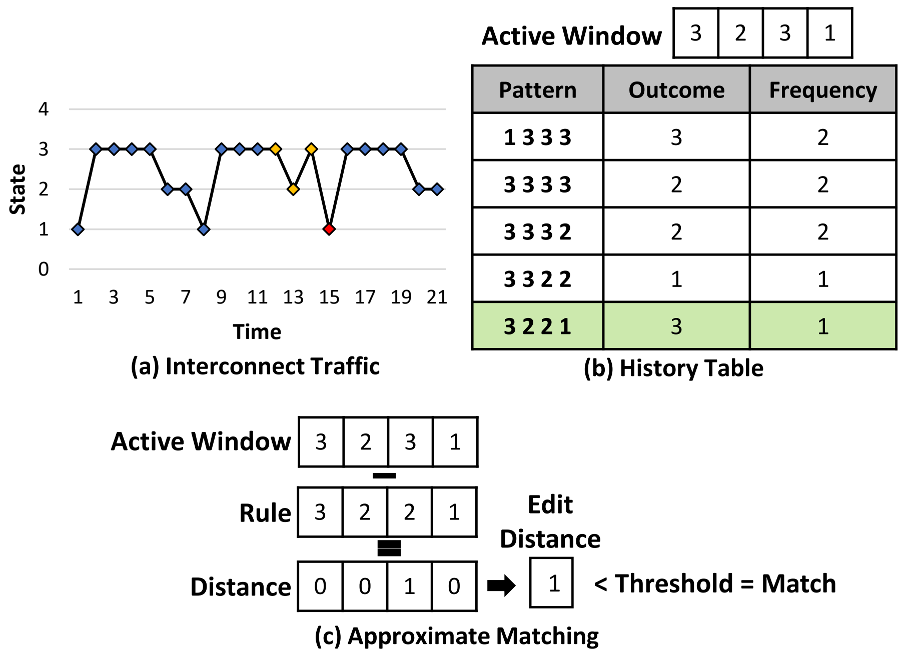
Figure 5.8: Accuracy vs. rule length for the HT predictor

5.6.3 Approximate Pattern Matching to Address Real-World Noise

The basic version of the HT predictor does not perform well in one case—when the application has a lot of noise. To address this problem, we introduce approximate pattern matching to the history table. Consider the traffic pattern shown in Fig. 5.9(a). In this pattern, we observe some noise (due to reasons such as quantization effect) in what would otherwise be a regular repeating pattern. Due to this noise, after executing the 15th kernel, the active window would be at *3-2-3-1* which does not have a match in the history table. In fact, there would be no match in the history table for the surrounding three kernels as well. The presence of noisy data in one of the kernels would result in misprediction for four kernels, which could be easily avoided.

To solve this problem, we propose to use approximate pattern matching instead of finding exact matches when comparing the active window against the rules in the history table. We calculate the edit distance between the active window and the rules in the table as shown in Fig. 5.9(c). If the edit distance is less than a threshold, we consider that a match. In our implementation, we allow a maximum edit distance of *one* for a successful match. With this approach the active window at *3-2-3-1* would match with the rule *3-2-2-1* and correctly predicts the next state as *3*.

The results for approximate pattern matching-based HT predictor are shown in Fig. 5.10. The applications that saw a significant improvement in prediction accuracy include `NMF`, `CL3D`, `DBLAS`, `AMG`, `NPB-MG`, `NPB-LU`, and `TRD`. On the other hand, applications such as `TL` and `SW` saw a decrease in prediction accuracy. In the latter two cases, the approximate pattern matching inadvertently matched with a different rule when an exact match was available. Nevertheless, the approximate pattern matching predictor reduced the average prediction error from 3.5% for exact matching predictor to 2.7%.



Pattern	Outcome	Frequency
1 3 3 3	3	2
3 3 3 3	2	2
3 3 3 2	2	2
3 3 2 2	1	1
3 2 2 1	3	1

Active Window	3	2	3	1
Rule	3	2	2	1
Distance	0	0	1	0

Figure 5.9: History table-based (HT) predictor with approximate matching to handle real-world noise

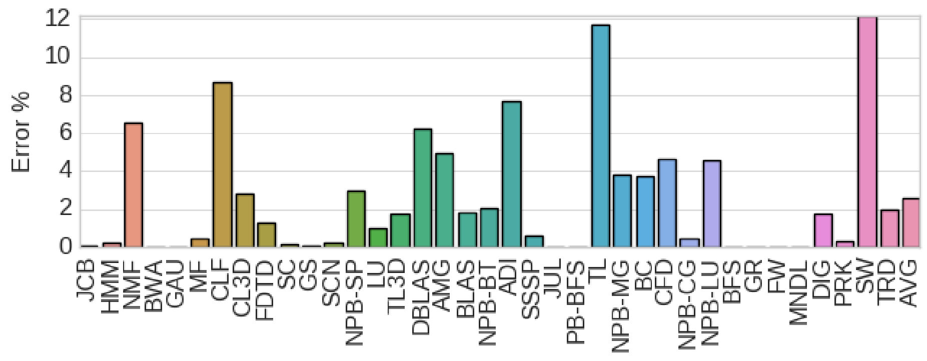
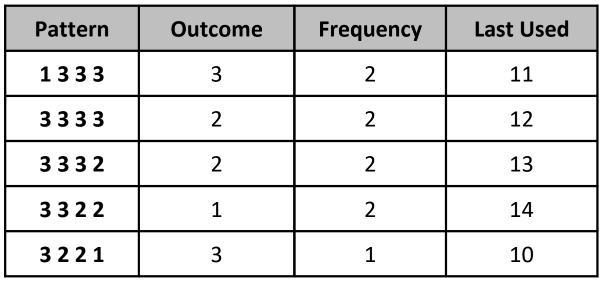


Figure 5.10: Approximate history table (HT) prediction accuracy for 10 DVFS states

5.6.4 HT predictor as a space-limited cache of rules

So far, we assumed that the history table can hold an infinite number of entries. However this is unrealistic in practice for two reasons: the memory footprint of the table and the cost of performing a search for matching patterns becomes prohibitive when we have infinite entries in the table. Therefore, we introduce a modification to the history table. We add a last used field, as shown in Fig. 5.11, and limit the number of entries in the table. We adopt a least recently-used (LRU) algorithm to replace old unused entries. The HT predictor now acts as a small cache of rules and outcomes.



Pattern	Outcome	Frequency	Last Used
1 3 3 3	3	2	11
3 3 3 3	2	2	12
3 3 3 2	2	2	13
3 3 2 2	1	2	14
3 2 2 1	3	1	10

Figure 5.11: Modifications to the history table for limiting the size of the table and associated overheads

Fig. 5.12 shows the average prediction error across 37 applications for space-limited HT predictor. We observe a point of diminishing returns upon reaching 125 entries. Also it is worth noting that the average prediction error is 2.1% for the space-limited HT predictor whereas for an unrestricted predictor, the error increases to 2.7%. In addition to saving memory footprint and search overheads, the replacement policy also helps deal with the long-term memory issues seen in other predictors.

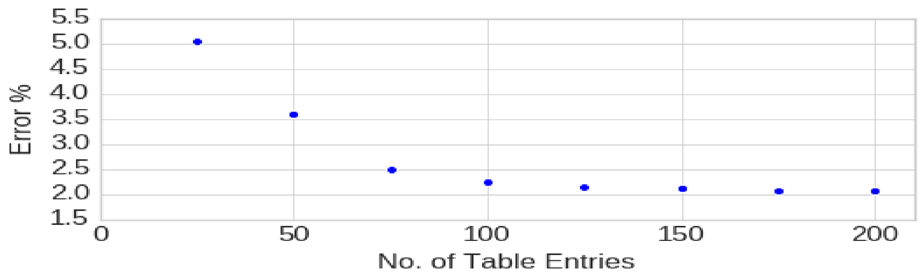


Figure 5.12: Limiting the table size of HT predictor to reduce memory footprint and search overhead.

5.7 Experimental Setup

In this section, we perform a trace-based simulation of the chip interconnect and compare the power savings and the performance offered by proactive and reactive power-management techniques. This section describes the architecture and its chip-interconnect parameters used for our evaluation. We also describe how the traces were collected for the simulation.

5.7.1 Architecture Evaluated

For our analysis, we assume a large heterogeneous processor composed of CPUs and GPUs similar to those proposed for exascale systems [139, 155]. An illustration of the heterogeneous processor used for our analysis is presented in Fig. 5.13. At the center are 8 CPU cores and at either sides of the CPUs are clusters of 4 GPUs each. Each GPU is assumed to be composed of 32 compute units (CUs) for a total of 256 CUs for the entire heterogeneous processor. On the top of each GPU chiplet, we also assume the existence of 3D-stacked DRAM such as some version of high-bandwidth memory (HBM). Each stack of DRAM is assumed to operate at 0.9 V and 1000 MHz providing a bandwidth of 0.5 TB/s. The eight stacks (one per chiplet) together provide an aggregate bandwidth of 4.0 TB/s. The parameters of this heterogeneous processor are provided in Table 5.4.

Table 5.4: Architectural parameters

Parameter	**Value**
Total number of CUs	256
CUs per chiplet	32
Total number of chiplets	8
Voltage (V)	0.9
Frequency (MHz)	1000
HBM bandwidth per stack	0.5
Total HBM stacks	8
Overall HBM bandwidth (TB/s)	4.0
Technology node (nm)	7

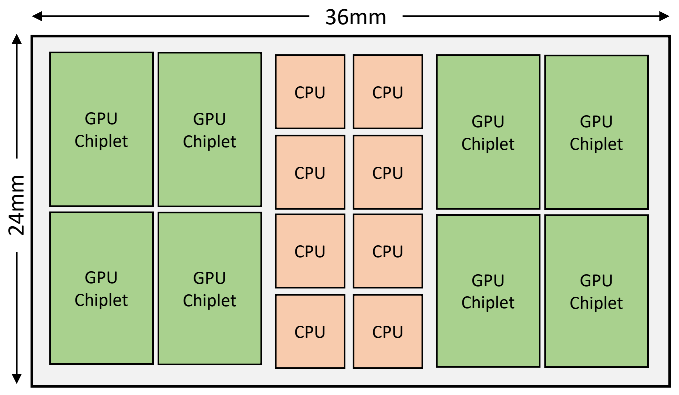


Figure 5.13: Pictorial representation of the target heterogeneous processor

Data from the stacked memory is assumed to be sent to the processing units through chip interconnects on an active interposer layer similar to the ones found in literature [78]. The chip interconnect is assumed to be built with 14 nm technology compared to 7 nm technology for the rest of the processor. This is consistent with server chips available in the market today (2019). On average, we assume that to deliver data from the high-bandwidth memory to the L2 cache (the last level within the GPU), the data has to travel a distance of 28.5 mm. The interposer is assumed to operate at 1000 MHz and 0.9 V by default. Assuming an energy of 95.45 fJ per bit per mm, this translates to a maximum wire energy of 82.9 W. The power consumed on this chip interconnect is the target of optimization in this paper. A summary of the key parameters of the chip interconnect is provided in Table 5.5.

Table 5.5: Chip interconnect parameters

Parameter	**Value**
Number of nodes	16
Number of links per node	128
Link width	8 bytes
Chip interconnect Voltage (V)	0.6-0.9
Chip interconnect Frequency (MHz)	100 - 1000
Chip interconnect Technology node (nm)	14
L2-HBM distance	28.50
Energy/bit/mm (for interposer)	95.45
L2-HBM max. power	82.90

5.7.2 Chip-Interconnect Traffic Trace Collection

The raw chip-interconnect traffic trace was collected on a Hawaii-generation AMD GPU using performance counters and translated into chip-interconnect utilization. The utilization is assumed to be the same on the evaluated *exascale* architecture as well. This is based on two implicit assumptions. First, we assume that the performance of a GPU-accelerated program and thus its memory usage scales uniformly with increasing CU count. Second, we assume that a NUMA-aware programming model for GPUs takes time to evolve and programmers continue to write code that results in memory accesses from a CU spread across memory stacks. When programmers rewrite their code to ensure that accesses go to a memory stack that is closer to the CU that the request originates from, the average distance that the data has to move reduces. However, given the pace at which GPU applications are modified to take advantage of the latest GPU innovations, we believe this change will happen over several years and for the foreseeable future our assumption holds good.

5.8 Results

We perform a trace-based simulation of the targeted architecture and compare the power savings and the performance offered by proactive and reactive power-management techniques. For the chip-interconnect power model, we use our model developed in Chapter 3 and update the parameter values to represent 14 nm technology. The following methods are compared in this section.

Baseline. In the baseline case, we assume that the DVFS settings of the chip interconnect is set to the highest state possible. That is, the chip interconnect is set to 0.9 V and 1000 MHz throughout the course of execution of an application.

Reactive. In a reactive approach, the voltage can vary anywhere between 0.6 V to 0.9 V and frequency between 100 MHz to 1000 MHz. The reactive technique sets the frequency of the chip interconnect (and the corresponding voltage) to match the chip-interconnect traffic requirements of the most-recently observed GPU kernel.

Proactive (Our Approach). In the proactive approach, the same range of voltage and frequency values as the reactive approach is allowed. The proactive power-management technique uses our proposed rule-based, space-limited, approximate pattern-matching HT predictor. A rule length of 4, a table size of 100 entries, and an edit distance threshold of 1 is used as the parameters of the predictor that forecasts chip-interconnect traffic requirements. In this approach, the frequency of the chip interconnect is set to match the chip-interconnect traffic requirements of the upcoming GPU kernel as estimated by our predictor.

Oracle. The oracle has omnipotent information and knows a priori what the optimal DVFS state should be to minimize power consumption with negligible impact to performance. The accuracy of the oracle is theoretical and would be impossible to achieve by an actual predictor. However, we include oracle as a point of comparison to show that our proposed proactive technique approaches theoretical maximum.

Fig. 5.14 shows the power consumed by the chip interconnect when the above approaches are used. The reactive approach can save up to 8.6 W (38.5%) of power on the chip interconnect as in the case of DIG. For the same application, our proactive approach saves 8.4 W while the oracle would save 8.6 W. For the top 10 most power-hungry applications, the reactive, proactive, and the oracle approaches would save 7.5 W, 6.0 W, and 5.9 W, respectively over the baseline. While the reactive approach would save significantly more power than the proactive approach, it does so at the cost of significant performance degradation of 8% for interconnect-bound applications as shown in Fig. 5.15. In contrast, the proactive approach degrades performance by only 1.5% over the baseline. The power saved by the proactive approach is similar to that of the oracle.

Overhead. We intend the predictor and the power manager to be implemented as a software running on a microprocessor while consuming a low amount of power (e.g., less than 0.5 W). The predictor runs on a separate microcontroller implementing power-management algorithms. The decision to run the power manager on a microprocessor is based on current designs by vendors. IBM, for instance, runs their power management on an on-chip controller based on PowerPC 405. Intel's power-package control unit (PCU) manages power consumption and runs on an embedded controller. The power constraints were chosen to be representative of microcontrollers used in the real world [149].

We also intend that the predictions are made between kernel launches on the GPU and the overhead of the predictions to be sufficiently low enough for the kernel launch cost (5 μs [92]) to entirely hide the overhead of phase prediction. To satisfy these conditions, the microprocessor must be able to provide 1680 MOP/s per watt. A memory size of 100 kB would sufficient to implement our predictor and would be comfortably satisfied by today's microcontrollers. With today's 32-bit microcontrollers capable of performing 3600 MOPs/W [110], our proposed approach should consume less than 0.5 W of power. With reduced-precision arithmetic, it is possible to implement our predictor at an even lower power consumption.

5.9 Summary

In this chapter, we identified chip-interconnect power to be a limiting factor in reaching the exascale goal of 20 MW. We proposed operating the chip interconnect on a separate DVFS domain and proactively setting the chip interconnect's P-state to lower the chip interconnect's power. We performed a detailed characterization study of the chip-interconnect traffic and identified several issues in proactively managing the chip interconnect. We proposed modifications to existing phase predictors resulting in a rule-based, space-limited, approximate pattern matching predictor which addresses the issues regarding generality of approach, handling noise in observations, and power and capacity constraints. Finally, we showed that our proactive approach saves up to 6.0 W of power on the chip interconnect while incurring only a 0.5 W overhead for a separate processor to run the predictor. The power saved with our approach is comparable to an oracle while achieving 98.5% of its performance, thereby making it a viable option for chip-interconnect power management.

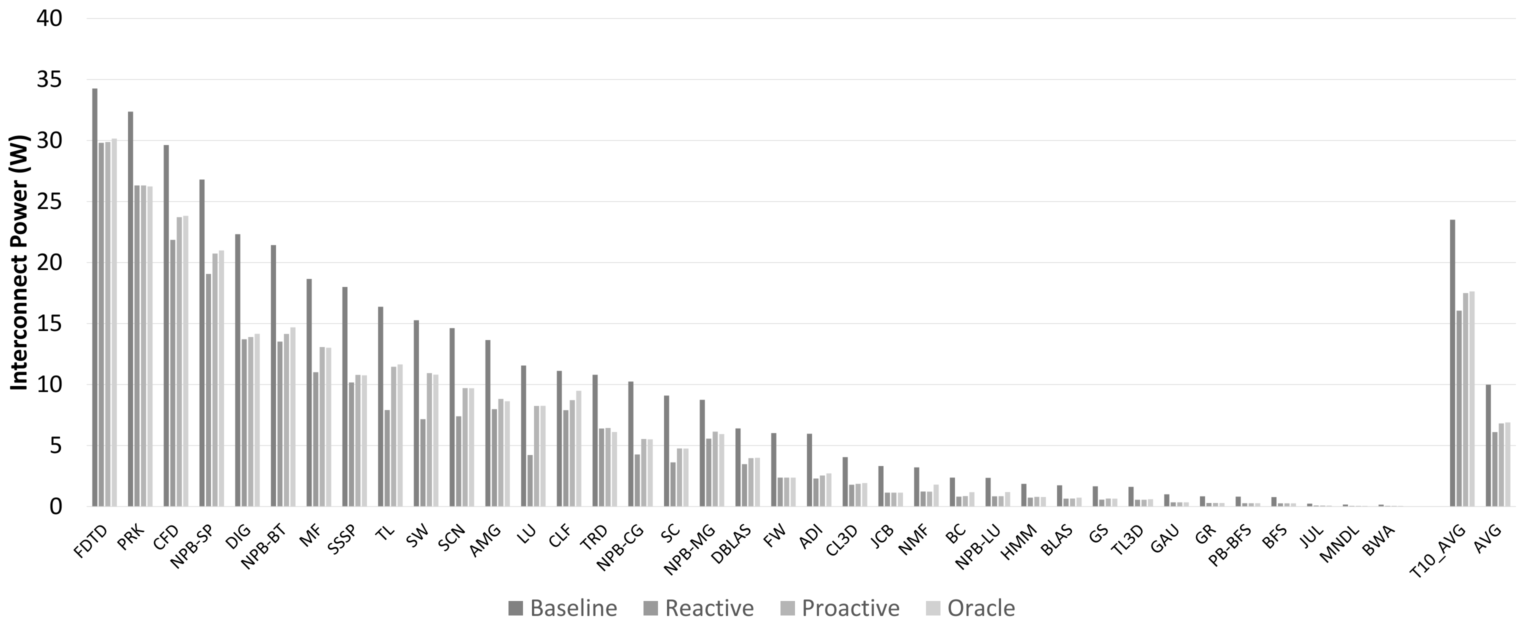


Figure 5.14: Power consumed by the chip interconnect for reactive and proactive power-management approaches compared against the baseline and oracle.

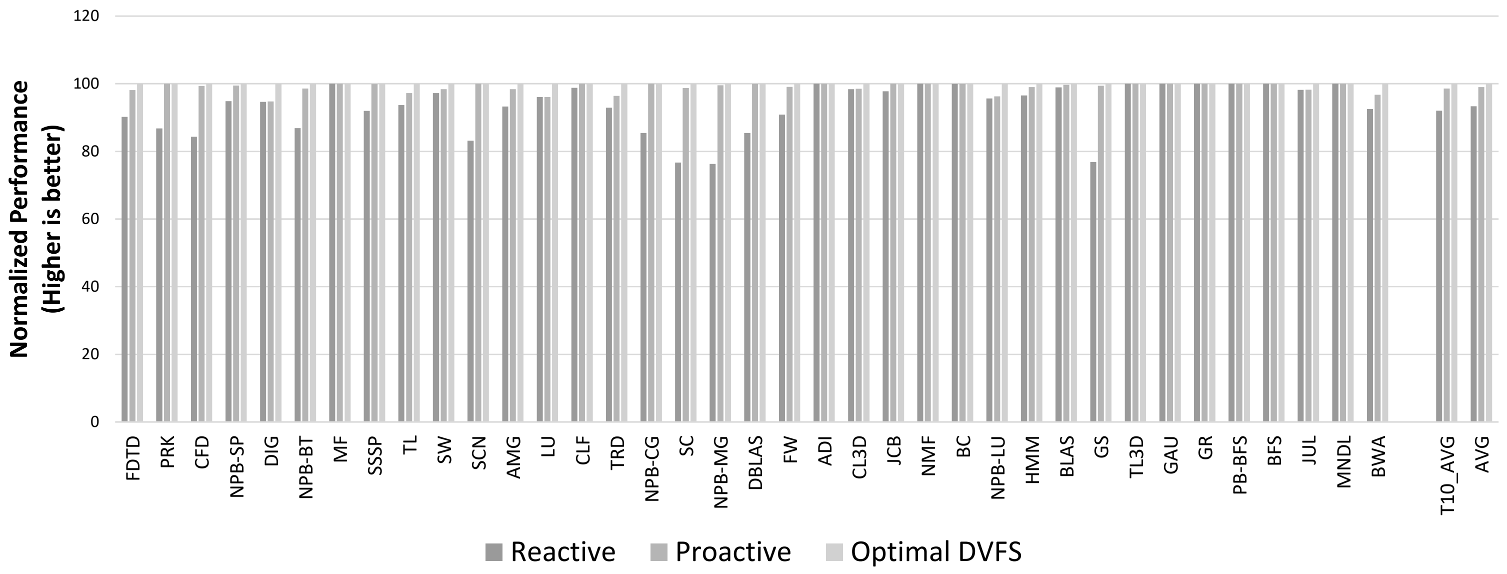


Figure 5.15: Normalized performance for reactive and proactive approaches (Higher is better).

Chapter 6

Hybrid Power-Capping Approach for Heterogeneous Nodes

6.1 Introduction

Realizing the U.S. Department of Energy's original goal of operating an exascale system under 20 MW requires optimally using this power budget at all levels of the supercomputing system—cluster, rack, node, and the chip. At the cluster level, researchers have investigated how to optimally distribute the available power budget among the nodes of the system. Once a node is given a power budget, a power-capping mechanism such as Intel's running average power limit (RAPL) ensures that the node does not exceed its allocated budget. Past work in this area assumes homogeneity within a node [93, 45, 130, 117, 17, 104, 18, 118, 138, 55, 164, 132, 56]. However, supercomputers are increasingly heterogeneous; 145 of the fastest 500 supercomputers use an accelerator or a co-processor (typically, a GPU), as of November 2019. In this chapter, we address the problem of capping the power consumption of a heterogeneous node. We answer the following questions:

- How do we design a power-capping framework for a heterogeneous node that minimizes the time spent over an applied power cap?
- What performance can we obtain out of such a framework?

While there is a rich body of work in power capping a homogeneous node, our work focuses on heterogeneous nodes—specifically, a node with a CPU and a discrete GPU. Compared to related work, the problem we solve is harder due to the difference in capabilities of the two devices. Foremost, CPUs can respond faster to DVFS state change request by a software power manager compared to GPUs. Second, DVFS decisions are based on the characteristics of an application, which are often determined based on performance counter values that can be read with fewer restrictions on the CPU than on the GPU. Considering these differences, we investigate how best to distribute the node's power budget to a heterogeneous node in order to maximize an application's performance without exceeding a power cap.

Our research proposes a new framework to enforce a node-level power cap (P_{cap}) on a heterogeneous node. The framework is composed of three major components as follows:

- A **power manager** to manage CPU's and GPU's DVFS states. The GPU manager uses a model-based *proactive* approach for choosing the optimal GPU configuration. The CPU manager *reacts* by observing the GPU, shrinking its own power usage when the GPU exceeds its budget and reclaiming any unused power to improve its own performance.
- A **power model** that estimates the power drawn by GPU kernels at different configurations. The output of this model serves to inform the power manager on the optimal configuration for a heterogeneous device.
- A **phase predictor** that predicts the characteristics (i.e., performance monitor coun-

ters (PMCs)) of the upcoming GPU kernel. The predicted values of the PMCs serve as input to the models.

Using data collected from real hardware, we show that in our hybrid approach, the measured power was under the allocated power cap 99.6% of the time compared to 96.3% and 92.5% of the time for the reactive and proactive approaches, respectively.

The rest of this chapter is organized as follows. Section 6.2 presents some background information. The scope of this work is presented in Section 6.3 where we explain the types of applications targeted by our solution. A simple power-balancing approach is presented as a baseline and the problems associated with this approach is explained in Section 6.4. The hybrid approach for power balancing is presented in Section 6.5 and details of an idealized implementation is presented in 6.6. Our experimental setup is presented in Section 6.7 and results in Section 6.8. We summarize our findings in Section 6.9.

6.2 Background and Motivation

In this section, we start with a brief description of over-provisioned systems and the role of power capping in such systems. Then we provide an overview of the state-of-the-practice power-capping mechanisms for CPUs and GPUs and explain why these are unlikely to be directly applicable for heterogeneous nodes.

6.2.1 Over-provisioned Systems

In today's high-performance computing (HPC) data centers, electrical capacity is typically over-provisioned. This means, these data centers purchase at least as much electrical capacity as necessary to operate the computational nodes at peak power consumption. However, these nodes rarely consume their peak power when running real-world applications [114, 117]. In response to this observation, researchers have proposed over-provisioned systems [118] where several *extra* nodes are deployed which can create a situation where the system's power consumption exceeds the provisioned electrical capacity. The expectation is that a power manager distributes the available power budget among the nodes in some manner, and these nodes actively readjust their configuration (e.g., DVFS state) or workload to ensure that the system operates under its allocated power budget [56, 55, 136, 137].

6.2.2 CPU Power Capping

Today, the most well-known power capping solution is Intel's RAPL, which caps the power consumption of a CPU's cores and memory individually [45]. The user provides a power cap and RAPL ensures that over some fixed period of time, the average power consumption does not exceed this user-provided power cap. In RAPL's implementation, the power cap is multiplied with the time window to obtain energy credits for the given time window. Periodically, the consumed energy credits (monitored via real measurement or a performance counter-based model) is compared with what is ideal, given the power cap. If the consumed credit is lower, then the component's frequency is increased or vice versa. In effect, Intel's RAPL has a reactive loop to ensure that an allocated power cap is generally not exceeded. While Intel's RAPL mechanism is implemented in hardware, there are similar solutions [164] that are implemented in software. While not as responsive as the hardware implementation, software power-capping solutions for CPUs can achieve the desired power cap in a few milliseconds.

6.2.3 GPU Power Capping

Graphics processing units (GPUs) have similar reactive mechanisms (e.g., power-capping features available via *nvidia-smi* and *rocm-smi*) to ensure that the allocated power cap is not exceeded. We posit that a model-driven proactive approach to set frequencies for the components of the GPU is a better approach than a simple reactive technique to ensure power caps are reached quickly. Hardware implementations work by decreasing the frequency, one step at a time, until the power is under the specified cap. In a hardware implementation, this can be achieved in a few milliseconds. However, the same technique when implemented in software suffers from significant overhead. *Why?* Setting the frequency via software involves making multiple system calls to write to a `/sys` file, the GPU driver reading the requested frequency value and passing it to GPU's hardware DVFS manager, and the hardware setting the frequency. The overhead involved in this sequence of events quickly adds up and results in severe delays between the time a DVFS decision is made and the time when it is enforced. Several such state changes need to take place before the desired power budget is met and by the time this happens, the GPU is often in a different phase consuming a different amount of power. This problem is not as prevalent in CPUs because the software manager can directly write to a model-specific register (`MSR`), which is used by the hardware DVFS manager to set the CPU's frequency. Therefore, a simple reactive approach will not work for software-based GPU power management, and we need to intelligently set the frequency in a proactive manner to keep the overhead low.

6.2.4 Power Capping a Heterogeneous Node

In a heterogeneous node composed of CPUs and discrete GPUs, we need a software power-capping mechanism, as a hardware solution is unlikely to be implemented as the various processors within a node typically come from different vendors. Therefore, we expect a node-level power-capping system to be implemented in software that will run on CPUs. A key challenge here is to address the differences in "reconfiguration" capabilities of the different types of processors within a node. For instance, the CPU has arguably better DVFS support with faster response times compared to the GPU. Also, the support for performance counters in CPUs is generally superior to that of GPUs. This chapter seeks to make use of the relative advantages of CPUs and ensure that the node as a whole stays under its allocated power budget.

6.2.5 Problem Statement

In this chapter, we seek to address the following problem. How do we maximize the performance of an application running on a heterogeneous node while ensuring that the node operates under the applied power cap P_{cap}? Mathematically, the goal of this chapter is to find appropriate DVFS states for CPU ($C_{i,j}$) and GPU ($G_{i,j}$) at *runtime* such that the power consumed at the node level (i.e., $P_{CPU} + P_{GPU}$) is less than the applied power cap (P_{cap}).

6.3 Execution Models for Heterogeneous Applications

In this section, we provide a brief overview of two different execution models for heterogeneous applications that make use of GPUs. We call them the *ping-pong* execution model and *cooperative* execution model. We also explain the unique challenges imposed by these models in the context of power capping a heterogeneous node.

6.3.1 Ping-Pong Execution Model

A vast majority of today's applications follow the *ping-pong* execution model where the processing activity is restricted to one of CPU or GPU at any given point in time. That is, the CPU and GPU do not compute simultaneously. The execution flow of the ping-pong model is shown in Fig. 6.1. Here, the program begins on the CPU where some initial processing is performed. Upon reaching a data-parallel region, the CPU transfers the required data to the GPU and launches a computational kernel on the GPU. When the GPU is processing the transferred data, the CPU is typically idle. The processed data is optionally transferred back to the CPU after which some additional processing may be done by the CPU.

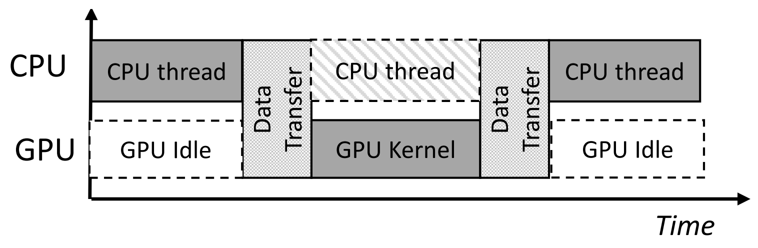


Figure 6.1: Execution flow of a heterogeneous application executing in the ping-pong model

To manage the power consumption of such an application, a power manager will have two decision-making points: (1) any time that the control is with the CPU and (2) immediately before the control is transferred to the GPU. At any given point of control, the power manager must determine the optimal configuration for the CPU or GPU.

6.3.2 Cooperative Execution Model

In the *cooperative* execution model, the processing activity may be shared by the CPU and GPU simultaneously as shown in Fig. 6.2. Here, upon reaching a data-parallel region within the program, the data to be processed is split and a portion is transferred to the GPU. Each processor operates on its allotted portion of the data before combining them on one of the devices (depicted as CPU here) to compute the final output from the partial results.

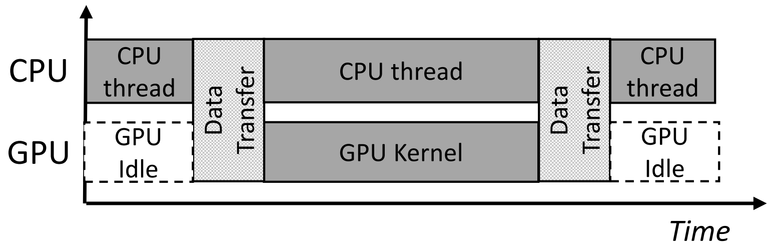


Figure 6.2: Execution flow of a heterogeneous application executing in the cooperative model

In this model, the power manager needs to handle a few additional scenarios, making its decision-making process more complex than that of the ping-pong model. For instance, upon reaching a cooperative region, it has to distribute the available power between the two devices. At the same point in time, it has to find optimal performance configurations for both these devices. Furthermore, when either of CPU or GPU finishes its respective portion of computations first, the power manager has to redistribute the available power budget and recalculate the optimal configuration.

6.4 A Naïve Power-Sloshing Approach for Heterogeneous Nodes

For a typical GPU-accelerated application represented by the execution flow in Fig. 6.1, a simple solution to manage power is shown in Fig. 6.3. In this approach, when the application begins execution, the CPU is set to its highest frequency and the GPU to its lowest. Before the CPU initiates data transfer to the GPU, the GPU is set to its highest frequency. The CPU's frequency remains unchanged as both CPU and GPU participate in the data transfer. After the data transfer is complete and before the GPU kernel begins execution, the CPU is set to its lowest frequency, as during this time, the CPU does not perform any work. After the GPU finishes kernel execution (and before GPU data is transferred to the host), the CPU is clocked at its highest frequency. Finally, after the transfer completes (and when the GPU goes back to being idle), the GPU is clocked to its lowest frequency. When operating under a tighter power budget, instead of setting the CPU and the GPU to the highest frequency, the naïve power-sloshing approach sets their respective frequencies to lower values based on calculations from a DVFS model.

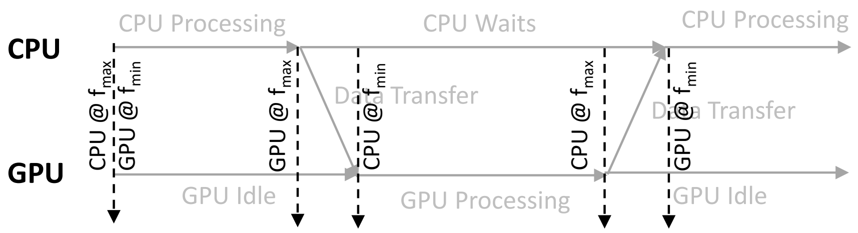


Figure 6.3: A simple power-sloshing solution targeting typical GPU-accelerated programs

Drawbacks of Naïve Power-Sloshing Approach

The expectation is that this simple power-sloshing approach will reduce the power consumption of the heterogeneous node without significantly affecting the performance of the application. When tested with an FFT application from the SHOC benchmark suite, we observed that performance dropped by as much as 2.7-fold. The drawbacks of the simple power-sloshing solution are explained below.

Overhead cost associated with kernels running for a short duration. When the clock frequency for the GPU is set, it is typically through system calls which can take tens of thousands of clock cycles. For low-latency kernels, this overhead becomes too high. Second, once the application requests a desired frequency by writing to the system files, it takes time for the GPU frequency manager to effect the changes. This causes an additional latency to set the requested DVFS state before which the application could move to a different stage (e.g., CPU processing).

Phase behavior in GPU applications. When operating under a tight power cap, the power-sloshing technique determines the optimal frequency for the GPU based on a GPU DVFS model. The inputs for the model are fed by reading performance counter values from the hardware. This technique, however, does not work if the application shows a phase behavior where its characteristics change over time. This is shown in Fig. 6.4 which shows the GPU utilization of a representative application illustrating phase behavior. Trying to optimize the GPU's upcoming state based on its current state could lead to ineffective solutions.

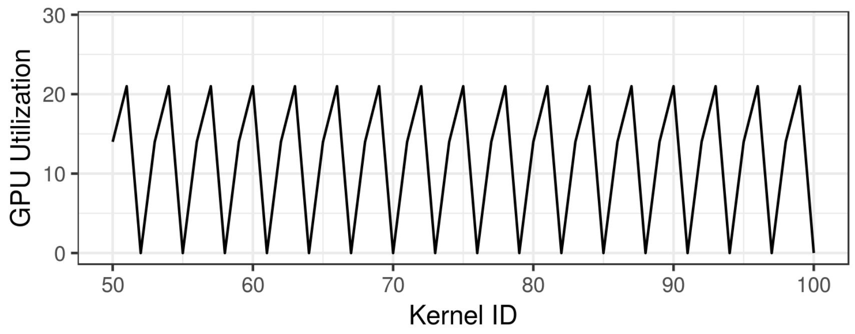


Figure 6.4: GPU utilization of a representative application illustrating phase behavior.

Cooperative Application Use Cases. In many of the GPU-accelerated applications written today, the CPU remains idle while the GPU executes a kernel; this need not always be true. Thus, setting the CPU to its lowest frequency before running a GPU kernel may not the best solution as the CPU may now end up being in the critical path. In this situation, a naïve power-sloshing solution is not preferred.

6.5 NOMAD: A Hybrid Approach for Power Capping

In this section, we present an overview of our proposed hybrid framework, next-generation optimizing manager and daemon (NOMAD[1]), for capping the power consumption of a heterogeneous node, and explain how its various components are designed.

6.5.1 An Overview of NOMAD

Here we present the organization and operation of NOMAD and explain its operation.

Organization of NOMAD

Figure 6.5 shows the organization of NOMAD, which has the following three key components:

- A *power manager* to manage CPU's and GPU's DVFS states. The *CPU manager*, like existing power-capping frameworks, follows a reactive approach whereas the *GPU manager* uses a model-based proactive approach for choosing the optimal configuration.
- A *power model*, which estimates the power of GPU kernels at different configurations. The output of this model serves to inform the *power manager* on the optimal configuration for the processors.
- A *phase predictor*, which predicts the characteristics (i.e., PMCs) of the upcoming GPU kernels. The predicted values of the PMCs serve as input to the *models*.

[1]Nomad also invokes the image of a person who moves from place to place. In our framework, instead of a person being a nomad, power is.

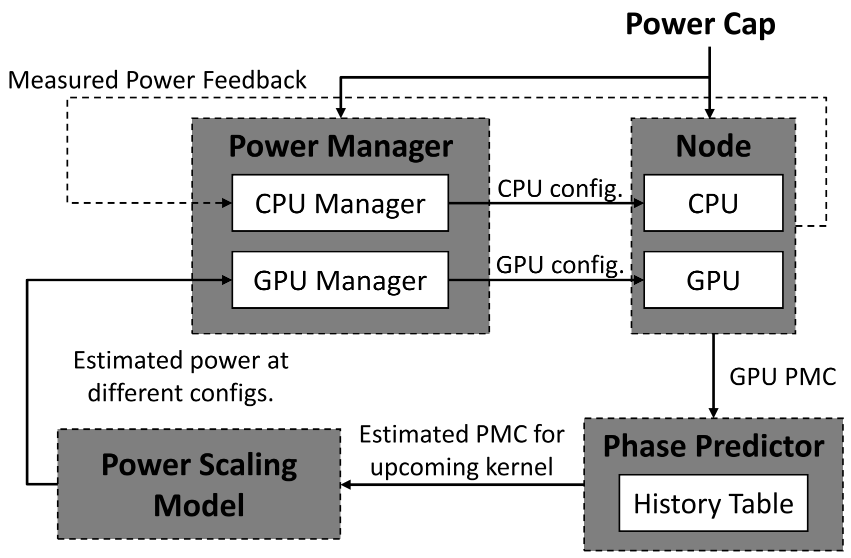


Figure 6.5: Overview of proposed framework

Operation of NOMAD

The remainder of this section explains the operation of NOMAD. Initially, NOMAD performs the following sequence of steps:

1. The cluster-wide power manager allocates individual power caps to individual nodes based on the work allocated to them.

2. Our NOMAD framework receives the node-level power cap (P_{cap}), calculates effective node-level power caps for processing elements ($P_{proc_{cap}}$)[2] after subtracting fixed costs for non-processing elements, and then allocates individual power caps for the CPU and GPU. The individual power caps for the processors are determined as follows:

    - For applications operating in the ping-pong model, initially, the inactive processor gets its idle power as its budget, and the active processors get the remainder of the node-level budget.

[2]In the rest of this chapter, we use to term node-level power cap to more specifically refer to effective node-level power cap for processing elements for ease of exposition

- For applications operating in the cooperative model, initially, the node-level power cap is allocated in proportion to the thermal design power (TDP) of individual processors. However, this allocated budget changes over time as explained next.

3. The *GPU power manager* estimates the power consumption at different frequencies using a pretrained model and sets the frequency of the GPU to the highest possible frequency that results in the GPU operating under its power budget.

4. The *CPU power manager* recalculates its own power budget by observing the GPU's power draw and adjusts the CPU's frequency until the node-level power cap is met.

The *fast*, reactive CPU manager ensures that the problems identified for the cooperative model is solved and ensures that NOMAD operates in a near-optimal fashion in most cases, as explained next. If the GPU ends up consuming more power than allocated due to modeling errors, the CPU frequency is lowered to meet the node-level budget and vice versa. Similarly, in a cooperative application, if the GPU finishes its execution earlier than the CPU, then the CPU's frequency is gradually increased until the power drawn becomes equal to the applied cap. This *fast* response from the CPU thus improves the application's performance, relative to the naïve power-balancing approach in the ping-pong model.

Rationale behind choosing a hybrid approach. A prudent question to ask here is the need for using a reactive approach for the CPU and a proactive approach for a GPU. Here we discuss the drawbacks of a fully reactive and fully proactive approach.

- **Fully Reactive Design.** In a fully-reactive power manager, GPU DVFS state can be adjusted only at kernel boundaries. If a GPU kernel consumes more power than the applied cap, change can be effected only after the kernel finishes execution.

- **Fully Proactive Design.** A proactive design relies heavily on prediction models; but models incur errors. With our frameworks relying on multiple models for PMC value prediction via phase-prediction techniques and DVFS models for power estimation, the errors add up. With no reactive feedback loop to correct the errors, the power manager might end up spending a significant portion of the time over the applied power cap.

6.5.2 Designing a Low-overhead, High-accuracy Power Model

Previous research has shown that it is possible to predict the power consumption of a GPU application at different P-states by collecting performance counter values and applying a statistical or a machine-learning model to predict the power consumption at different frequencies [49, 59, 161]. We adopt a similar approach in this section.

Data Gathering

First, we need to gather sufficient data to cover a wide range of scenarios to train the model. For this, we write a series of microbenchmarks based on the SHOC Level 0 benchmark suite to stress the compute and memory portions of the GPU. The compute microbenchmarks run double-precision *additions*, *multiplications*, and *multiply-adds*. Four different versions of each of these benchmarks are used, each differing in the number of computations per second. The memory microbenchmarks read from the global memory at four different bandwidths. These microbenchmarks are randomly combined to create synthetic applications. This approach gives us the ability to construct arbitrarily many number of training applications with a wide range of characteristics. We then run these synthetic applications at different frequencies and obtain their power consumption and the performance counters listed in Table 6.1. Alternatively, a statistically rigorous approach [4] may also be used to choose applications for training the model. Nevertheless, our microbenchmarking approach gives us precise control over what application characteristics get chosen for model training.

Table 6.1: List of counters considered for power modeling

Name	**Description**
VALUInsts	Avg. vector instructions per second
SALUInsts	Avg. scalar instructions per second
VFetchInsts	Avg. vector fetch instructions per second
SFetchInsts	Avg. vector fetch instructions per second
VWriteInsts	Avg. vector write instructions per second
VALUUtilization	Percentage of total time vector ALU units active
VALUBusy	Percentage of total time time vector instructions are processed
SALUBusy	Percentage of total time time scalar instructions are processed
LDSInsts	Avg. number of LDS read or LDS write
FetchSize	Data size fetched from main memory per second
WriteSize	Data size written to main memory per second
CacheHit	Percentage of total time L2 accesses that result in a hit
MemUnitBusy	Percentage of total time time memory unit is active
MemUnitStalled	Percentage of total time time memory unit is stalled
WriteUnitStalled	Percentage of total time time write unit is stalled

Performance Counter Selection

We run the synthetic applications constructed for training our model at different frequency settings and collect the power and performance counter values at these settings. Then we calculate the Pearson's correlation coefficient between power and performance counters and select those counters that show a high correlation (greater than 0.65 as determined empirically in past work [7]) for modeling the power consumption. For these chosen counters, we calculate their correlation with the power-scaling coefficient, which is the power consumed by the application at its highest frequency divided by the power consumed by the application at its lowest frequency. We select the top two counters—`VALUBusy` and `MemUnitBusy`—for constructing the DVFS power-scaling model. Next, we describe how we choose the model for predicting the power consumed by the kernel at different operating frequencies.

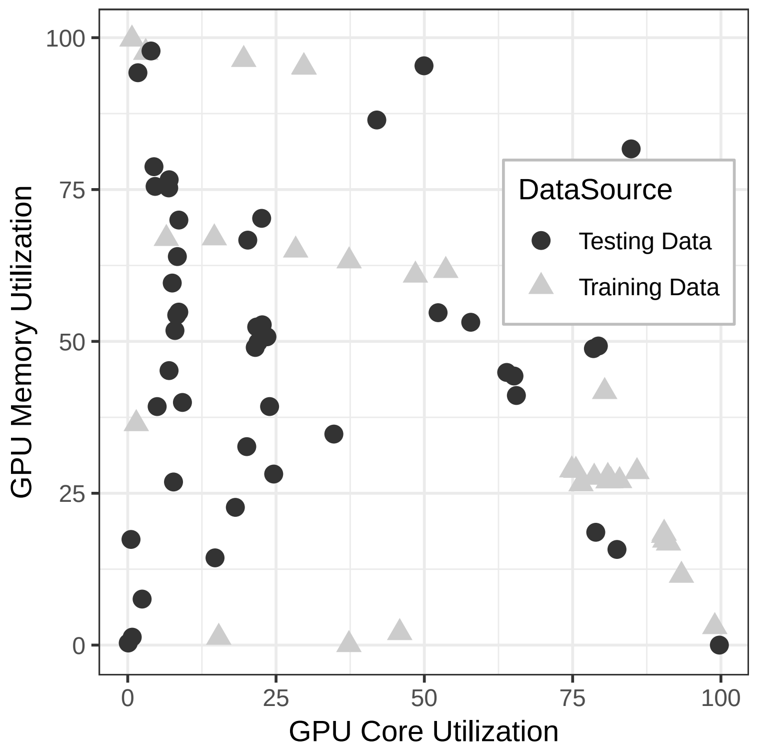


Figure 6.6: Comparison of training and testing benchmarks used for constructing the power and performance scaling models

Model Selection

Past work has indicated that to model the performance and power-scaling curves, a multi-linear regression (MLR), support vector-based model [49], or a neural network [161] may be appropriate. Past work [49] has also documented issues with overfitting a neural network-based model in the presence of a limited amount of data. Therefore, we examine the other two models—MLR and support-vector model—that have been previously used to successfully model DVFS power scaling and test their suitability for our problem. The details of the models explored and their accuracy are described next.

- **Multi-linear Regression (MLR).** In MLR, the output variable (i.e., power drawn) is modeled as a function of several explanatory variables such as core frequency, memory frequency, and various performance monitoring counters indicating the activity factor of various architectural blocks. In this chapter, we model power as a quadratic combination of the input parameters. We explore two different types of model within MLR—one assuming that the input parameters independently affect the power, denoted by MLR, and the other assuming that the interaction of the input parameters have an effect on the power drawn, denoted by MLRI, meaning MLR model with interaction effects modeled. The coefficients of the model parameters are shown in Table 6.2, where CU_F represents the frequency of the compute units, Mem_F represents the frequency of memory, PC1 represents the `VALUBusy` counter, and PC2 represents the `MemUnitBusy` counter.

Table 6.2: Coefficient values for MLR models used in this study

Model	**Intercept**	**CU_F**	**Mem_F**	**PC1**	**PC2**	**CU_F:PC1**	**Mem_F:PC2**
MLR	-204.1	2.8365	0.1542	0.5683	0.1158	–	–
MLRI	-198.9	2.679	0.2830	0.2225	0.3486	3.628E-03	-3.097E-03

- **Support-Vector Regression (SVR).** SVR uses support-vector machines (SVM) to model a regression problem instead of the traditional classification model. In this regression model, we try to fit any arbitrary curve by transforming the input parameters into a higher dimension and allowing a parameterizable error band. When fitting a curve, instead of trying to minimize the sum of the least squares, we try to minimize the data points that fall out of the error band. Various transformation functions may be used to fit a curve. We explore four types of functions, namely, linear, polynomial, radial-basis function (RBF), and sigmoid function. We use `R` to sweep the parameter space and calculate the optimal parameter values for the SVR models.

Fig. 6.7 shows a boxplot comparing the error percentage for the different modeling approaches. The lowest error across applications was observed for SVR with RBF as the kernel with a mean absolute error (MAE) of 9.9%. The highest error was also observed for SVR when a sigmoid function is used to model the relationship between input parameters (i.e., performance counters) and the output (i.e., power). The mean error for this method was 15.5%. This implies that the method used to model the DVFS scaling curve is not as important as the mathematical function forming the basis of the model. In between these two models, in decreasing order of accuracy, are MLRI, MLR, SVR-Linear, and SVR-Polynomial models with an error of 10.6%, 11.2%, 11.3%, and 11.6%, respectively. While a support-vector model showed the highest accuracy, we ended up using the MLRI model for our power-capping framework. This is because a support-vector model has a time complexity of O(d^2), where d is the number of support vectors. With 96 support vectors needed to achieve 9.9% accuracy, the cost of estimating the power consumed by an application at different operating points quickly become unwieldy. On the other hand, the MLRI model is only linear in the number of model parameters, and n is 6 in our case. Therefore, we use the MLRI model in our framework.

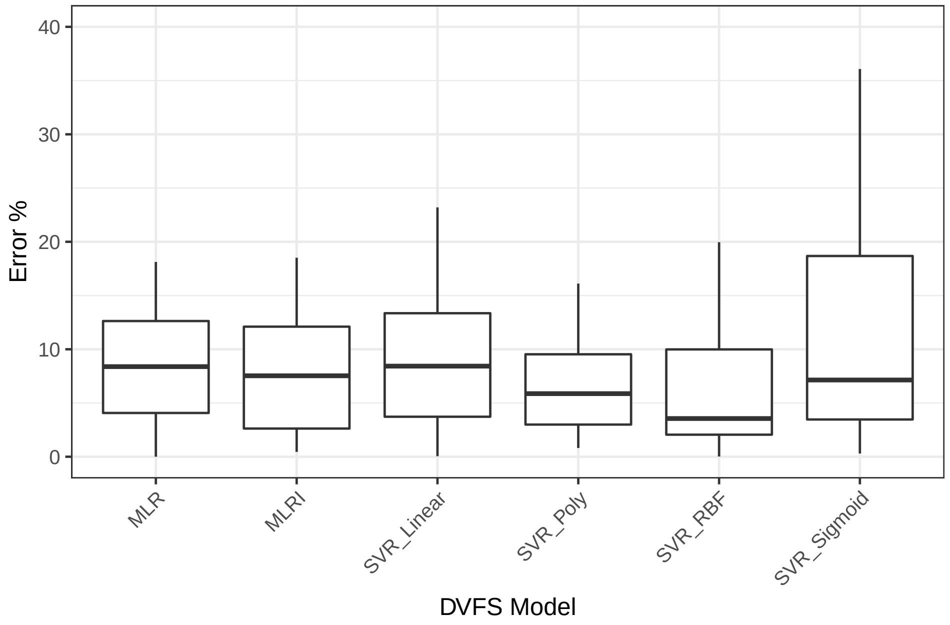
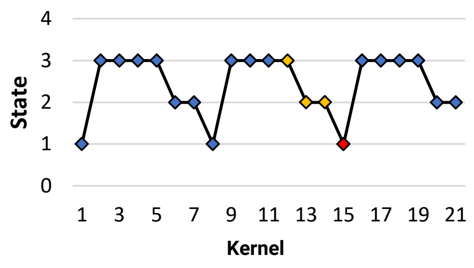


Figure 6.7: Comparison of accuracy for different DVFS power models

6.5.3 Phase-Prediction Model

As shown earlier in Fig. 6.4, the resource utilization of GPU applications exhibits a phase behavior. Therefore, a decision made based on the most recently-seen phase, need not be applicable to the upcoming phase of the GPU application. To combat this, we estimate the GPU utilization and memory utilization of an upcoming phase using a history table that contains data saved historically from previously completed phases. The approach used here is the same as the one introduced in Chapter 5. We briefly summarize the approach again in this section. The history table uses two primary fields for its operation—pattern (also referred to as rule) and outcome. A third field called frequency is used to resolve ties when multiple outcomes are seen for the rules. The history-table predictor operates as follows. It keeps monitoring the kernel's utilization values using performance monitoring counters (PMCs) and associates a discrete level for this utilization. This information is continuously pushed into a FIFO queue-like structure called an *active window*. Once the data in the active window reaches a predetermined length (which is a configurable parameter called the pattern length), it compares the active window to the pattern entries in the history table. If a match is found, the corresponding outcome entry is determined to be the expected utilization level for the upcoming kernel.



(a) Interconnect Traffic

Active Window | 3 | 2 | 2 | 1 |

Pattern	Outcome	Frequency
1 3 3 3	3	2
3 3 3 3	2	2
3 3 3 2	2	2
3 3 2 2	1	2
3 2 2 1	3	1

(b) History Table

Figure 6.8: Illustration of history table-based phase prediction

In the example shown in Fig. 6.8, a pattern length of 4 is used, which means we look at the previous four kernels' statistics to determine the utilization values of the upcoming kernels. At the end of the fourth kernel, the active window would be *1-3-3-3* and at the end of the fifth kernel, the first entry in the table is populated as pattern = *1-3-3-3* and outcome = *3*. At the end of the 11th kernel, the active window would again be *1-3-3-3*. This is now compared against the pattern entries in the history table, and a match is found. The utilization level for the next kernel is then estimated to be the corresponding outcome *3*, and the GPU core and memory frequencies are set appropriately. If there is no match to be found, we overprovision the GPU to the highest DVFS state possible to avoid adversely affecting the performance.

For our implementation, we tune the rule length of the history table-based predictor. Our experiments show that we achieve the maximum accuracy when a rule length of two is used. The prediction error for memory utilization (MemUnitBusy) and core utilization (VALUBusy) are presented in Fig. 6.9 and Fig. 6.10, respectively. On an average, this technique resulted in a phase-prediction error of less than 2% for both these performance counters. In general, for both these metrics, FFT and GEMM showed the least accuracy with their respective errors topping 6% in both cases.

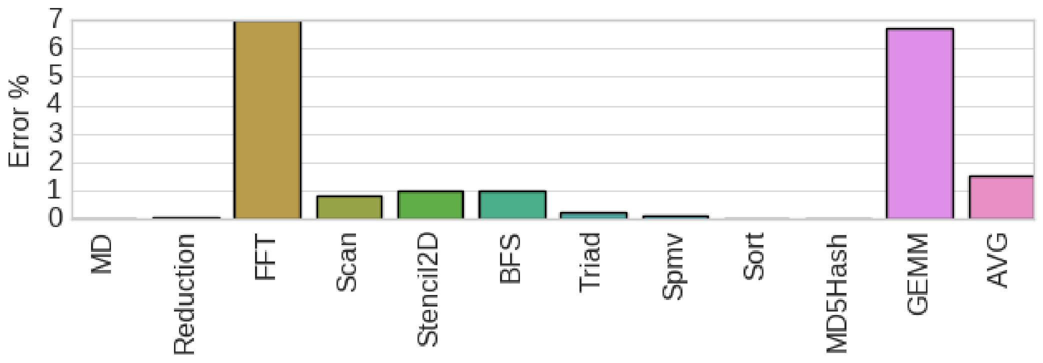


Figure 6.9: Prediction error for predicting memory phases using history table-based predictor

To examine the reasons for the lower accuracy for these applications, we analyzed their

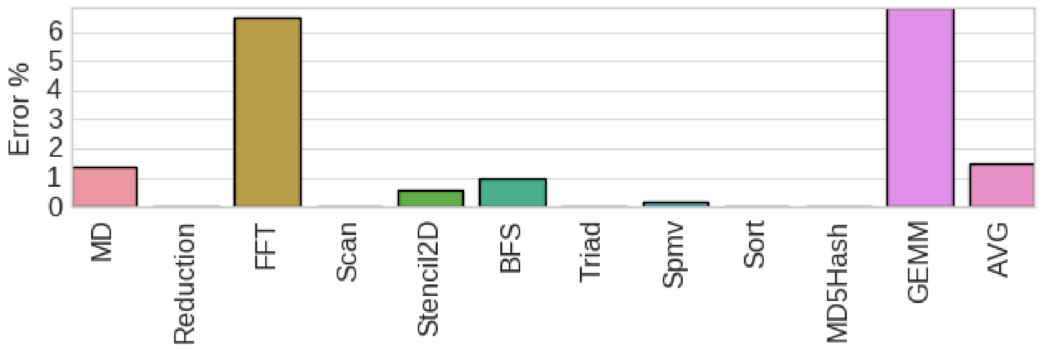


Figure 6.10: Prediction error for predicting core phases using history table-based predictor

respective traces. The memory-utilization trace showing the predicted and actual memory phases for GEMM is presented in Fig. 6.11. This application has distinct phase changes over time, which are being reliably predicted by the history table predictor. However, in the initial phase of the application's execution, the properties of the application change which is not picked up by the predictor early enough and it makes a few errors in prediction. While it is a long-running application, there are only few kernels in GEMM. Therefore, the errors made by the predictor in the initial phase has a larger impact on the overall accuracy. For both FFT and GEMM, this is the case, and if the application were to be run for longer than 60 seconds, the error would have been amortized over time. To show the effectiveness of the phase predictor, we also show the predicted and actual memory phases for breadth-first search (BFS), an application with distinct phase-change behavior that is correctly being picked up by our phase predictor.

6.6 Prototype Implementation

In our prototype implementation of NOMAD, we have a calibration phase and an optimization phase, which are described next.

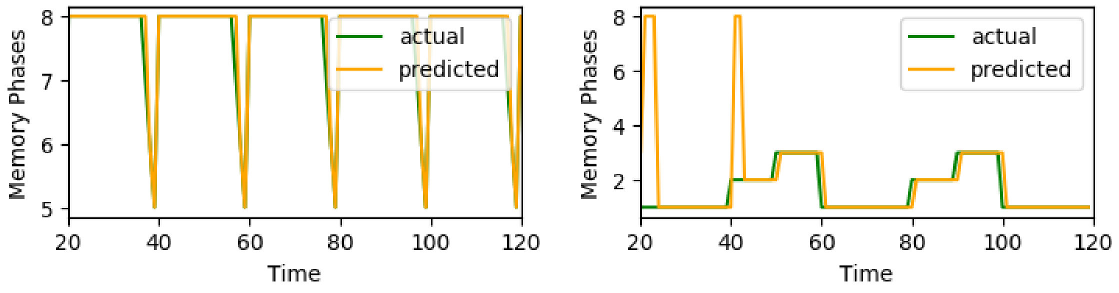


Figure 6.11: Examples of predicted phase and actual phase for BFS and GEMM

6.6.1 Calibration Phase

In our prototype, the first ten seconds of an application constitute the calibration phase of the application. The calibration serves two purposes. First, the phase predictor needs some history to start making reliable predictions. The framework uses this time to populate the history table. Second, to reduce modeling errors, the power estimated by our model is compared against the measured value from power sensors in the calibration phase. Then, we compute a scaling factor from the measured and estimated values and multiply our model with the scaling factor. This reduces the overall error incurred by the model. (Note that the time taken by the calibration stage is not included in the evaluation.)

6.6.2 Optimization Phase

In this phase of the application, we apply the power cap. In the evaluation sections that follow, we collect statistics during the optimization phase.

Monitor. The monitor thread is responsible for the collection of power data from the GPU and CPU power sensors. It also collects and gathers data from the GPU performance counters. In our implementation, we pre-profiled the performance counter data and read the data from memory. For short-running kernels, we batch several kernels together so that their collective runtime is at least 200 ms and aggregate their performance counters before the optimizer calculates the optimal configuration for the upcoming time period. In an ideal framework, this data would be collected by wrapping GPU function calls with a runtime profiler.

Optimizer. The optimizer is responsible for calculating the optimal P-state. For this, the optimizer needs to calculate the expected power consumption for the various P-states. We used an approach similar to Zhang et al. [164] to reduce the amount of overhead involved in estimating the optimal state. We traverse the configurations in a predefined order. The order is determined offline by comparing the performance of two reference applications at different P-states. Therefore, the first configuration that comes under the allocated power budget is determined to be the best configuration for the upcoming kernel.

Actuator. The actuator is responsible for applying the decisions made by the optimizer. It does so by writing the selected values to the `/sys` files corresponding to the compute and memory frequencies of the GPU. The actuator also writes to the P-state control model-specific register (MSR) for selecting CPU core frequencies.

6.7 Experimental Setup

In this section, we describe the platform and benchmarks used in this study. We also describe two other power-capping mechanisms against which we compare our approach.

6.7.1 Platform Details

We perform our experiments on a heterogeneous node equipped with a quad-core AMD Ryzen 5 1400 CPU with 8GB of DDR4 RAM and AMD Radeon RX 580 GPU with 4GB of GDDR5 memory. The machine runs Ubuntu 16.04 with version 2.0 of the Radeon Open Compute (ROCm) runtime. The CPU supports three P-states and can be run at 2200, 2600, and 3200 MHz. The CPU memory frequency is fixed at 2667 MHz. The GPU core frequency can vary between 300-1380 MHz supporting eight different frequencies. The GPU memory frequency can be set to 300, 1000, or 2000 MHz. The CPU's TDP is 65 W, and the GPU's TDP is 185 W. In our experiments, we disable frequency boosting.

6.7.2 Applications

In this section, we explain the ping-pong and cooperative applications used in this study.

Ping-Pong Applications

For our evaluation, we used the following applications. These applications are based on SHOC Level 1 benchmark suite and modified by increasing the input data size to 2048 MB and increasing the number of passes so that each application runs for at least 60 seconds.

FFT. This benchmark repeatedly performs a two-dimensional fast Fourier transform in the forward and backward directions on the GPU and checks their results on a CPU.

GEMM. This benchmark repeatedly performs double-precision matrix multiplication using the GEMM BLAS routines.

Stencil2D. This benchmark performs nine-point stencil computations on a large dense matrix repeatedly.

SpMV. This benchmark multiplies a large sparse matrix with a dense vector using CSR Scalar, CSR Vector and ELLPACKR formats repeatedly.

Scan. This benchmark computes the parallel prefix sum of double-precision floating-point data.

Sort. This benchmark sorts a large array of unsigned integers using the radix-sort algorithm.

Reduction. The sum of the elements of a large double-precision floating-point array is computed.

BFS. This benchmark searches for nodes in an undirected graph using the breadth-first search technique.

Cooperative Applications

The following applications are evaluated under this model:

Reduction. The sum of elements of a 2048 MB double-precision floating-point array is computed repeatedly (same as in the case of the ping-pong model). 256 MB of the data is computed by the CPU and the remaining by the GPU with the final merging of the results taking place in the CPU.

GEMM. A double-precision matrix-multiplication is performed with the CPU computing 10% of the elements of the matrix and the GPU computing the remaining.

Stencil2D. Nine-point stencil computations are performed on a 2048 MB matrix holding double-precision values. The CPU calculates 10% of the values, and the GPU calculates the rest 90% of the values.

6.7.3 Points of Comparison

The proposed hybrid approach is compared against two other approaches:

- **Reactive Approach.** In the reactive approach, after determining the respective power budget for the CPU and GPU, their respective frequencies are adjusted based on the most recent power measurements from the CPU and the GPU. If, for instance, the GPU consumed less power than what was allocated, its frequency is increased by one level for the subsequent time period. Similarly, when it exceeds its allocated power budget, the GPU's frequency is decreased by one level for the next time period. This process is continued until the respective device's power budget are met.

- **Proactive Approach.** In a proactive approach, after determining the power budget for the CPU and GPU, the core and memory utilization values are estimated for the upcoming time period, and based on this predicted value and a power model, their respective frequencies are determined. If there is an error in the estimate, the frequencies of the processors do not re-adjust themselves until the next prediction interval. The hybrid approach, in contrast, will allow the CPU to change its own state even during the current time period.

6.8 Evaluation

In this section, we evaluate our proposed hybrid approach and compare it against the reactive and proactive approaches. First, we present the metrics used for comparison, followed by the results for ping-pong applications and cooperative applications.

6.8.1 Metrics of Evaluation

We use two metrics, power-capping effectiveness and performance, to evaluate our proposed approach and compare it against competing approaches.

Power-capping Effectiveness. *Effectiveness* is the percentage of execution time spent under the allocated power cap for a given application.

$$\text{Effectiveness} = \frac{n - |P_i <= P_{cap}|}{n} \tag{6.1}$$

where, n is the total number of power readings, P_i are the individual power readings, and P_{cap} is the applied power cap. Higher values for effectiveness is the desired outcome.

Performance

The performance of the power-capping approaches is measured as the slowdown incurred when a power cap is applied (compared to when the application is run without applying a power cap). This metric is calculated as follows.

$$\text{Slowdown} = \frac{\text{Execution Time with power cap applied}}{\text{Execution time without applying power cap}} \tag{6.2}$$

This metric tells the percentage increase in net execution time as a result of applying a power cap and a lower value indicates a better performance of the power-capping mechanism.

6.8.2 Ping-Pong Applications

This section presents the results for the ping-pong applications listed in Section 6.7. Foremost, we need a power-capping mechanism to ensure that the system operates under the power cap most of the time. Therefore, we present the power-capping effectiveness of the reactive, proactive, and hybrid approaches at two different power caps in Fig. 6.12 and Fig. 6.13, respectively.

Power-Capping Effectiveness. Fig. 6.12 shows the power-capping effectiveness of the three approaches for different applications when operating at a 140 W power cap. Overall, our proposed hybrid approach operates under the power cap over 98.9% of the time. In comparison, the reactive and proactive approaches operate under the power cap only 94.2% and 97.9%, respectively. Similarly, in Fig. 6.13, we can observe that the hybrid power-capping mechanism was more effective at keeping the system under a power budget of 130 W. In the hybrid approach, the measured power was under the allocated power cap 99.6% of the time compared to 96.3% and 92.5% for reactive and proactive approaches, respectively. This result also highlights the importance of the feedback-driven CPU manager. When the proactive manager, due to modeling errors, makes sub-optimal decisions as in the case of `GEMM` and `Stencil` shown in Fig. 6.13, the CPU manager is able to re-adjust itself to improve effectiveness from 82.5% to 99.4% for `GEMM` and 76.6% to 98.9% for `Stencil`.

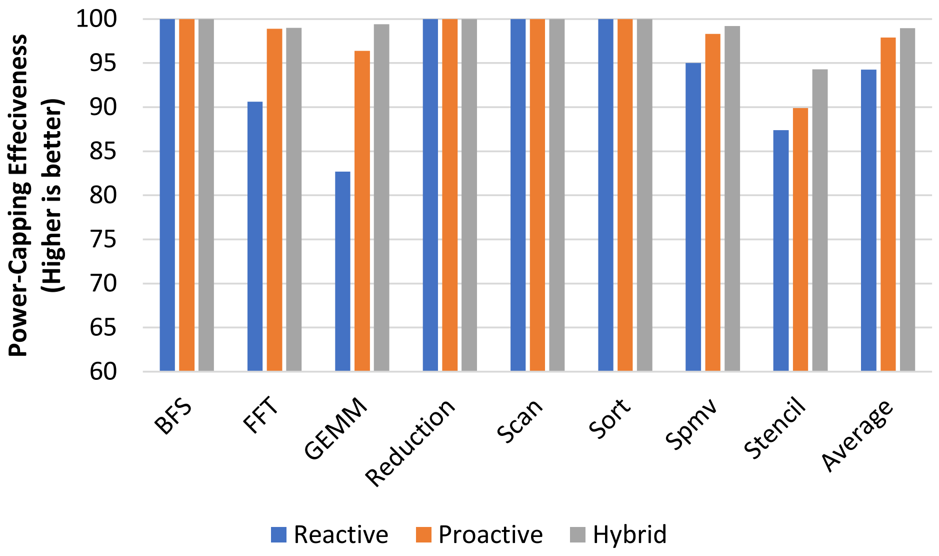


Figure 6.12: Effectiveness of different power-capping approaches when applications run under a 140W power cap

Performance. Next, we present the performance results for applications running under a power cap using our power-capping approach and compare it against other approaches. While performance is only of secondary interest in a power-capping system, we are interested in learning how much performance one could obtain in a power-constrained environment.

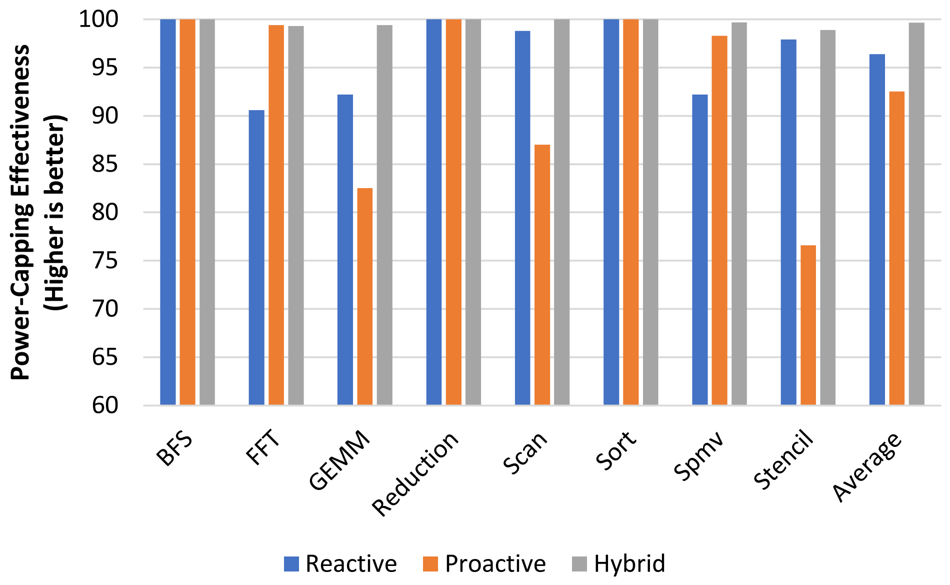


Figure 6.13: Effectiveness of different power-capping approaches when applications run under a 130W power cap

Fig. 6.14 shows the normalized performance for applications running under a 140 W power cap. Our hybrid approach incurs a slowdown of nearly 10% compared to a baseline with *no* power cap. The slowdown itself is to be expected since in the baseline we are not constrained by power. Compared to the reactive and proactive approaches, whose slowdown are 7.7% and 11.1%, this slowdown is not significant, especially considering the fact that the other approaches operate over the power cap more often than our proposed hybrid approach. Under a tighter power cap, we in fact, observe a better performance for our hybrid approach compared to the other two approaches, making a stronger case for our proposed hybrid approach.

6.8.3 Cooperative Applications

In this section, we present results showing the power-capping effectiveness and performance for cooperative applications in Fig. 6.16 and Fig. 6.17 respectively, when operating under

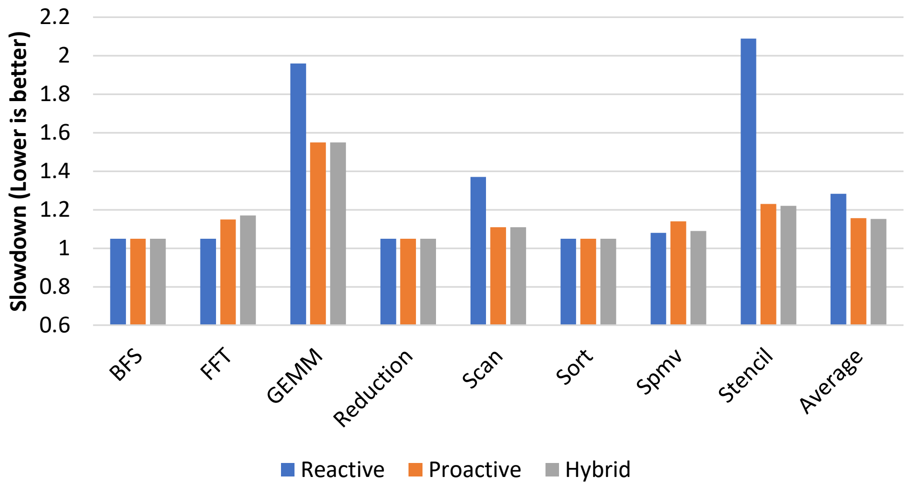


Figure 6.14: Performance results for ping-pong applications under a 140 W power cap

a 150 W power cap. We evaluate a higher power cap for cooperative applications because since both CPU and GPU are being used simultaneously, it is natural to expect that the cluster-wide power manager allocates a higher power cap for nodes running cooperative applications.

The hybrid approach is seen to be more effective than the other two approaches with the applications operating *under* the power cap, 98.6% of the time. In comparison, the reactive approach operated under the power cap only 97.7% of the time and the proactive approach only 84.5% of the time. The hybrid approach shows better performance, especially for the GEMM application where the CPU threads run longer than the GPU application; and when the GPU first finishes execution, the CPU reclaims some of the unused power. Overall, this reduces the slowdown due to power capping from 26% for proactive capping to 16% for hybrid capping for GEMM and from 13.3% to 7.8% overall.

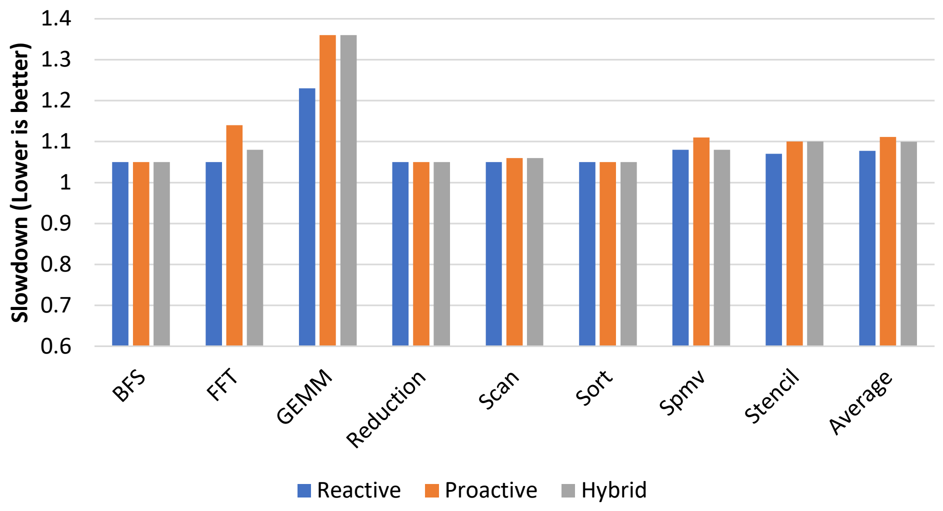


Figure 6.15: Performance results for ping-pong applications under a 130 W power cap

6.9 Summary

In this chapter, we introduced a hybrid power-capping framework, known as NOMAD, for capping the power consumption of a heterogeneous node. The framework proactively manages the GPU's P-state to optimize for performance under a power budget. The CPU reacts to the proactive power manager's decisions by shrinking its power usage when the GPU exceeds its budget or reclaims any unused power budget for itself to improve its own performance. The various components required for this framework, namely power models and phase predictors, are designed and optimized. Experiments with a heterogeneous system composed of a CPU and GPU showed that the hybrid power-capping framework results in the system consuming power that is less than the applied power cap over 98% of the time.

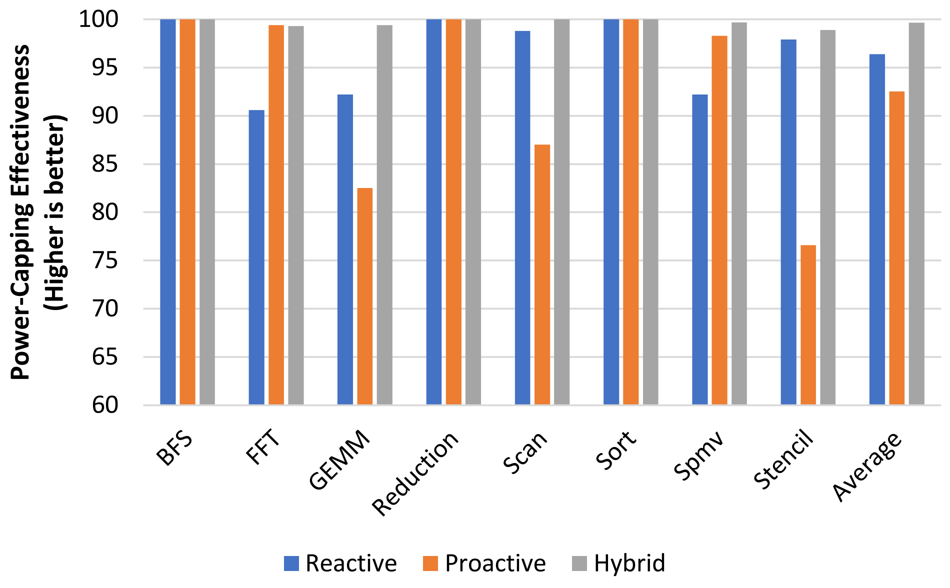


Figure 6.16: Power-capping effectiveness results for co-operative applications under a power cap

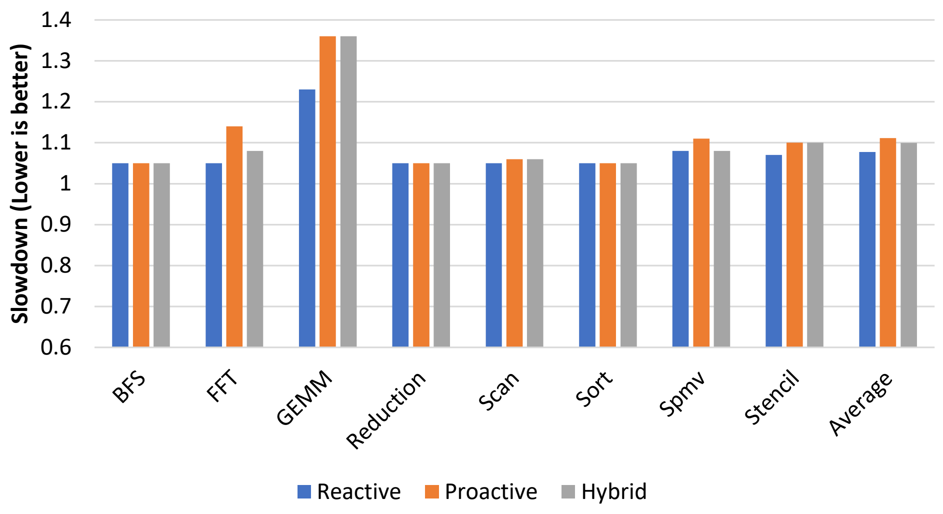


Figure 6.17: Performance results for co-operative applications under a power cap

Chapter 7

Reducing Cluster Energy via In-situ Visualization

7.1 Introduction

This chapter deals with addressing greenness-related issues for an important class of applications: scientific data visualization. Let's consider the example of a traditional visualization shown in Figure 7.1. In this type of visualization, a simulation is performed; and at the end of each iteration of the simulation, raw data is written to the storage system. This storage system is typically a parallel file system in high-performance computing (HPC) environments. After the simulation is complete, the data is transferred to a rendering cluster where an image is rendered for each iteration. This type of visualization results in large amounts of data transfers to the disk – even upwards of hundreds of petabytes on high-end machines. Clearly the bottleneck resource for these applications is in the storage subsystem. This means, to improve the performance of such applications, it makes sense to move power away from the compute subsystem to the storage subsystem. However, due to various reasons such as funding models in HPC datacenters, it is unlikely that the storage subsystem gets more than 10% of the monetary and power budget.

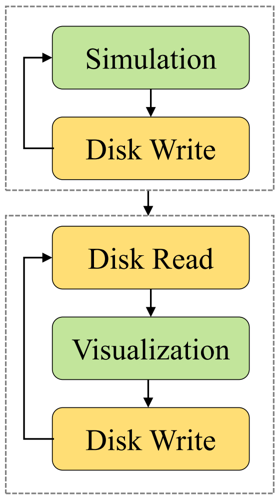


Figure 7.1: Traditional post-processing visualization

In this chapter, we look at techniques to invert the bottleneck to compute resources and to move power away from the storage subsystem to the compute subsystem to improve the performance of scientific visualization applications. The broad class of techniques that we adopt here to invert the bottleneck is insitu technique.

Our experimental results show that an *in-situ* pipeline runs 51% faster, consumes 50% less energy, and occupies 99.5% less disk space than a post-processing pipeline for an ocean simulation application. This means nearly 10% of the total power which is spent towards the storage subsystem can be re-routed to the compute units, which our experimental results show can improve the performance by as much as 6.4%.

The rest of the chapter is organized as follows. We provide background on in-situ techniques in Section 7.2. We present a characterization of the two approaches, i.e., post-processing and in-situ, towards scientific visualization in Section 7.3. Based on the characterization, we build an empirical model in Section 7.4. Use cases for the model are presented in Section 7.5.

Finally, we draw conclusions from our experiments in Section 7.6.

7.2 In-situ Techniques

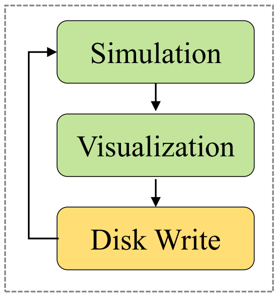


Figure 7.2: In-situ visualization

While in-situ techniques have several practical advantages, they can also help reduce the storage requirements in an HPC data center. As can be seen from Figure 7.2, the simulation and visualization tasks are performed on the same machine *concurrently* in the case of an in-situ pipeline. Instead of saving the raw data at the end of each iteration, an image corresponding to the data is rendered and saved to disk. The image thus produced is typically orders of magnitude smaller than the raw data used for post-processing visualization (Figure 7.1).

Similarly, we can filter a subset of the data generated to be later visualized, which can further reduce the size of the data that gets saved to the disk. This filtering can occur in the time domain where the data is saved only every few time steps or in the space domain where a subset of data points are rendered instead of rendering all the data points. These techniques are known as temporal sampling and spatial sampling, respectively.

7.3 Characterization of In-situ Techniques

To get an idea of the amount of power that can be diverted away from the storage subsystem to the compute subsystem, we present a characterization of in-situ techniques using a climate simulation application as an example. The details are described next.

7.3.1 Experimental Setup

We describe the HPC test system, our power-monitoring setup, and the scientific application used as a driver for our study.

HPC system

For our characterization experiments, we used *Caddy*, a 150-node/2400-core cluster located at Los Alamos National Laboratory. Each node contains two sockets of 8-core Intel E5-2670 Sandy Bridge CPUs running at 2.6 GHz as well as 64 GB of DRAM. Nodes are interconnected using a QLogic InfiniBand QDR network.

A storage cluster running a Lustre file system is attached to this supercomputer. The storage cluster consists of five nodes, which are configured as follows: one node serves as the master node, two nodes are used for metadata servers (MDS), and two nodes are used as object storage servers (OSS). The storage cluster provides 7.7 TB of storage and over 160 MB/s of bandwidth for random reads and writes. Because this storage cluster is private to *Caddy*, interference from other clusters is eliminated. We also ran our test application on the entire cluster to ensure that we only measured the power consumed by our application.

Power monitoring

Here, we describe the setup used for monitoring the power consumption of the compute components and the storage components for the hardware setup used in this study.

The storage cluster was mounted on a Raritan intelligent rack, which is equipped with metered PDUs that are capable of measuring the power consumption at the power inlet. The frequency of data collection was set to one measurement per minute, which is the maximum possible for this type of power meter. Within this one-minute interval, multiple measurements are made and an average power value for that interval is reported. From this average power profile, we calculate other derived metrics such as energy.

We used the Appro power monitoring interface [15] to collect the power profile at the *cage level*, where a *cage* is a group of ten nodes. A group of three cages makes a rack. Like the rack-level power meter used for the storage cluster, the cage-level power meter is capable of providing an average power estimation every minute. We collected data from 15 such cage-level power meters, covering all 150 nodes across the five racks that make the cluster.

7.3.2 Driving Application

For our investigation, we use a climate-simulation application known as Modeling for Prediction Across Scales (MPAS) [128]. More specifically, we use the ocean component of MPAS (MPAS-O) to compare the different types of visualization pipelines (namely, *in-situ* and post-processing) with different system configurations. The visualization task here is to identify and track *eddies*, which are rotating bodies of fluid surrounded by shearing fluid [160].

To accomplish the tasks, we perform the following steps. First, we solve an unstructured grid problem in order to simulate the ocean. From the raw dataset produced by the simulation, we derive a metric known as Okubo-Weiss, which is used to identify eddies. For the post-processing pipeline, the Okubo-Weiss metric is extracted at the end of each timestep of the simulation and written as a netCDF file. For I/O, we use the PIO library, which in turn uses parallel netCDF so that the output can be written to the parallel file system faster. After the simulation is complete, the netCDF files from each timestep are read back and visualized in parallel using the ParaView framework. We show an example image from the simulation depicting the eddies using the Okubo-Weiss metric in Figure 7.3.

For the *in-situ* pipeline, the extracted Okubo-Weiss metric is not written directly as netCDF. Instead, it is passed to the *Paraview Cinema* framework [8], where it is visualized and then written to the disk. To accomplish this, we used Catalyst adaptors [19] that seamlessly copy simulation data structures to Paraview data structures. While this incurs additional memory operations, it also avoids the large (and much slower) data transfers to the storage system.

For all the direct measurements reported in this chapter, we use the following problem sizes:

- Grid size of 60 km x 60 km for the ocean
- Simulation period of six months
- Time step of half an hour

We evaluate the *in-situ* pipeline and the post-processing pipeline in three different configurations: the output products are written once in every (i) 8 simulated hours, (ii) 24 simulated hours, and (iii) 72 simulated hours. The output product can be either netCDF files or images depending on the type of pipeline being run.

7.3.3 Characterization Results

In this section, we present the characterization results for the various pipelines studied.

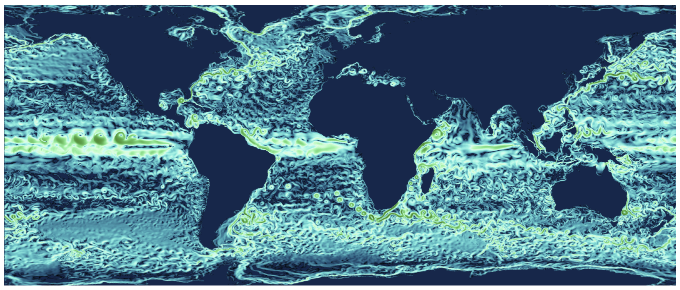


Figure 7.3: An example visualization from a climate simulation application

Figure 7.4 compares the storage requirements of the two kinds of pipelines. As Figure 7.4 indicates, *in-situ* techniques reduces the required storage from 230 GB to less than 1 GB when the temporal sampling rate was set to record output once every 8 hours. For the other two configurations, the storage requirements decreased from 80 GB and 27 GB, respectively, to negligible amounts. In all these cases, we observed a data size reduction of over 99.5%. As shown in Figure 7.5, the techniques explored have nearly zero overhead in terms of power.

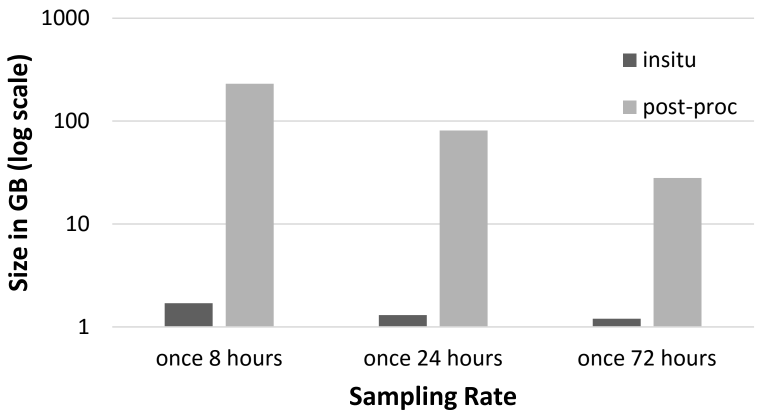


Figure 7.4: Comparison of storage requirements for *in-situ* and post-processing pipelines at three different sampling rates

These results indicate that a majority of the power budget allocated to the storage components can be moved to the compute components.

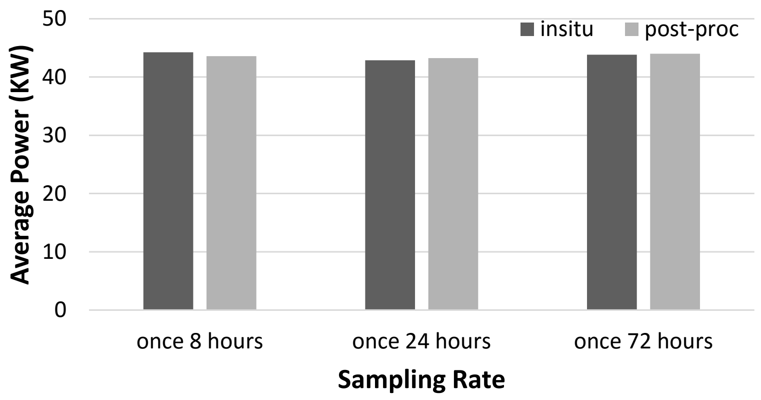


Figure 7.5: Comparison of power consumption of *in-situ* and post-processing pipelines at three different temporal sampling rates

7.4 An Empirical Modeling Approach

We model the performance, energy, and storage of the visualization pipelines in order to estimate those metrics at different sampling rates and application configurations. This will help in evaluating a number of scenarios and answering several what-if questions (Note that while the model derived here is architecture-specific and application-aware, the methodology itself is generic and can be applied to other computing systems and applications). The symbols used in the model are summarized in Table 7.1.

The energy consumed by a visualization pipeline can be expressed as the product of its average power and total execution time.

$$E = P \cdot t \tag{7.1}$$

As observed in Figure 7.5, the average power across all sampling rates can be considered constant. We need to only model the execution time of the application, which can be expressed as the summation of the time taken for the simulation (t_{sim}), I/O ($t_{i/o}$), and

Table 7.1: Summary of symbols used in our model

E	Total energy consumed by the visualization pipeline
P	Average power consumption for the visualization pipeline
t	Total execution time for a visualization pipeline
t_{sim}	Time taken by the simulation phase
$t_{i/o}$	Time taken by the I/O phase
t_{viz}	Time taken by the visualization phase
$S_{i/o}$	Size of output (in GB) produced by the simulation
N_{viz}	Number of images produced by the simulation
α	Time cost to write 1 GB of raw data; value estimated by linear solver
β	Time cost to generate one image corresponding to one timestep; value estimated by linear solver
$iter_{ref}$	Number of iterations or timesteps in the reference configuration
$iter_{any}$	Number of iterations or timesteps performed
$rate_{ref}$	Output sampling rate used in the reference configuration
$rate_{any}$	Output sampling rate for which performance metrics must be estimated
$t_{sim.ref}$	Total execution time of the reference configuration
$S_{i/o.ref}$	Size of output produced for the reference configuration
$N_{viz.ref}$	Number of images produced for the reference configuration
$S_{i/o.any}$	Estimated size of output produced by any given configuration
$N_{viz.any}$	Estimated number of images produced by any given configuration

visualization phases (t_{viz}):

$$t = t_{sim} + t_{i/o} + t_{viz} \tag{7.2}$$

Here, the time taken for the simulation phase, t_{sim}, is a constant for a given number of time steps or iterations of the simulation. The times taken for I/O and visualization phases are dependent on the amount of data written and the number of images visualized, which in turn are dependent on the sampling rate. Mathematically, we can express the time taken for an application as

$$t = t_{sim} + \alpha S_{i/o} + \beta N_{viz} \tag{7.3}$$

in which α and β denote, respectively, the time taken to write 1 GB of output and to produce one set of images corresponding to one timestep. $S_{i/o}$ and N_{viz} are, respectively, the total output size and number of images. These values can be estimated easily for any sampling rate given a reference point as they are linearly dependent on the sampling rate.

We can express Equation (7.3) in a more generic form as

$$t = \frac{iter_{any}}{iter_{ref}} \times t_{sim.ref} + \alpha S_{i/o} + \beta N_{viz} \tag{7.4}$$

where, $iter_{any}$ and $iter_{ref}$ are the number of iterations performed in the current configuration and reference configuration, respectively. That is, the simulation time will scale with the number of iterations or timesteps in the simulation.

We use a linear solver to estimate the values of α and β. The data collected from three different configuration points, namely, (i) *in-situ*, once every 8 hours, (ii) *in-situ*, once every 72 hours, and (iii) *post-processing*, once every 24 hours, was used to solve for α and β. Alternatively, regression techniques may be used to solve these equations using the following system of equations:

$$\begin{aligned} t_{sim} + 0.1\alpha + 60\beta &= 676 \\ t_{sim} + 0.6\alpha + 540\beta &= 1261 \\ t_{sim} + 80\alpha + 180\beta &= 1322 \end{aligned} \tag{7.5}$$

Solving this system of equations gives the following values: t_{sim}=603 , α=1.2, and β=6.3. That is, it takes 603 s to perform the simulation (for six simulated months), 1.2 s to produce one image, and 6.3 s to read/write 1 GB of data.

Model validation We plot the measured execution time against the modeled execution time in Figure 7.6. We observe that our model predicts the execution time accurately in all cases. The absolute error rate achieved was less than 0.5%.

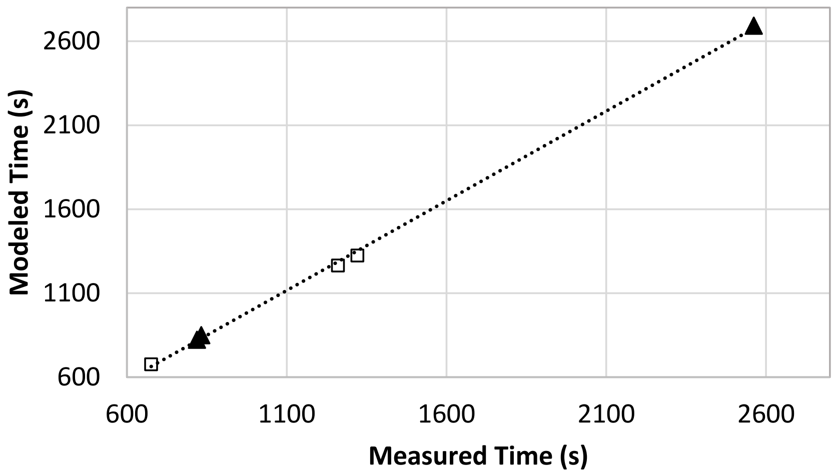


Figure 7.6: Evaluation of our model for execution time. White squares denote the data points used for constructing the model. Black triangles denote the data points used in evaluation.

Given a sample rate, it is also possible to estimate the storage requirements accurately. The storage size scales linearly with the sampling rate. That is, for example, when one samples at twice the rate of a reference configuration, the storage requirements double correspondingly:

$$S_{i/o.any} = S_{i/o.ref} \times \frac{rate_{any}}{rate_{ref}} \tag{7.6}$$

where, $rate_{ref}$ and $rate_{any}$ refer to the output sampling rate used in the reference configuration and the target configuration for which performance metrics must be estimated, respectively. Likewise, $S_{i/o.ref}$ and $S_{i/o.any}$ refer to the size of output produced for the reference configuration and target configuration, respectively. The data presented in Figure 7.4 is in agreement with Equation (7.6) for both the *in-situ* and post-processing cases, which validates our model for storage requirements. A similar equation is used to estimate the number of images produced during a simulation:

$$N_{viz.any} = N_{viz.ref} \times \frac{rate_{any}}{rate_{ref}} \tag{7.7}$$

where $N_{viz.any}$ and $N_{viz.ref}$ refer to the number of images produced for the reference configuration and target configuration, respectively.

Using our model, one can estimate the execution time, energy, and storage for any sampling rate $rate_{any}$ and timesteps $iter_{any}$ with data collected from one short run of the simulation.

7.5 What-If Analysis with Empirical Model

With the model shown in Equation (7.4) we can evaluate many scenarios and help domain scientists optimize their application configurations given various constraints. As an example, we show how to optimize a pipeline for a given energy target.

Energy vs. sampling rate Our model can evaluate the energy that is necessary to run a hundred-year simulation given a target sampling rate. Figure 7.7 shows the energy consumed by the two pipelines at different sampling frequencies. The x-axis represents how often the output products are written in terms of simulated hours. The y-axis represents the energy that would be consumed while running a hundred-year simulation. This graph can be used to evaluate the energy that can be saved from *in-situ* visualization under different sampling assumptions. Assume that the climate scientists need to track the eddies by the hour. In this case, using *in-situ* techniques will help them save 67.2% of the energy needed for the workflow. If the required sampling rate is once every 12 hours, up to 49.0% energy can be saved using *in-situ* techniques. Similarly, 38.0% of workflow energy can be saved at a sampling rate of once per day. Using the model, one can also evaluate the sampling rate possible for a given energy budget.

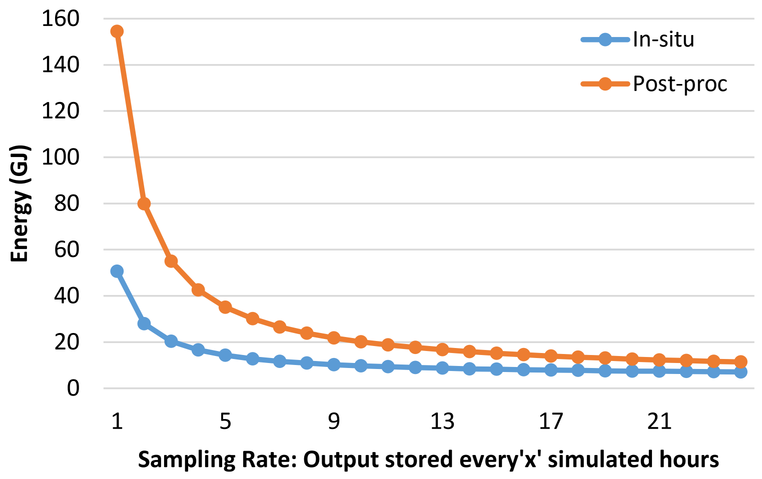


Figure 7.7: Relationship between energy and sampling rate

7.6 Summary

In this chapter, we showed *in-situ* pipeline runs 51% faster, consumes 50% less energy, and occupies 99.5% less disk space than a post-processing pipeline for an ocean simulation application. This means nearly 10% of the total power which is spent towards the storage subsystem can be re-routed to the compute units. Our preliminary investigation showed that we can improve the performance by re-routing storage power budget to compute by as much as 6.4%. Then, we showed how to model the power and energy consumption via analytical methods. We also demonstrated how to use such a model for optimizing a visualization pipeline under an energy budget.

Chapter 8

Conclusion and Future Work

The human brain has the remarkable ability to compensate for missing information [60, 126]. For instance, when presented with a picture with several missing pieces, it can still form a mental image of what the likely picture is. Analogous to that, in this dissertation, we tried to model and improve the greenness of HPC systems with limited information. We showed that it is indeed possible to build useful estimation and prediction models which can help improve the greenness of computing systems.

8.1 Dissertation Summary

Since data movement is considered to be one of the biggest sources of power consumption in future processors, we began this dissertation by trying to understand what factors affect the cost of data movement. More specifically, we studied the power and energy consumption of the chip interconnect. In the past, several researchers have attempted to measure the power and energy cost of various components within a processor via targeted micro-benchmarking. However, the chip interconnect is a particularly difficult component to isolate and study as it is difficult to stress an interconnect without stressing its end points. Therefore, we developed a novel micro-benchmarking methodology where we can change the distance that data has to travel within a processor without affecting the activity in the rest of the processor. We implemented our microbenchmarking methodology on real hardware and characterized the interconnect's power under different conditions. Using the data gathered from our extensive characterization study, we constructed a detailed power model. We validated the power model against vendor-provided data. An analysis of several applications using our model and performance counter data gathered on real hardware showed that the chip interconnect would be a major power consumer in future processors.

We also investigated the utility of a performance counter-based (PC-based) model in a run-time setting. We performed this study in the context of a graphics processing unit (GPU). More specifically, we investigated the utility of a PC-based model to act as power proxy which is capable of providing power measurements at run-time. This is an interesting use case because this scenario stresses the limitations of the hardware infrastructure which limits the number of performance counters that can be simultaneously monitored. While the hardware imposes such limits, a power model would ideally need as many counters as possible representing all possible hardware blocks. Our exploration of this problem revealed that despite hardware limitations, it is possible to use a PC-based model for runtime power monitoring as explained next. First, we showed that by applying *stepwise regression* and *statistics-driven heuristics* to increase useful information during model training we can reduce redundant parameters of the model. We showed that the accuracy of the PC-model can be enhanced with information from temperature sensors. We demonstrated that application-specific knowledge can significantly improve the accuracy of the runtime power-model, with the mean error going down from 6% to 1%. This approach can be made practical by using a low-level power meter when an application is running in order to fine-tune the model.

While the first part of the dissertation primarily focused on understanding greenness at different levels, in the second part our emphasis was on the application of such models to provide useful actionable insights that improved the greenness of applications.

Since we identified chip interconnect to be a major consumer of power in future processors, we focused on reducing its power consumption. We proposed to proactively manage the chip-interconnect power via dynamic voltage and frequency (DVFS) scaling. We demonstrated the potential of DVFS for chip interconnects using the power model we previously developed. We also showed that our proactive power-management technique comes close to realizing the full power-saving opportunity provided by DVFS. However, this would need a mechanism that would accurately predict an application's bandwidth requirements ahead of time. We investigated a few classical phase-prediction techniques and improved the prediction accuracy of the history table-based predictor by changing matching algorithm from exact matching to approximate matching and adding metadata to facilitate faster adaptation to changing phases. We limited the size of the history table and introduced a replacement algorithm to store commonly used patterns in the table. Together, these techniques solved several practical problems we identified with a detailed characterization study. Using a trace-based simulator, we demonstrated that proactive power management of the chip interconnect can be useful and reduce the data-movement power of a processor.

Then, we studied the problem of improving the greenness of a heterogeneous node via re-distributing the power allocated to different components. Building upon our past work in phase prediction and DVFS power modeling, we developed a technique to reconfigure the GPU quickly and near optimally in order to safely operate under a power budget. A power-sloshing mechanism on the CPU helps re-distribute the node-level power budget at runtime whenever an imbalance is observed. Using data collected from real hardware, we showed that dynamic redistribution of power budget can improve the greenness for some applications.

Finally, we studied the role played by off-chip data movement in affecting the greenness of a cluster. We performed characterization experiments and constructed analytical models that help provide insights to improve greenness of scientific visualization. Our experiments showed that in-situ methods can improve greenness by reducing off-chip data movement and system idling.

Overall, in our dissertation, we developed a broad set of approaches to model power consumption using data obtained from real hardware. The models thus developed can be used in many different ways to provide actionable insights to improve the greenness of a computing system. The accuracy of the models and the usefulness of the models were demonstrated at multiple levels of the system hierarchy.

8.2 Future Work

There are a number of directions in which our current research can be taken. We discuss them in the context of chip-interconnect power management, power redistribution, and in-situ workflows.

8.2.1 Crosstalk-Aware Interconnect Power Management

Noting that compressing data on the interconnect, while reducing interconnect bandwidth, could inadvertently increase power consumption, Pekhimenko et al. [122] propose toggle-aware data compression for on-chip interconnects. Our characterization results show that in addition to data toggle, crosstalk also has a major impact on interconnect power. Therefore, in addition to toggle-aware compression algorithms, crosstalk-aware compression schemes may also be an interesting future research direction. Such schemes would need to decide when to compress data depending upon the potential increase in latency compared to the reduction in toggle rate and crosstalk.

8.2.2 Power Management of Interconnect on Multiple DVFS Islands

Our work on proactive power management of interconnect assumed that the interconnect operates on a single DVFS island. In future processors, interconnects may exist on multiple DVFS islands to provide more power-saving opportunities. In this scenario, in addition to managing each island separately, the problem of inter-island communication should also be addressed. Furthermore, as we increase the number of islands, the potential power saved per island also diminishes, which may require investigating other phase prediction techniques that could be implemented in a smaller power budget than what we currently allow.

8.2.3 Power Management of a Heterogeneous Node

Both power sloshing and work distribution accomplish the same task: minimizing imbalance. In the Linux kernel, power management occurs in coordination with process scheduling. Likewise, in a heterogeneous system, with upcoming work distribution frameworks like CoreTSAR [140], it should be possible to integrate power-sloshing approaches with work-scheduling approaches. Such work-distribution frameworks are also capable of balancing computations across multiple heterogeneous devices. Likewise, we are interested in exploring power-sloshing approaches on multiple heterogeneous processors (e.g., CPUs + multiple GPUs, CPUs + GPUs + FPGAs) within a node. Furthermore, our work assumes that GPUs require CPUs to manage their DVFS state via software. When GPUs have native hardware support for managing their DVFS state, we can design more effective node-level management. Investigating this, particularly in the context of multiple devices requires further research.

8.2.4 Alternate In-situ Workflows

Past research has shown that multi-component applications can improve their performance and energy efficiency by intelligently placing the various application components to the hardware components of a heterogeneous node [89]. In our previous work, we demonstrated that by co-locating the simulation and data visualization (analytics) components of an in-situ scientific computation workflow on a CPU and embedded GPU within a board, we saved significant energy [3]. With nodes and clusters becoming increasingly heterogeneous, resources must be intelligently allocated to the applications to both minimize data movement and to map applications to an appropriate resource in the heterogeneous architecture. In the future, we plan to explore alternative workflows for in-situ computations via a design-space exploration framework such as exploration test harness (ETH) [2].

Bibliography

[1] Y. Abe, H. Sasaki, S. Kato, K. Inoue, M. Edahiro, and M. Peres. Power and Performance Characterization and Modeling of GPU-Accelerated Systems. In *28th IEEE International Parallel and Distributed Processing Symposium*, pages 113–122, May 2014.

[2] G. Abrams, V. Adhinarayanan, W. Feng, D. Rogers, J. Ahrens, and L. Wilson. ETH: A Framework for the Design-Space Exploration of Extreme-Scale Visualization. Technical report, Department of Computer Science, Virginia Polytechnic Institute & State University, 2017.

[3] V. Adhinarayanan, B. Dutta, and W. Feng. Making a Case for Green High-Performance Visualization via Embedded Graphics Processors. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 721–724. IEEE, 2018.

[4] V. Adhinarayanan and W. Feng. An automated framework for characterizing and subsetting GPGPU workloads. In *2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 307–317, 2016.

[5] V. Adhinarayanan, W. Feng, D. Rogers, J. Ahrens, and S. Pakin. Characterizing and Modeling Power and Energy for Extreme-Scale In-Situ Visualization. In *2017 IEEE*

International Parallel and Distributed Processing Symposium (IPDPS), pages 978–987, 2017.

[6] V. Adhinarayanan, I. Paul, J. L. Greathouse, W. Huang, A. Pattnaik, and W. Feng. Measuring and modeling on-chip interconnect power on real hardware. In *2016 IEEE International Symposium on Workload Characterization (IISWC)*, pages 1–11, 2016. Best Paper Award.

[7] V. Adhinarayanan, B. Subramaniam, and W. Feng. Online Power Estimation of Graphics Processing Units. In *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 245–254. IEEE, 2016.

[8] J. Ahrens, S. Jourdain, P. O'Leary, J. Patchett, D. Rogers, and M. Petersen. An Image-based Approach to Extreme Scale in Situ Visualization and Analysis. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pages 424–434, 2014.

[9] AMD. FirePro W9100 Data Sheet. `https://www.amd.com/Documents/FirePro_W9100_Data_Sheet.pdf`.

[10] AMD. Reference Guide: Southern Islands Series Instruction Set Architecture. `http://developer.amd.com/wordpress/media/2012/12/AMD_Southern_Islands_Instruction_Set_Architecture.pdf`, 2012.

[11] AMD. Reference Guide: Sea Islands Series Instruction Set Architecture. `http://developer.amd.com/wordpress/media/2013/07/AMD_Sea_Islands_Instruction_Set_Architecture.pdf`, 2013.

[12] APP SDK - A Complete Development Platform.

`http://developer.amd.com/tools-and-sdks/opencl-zone/amd-accelerated-parallel-processing-app-sdk/`.

[13] High-Bandwidth Memory (HBM): Reinventing Memory Technology. `https://www.amd.com/Documents/High-Bandwidth-Memory-HBM.pdf`.

[14] J. H. Anderson and F. N. Najm. Switching Activity Analysis and Pre-layout Activity Prediction for FPGAs. In *Proc. of the Int'l Workshop on System-level Interconnect Prediction (SLIP)*, 2003.

[15] Appro International, Milpitas, California, USA. *Appro GreenBlade 2 SR5110–GB512X User Manual*, Nov. 2011. Version 1.0.

[16] A. Arunkumar, E. Bolotin, B. Cho, U. Milic, E. Ebrahimi, O. Villa, A. Jaleel, C.-J. Wu, and D. Nellans. MCM-GPU: Multi-chip-module GPUs for continued performance scalability. *ACM SIGARCH Computer Architecture News*, 45(2):320–332, 2017.

[17] P. E. Bailey, D. K. Lowenthal, V. Ravi, B. Rountree, M. Schulz, and B. R. De Supinski. Adaptive Configuration Selection for Power-Constrained Heterogeneous Systems. In *Parallel Processing (ICPP), 2014 43rd International Conference on*, pages 371–380. IEEE, 2014.

[18] P. E. Bailey, A. Marathe, D. K. Lowenthal, B. Rountree, and M. Schulz. Finding the Limits of Power-Constrained Application Performance. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 79. ACM, 2015.

[19] A. C. Bauer, B. Geveci, and W. Schroeder. The paraview catalyst users guide, 2013.

[20] D. Bedard, M. Y. Lim, R. Fowler, and A. Porterfield. PowerMon: Fine-Grained and

Integrated Power Monitoring for Commodity Computer Systems. In *Proc. of the IEEE SoutheastCon 2010 (SoutheastCon)*, 2010.

[21] F. Bellosa. The Benefits of Event: Driven Energy Accounting in Power-sensitive Systems. In *Proceedings of the 9th Workshop on ACM SIGOPS European Workshop*, pages 37–42. ACM, 2000.

[22] K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, K. Hill, J. Hiller, et al. Exascale computing study: Technology challenges in achieving exascale systems. *Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Tech. Rep*, 15, 2008.

[23] R. Bertran, M. Gonzalez, X. Martorell, N. Navarro, and E. Ayguade. Decomposable and Responsive Power Models for Multicore Processors using Performance Counters. In *Proc. of the Int'l Conf. on Supercomputing (ICS)*, 2010.

[24] W. L. Bircher and L. K. John. Complete System Power Estimation Using Processor Performance Events. *IEEE Transactions on Computer*, 61(4):563–577, Apr. 2012.

[25] J. D. Booth, J. Kotra, H. Zhao, M. Kandemir, and P. Raghavan. Phase Detection with Hidden Markov Models for DVFS on Many-Core Processors. In *2015 IEEE 35th International Conference on Distributed Computing Systems*, pages 185–195. IEEE, 2015.

[26] S. Borkar. Exascale Computing – A Fact or Fiction. *Keynote Presentation at the 2013 International Parallel and Distributed Processing Symposium (IPDPS)*, 2013.

[27] S. Borkar. Exascale Computing - A Fact or a Fiction? In *Int'l Symp. on Parallel Distributed Processing (IPDPS)*, pages 3–3, 2013.

[28] M. Burtscher, I. Zecena, and Z. Zong. Measuring GPU Power with the K20 Built-in Sensor. In *Proceedings of the 7th Workshop on General Purpose Processing on Graphics Processing Units (GPGPU)*, pages 28–36. ACM, 2014.

[29] M. R. Casu and P. Giaccone. Rate-based vs Delay-based Control for DVFS in NoC. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1096–1101. EDA Consortium, 2015.

[30] P.-Y. Chang, M. Evers, and Y. N. Patt. Improving Branch Prediction Accuracy by Reducing Pattern History Table Interference. *International journal of parallel programming*, 25(5):339–362, 1997.

[31] S. Che, B. M. Beckmann, S. K. Reinhardt, and K. Skadron. Pannotia: Understanding Irregular GPGPU Graph Applications. In *Proc. of the IEEE Int'l Symp. on Workload Characterization (IISWC)*, 2013.

[32] S. Che, M. Boyer, J. Meng, D. Tarjan, J. Sheaffer, S.-H. Lee, and K. Skadron. Rodinia: A Benchmark Suite for Heterogeneous Computing. In *Proceedings of the 2009 IEEE International Symposium on Workload Characterization (IISWC)* , pages 44–54, Oct 2009.

[33] 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 *IEEE International Symposium on Workload Characterization (IISWC)*, pages 44–54. Ieee, 2009.

[34] I.-C. K. Chen, J. T. Coffey, and T. N. Mudge. Analysis of Branch Prediction via d=Data Compression. *ACM SIGPLAN Notices*, 31(9):128–137, 1996.

[35] J. Chen, A. Choudhary, S. Feldman, B. Hendrickson, C. Johnson, R. Mount, V. Sarkar, V. White, and D. Williams. Synergistic Challenges in Data-Intensive Science and

Exascale Computing. *DOE ASCAC Data Subcommittee Report, Department of Energy Office of Science*, pages 1–70, 2013.

[36] J. Chen et al. Synergistic Challenges in Data-Intensive Science and Exascale Computing. Technical report, DOE ASCAC, Mar. 2013.

[37] T. Chen, A. Rucker, and G. E. Suh. Execution time prediction for energy-efficient hardware accelerators. In *Proceedings of the 48th International Symposium on Microarchitecture*, pages 457–469. ACM, 2015.

[38] X. Chen, Z. Xu, H. Kim, P. Gratz, J. Hu, M. Kishinevsky, and U. Ogras. In-network Monitoring and Control Policy for DVFS of CMP Networks-on-Chip and Last Level Caches. In *Proceedings of the Sixth IEEE/ACM International Symposium on Networks on Chip (NoCS)*, pages 43–50. IEEE, 2012.

[39] X. Chen, Z. Xu, H. Kim, P. Gratz, J. Hu, M. Kishinevsky, and U. Ogras. In-network Monitoring and Control Policy for DVFS of CMP Networks-on-Chip and Last Level Caches. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 18(4):47, 2013.

[40] X. Chen, Z. Xu, H. Kim, P. V. Gratz, J. Hu, M. Kishinevsky, U. Ogras, and R. Ayoub. Dynamic Voltage and Frequency Scaling for Shared Resources in Multicore Processor Designs. In *Proceedings of the 50th Annual Design Automation Conference (DAC)*, page 114. ACM, 2013.

[41] G. Contreras and M. Martonosi. Power Prediction for Intel XScale® Processors using Performance Monitoring Unit Events. In *Proc. of the Int'l Symp. on Low Power Electronics and Design (ISLPED)*, 2005.

[42] W. Dally. Challenges for Future Computing Systems. Available at `https://www.cs.colostate.edu/~cs575dl/Sp2015/Lectures/Dally2015.pdf`.

[43] D. C. Daly, L. C. Fujino, and K. C. Smith. Through the Looking Glass-The 2018 Edition: Trends in Solid-State Circuits from the 65th ISSCC. *IEEE Solid-State Circuits Magazine*, 10(1):30–46, 2018.

[44] A. Danalis, G. Marin, C. McCurdy, J. S. Meredith, P. C. Roth, K. Spafford, V. Tipparaju, and J. S. Vetter. The Scalable Heterogeneous Computing (SHOC) Benchmark Suite. In *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units (GPGPU)*, pages 63–74. ACM, 2010.

[45] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le. RAPL: Memory Power Estimation and Capping. In *Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design*, pages 189–194. ACM, 2010.

[46] J. D. Davis, S. Rivoire, M. Goldszmidt, and E. K. Ardestani. CHAOS: Composable Highly Accurate OS-based Power Models. In *Proceedings of the 2012 IEEE International Symposium on Workload Characterization (IISWC)*, pages 153–163. IEEE, Nov. 2012.

[47] DesignForward Program. `https://sites.google.com/a/lbl.gov/exascale-initiative/design-forward`.

[48] E. Duesterwald, C. Cascaval, and S. Dwarkadas. Characterizing and Predicting Program Behavior and its Variability. In *Proceedings of the 12th International Conference on Parallel Architectures and Compilation Techniques*, pages 220–231. IEEE, 2003.

[49] B. Dutta, V. Adhinarayanan, and W. Feng. GPU Power Prediction via Ensemble

Machine Learning for DVFS Space Exploration. In *ACM International Conference on Computing Frontiers (CF)*, Ischia, Italy, May 2018.

[50] FastForward Program. `https://sites.google.com/a/lbl.gov/exascale-initiative/fast-forward`.

[51] W. Feng. Making a case for efficient supercomputing. *Queue*, 1(7):54, 2003.

[52] W. Feng, H. Lin, T. Scogland, and J. Zhang. OpenCL and the 13 dwarfs: A Work in Progress. In *Proc. of the Int'l Conf. on Performance Engineering (ICPE)*, 2012.

[53] R. Garg, S. W. Son, M. Kandemir, P. Raghavan, and R. Prabhakar. Markov Model Based Disk Power Management for Data Intensive Workloads. In *2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 76–83. IEEE, 2009.

[54] R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, and K. W. Cameron. Powerpack: Energy Profiling and Analysis of High-Performance Systems and Applications. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 21(5):658–671, 2010.

[55] N. Gholkar, F. Mueller, and B. Rountree. Power Tuning HPC Jobs on Power-Constrained Systems. In *Proceedings of the 2016 International Conference on Parallel Architectures and Compilation*, pages 179–191. ACM, 2016.

[56] N. Gholkar, F. Mueller, B. Rountree, and A. 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*, pages 106–117. ACM, 2018.

[57] S. Ghosh, S. Chandrasekaran, and B. Chapman. Statistical Modeling of Power/Energy

of Scientific Kernels on a Multi-GPU System. In *Proceedings of the 2013 International Green Computing Conference (IGCC)*, pages 1–6. IEEE, June 2013.

[58] The Green500 List. `http://www.green500.org/greenlists`.

[59] J. Guerreiro, A. Ilic, N. Roma, and P. Tomás. GPGPU Power Modeling for Multi-domain Voltage-Frequency Scaling. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 789–800. IEEE, 2018.

[60] K. Harmon. The brain adapts in a blink to compensate for missing information. *Sci Am. http://www. scientificamerican. com/article/brain-adapts-in-a-blink*, 2015.

[61] M. A. Heroux, D. W. Doerfler, P. S. Crozier, J. M. Willenbring, H. C. Edwards, A. Williams, M. Rajan, E. R. Keiter, H. K. Thornquist, and R. W. Numrich. Improving Performance via Mini-Applications. *Sandia National Laboratories, Tech. Rep. SAND2009-5574*, 3, 2009.

[62] M. A. Heroux, D. W. Doerfler, P. S. Crozier, J. M. Willenbring, H. C. Edwards, A. Williams, M. Rajan, E. R. Keiter, H. K. Thornquist, and R. W. Numrich. Improving Performance via Mini-Applications. *Sandia National Laboratories, Tech. Rep. SAND2009-5574*, 3, 2009.

[63] R. Hesse and N. E. Jerger. Improving DVFS in NoCs with Coherence Prediction. In *Proceedings of the 9th International Symposium on Networks-on-Chip (NOCS)*, page 24. ACM, 2015.

[64] Hewlett-Packard. HP Apollo 8000 System Manager. Available at `http://www.hp.com/hpinfo/newsroom/press_kits/2014/HPDiscover2014/Apollo8000datasheet.pdf`.

[65] S. Hong and H. Kim. An Integrated GPU Power and Performance Model. In *Proceed-*

ings of the 37th Annual International Symposium on Computer Architecture (ISCA), pages 280–289. ACM, June 2010.

[66] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar. A 5-GHz Mesh Interconnect for a Teraflops Processor. *IEEE Micro*, 27(5):51–61, 2007.

[67] C. Hsu and W. Feng. A Power-Aware Run-Time System for High-Performance Computing. In *Proceedings of the 2005 ACM/IEEE Conference on High Performance Networking, Computing, Storage and Analysis (SC)*, pages 1–9, Nov. 2005.

[68] C.-H. Hsu and W. Feng. The Right Metric for Efficient Supercomputing: A Ten-Year Retrospective. In *Parallel and Distributed Processing Symposium Workshops, 2016 IEEE International, pages=1090–1093*. IEEE, 2016.

[69] C.-H. Hsu and S. W. Poole. Power measurement for high performance computing: State of the art. In *Green Computing Conference and Workshops (IGCC), 2011 International*, pages 1–6. IEEE, 2011.

[70] C. Isci, A. Buyuktosunoglu, C.-Y. Cher, P. Bose, and M. Martonosi. An Analysis of Efficient Multi-core Global Power Management Policies: Maximizing Performance for a Given Power Budget. In *Proceedings of the 39th annual IEEE/ACM international symposium on microarchitecture*, pages 347–358. IEEE Computer Society, 2006.

[71] C. Isci, A. Buyuktosunoglu, and M. Martonosi. Long-Term Workload Phases: Duration Predictions and Applications to DVFS. *IEEE Micro*, 1(5):39–51, 2005.

[72] C. Isci, G. Contreras, and M. Martonosi. Live, Runtime Phase Monitoring and Prediction on Real Systems with Application to Dynamic Power Management. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 359–370. IEEE Computer Society, 2006.

[73] C. Isci, G. Contreras, and M. Martonosi. Live, Runtime Phase Monitoring and Prediction on Real Systems with Application to Dynamic Power Management. In *39th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 359–370. IEEE, 2006.

[74] C. Isci and M. Martonosi. Runtime Power Monitoring in High-End Processors: Methodology and Empirical Data. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE Computer Society, 2003.

[75] C. Isci and M. Martonosi. Detecting Recurrent Phase Behavior under Real-System Variability. In *Proceedings of the IEEE International Workload Characterization Symposium*, pages 13–23. IEEE, 2005.

[76] C. Isci and M. Martonosi. Phase Characterization for Power: Evaluating Control-flow-based and Event-counter-based Techniques. In *Proceedings of the 12th International Symposium on High-Performance Computer Architecture (HPCA)*, pages 121–132. IEEE, Feb. 2006.

[77] C. Isci and M. Martonosi. Phase Characterization for Power: Evaluating Control-Flow-Based and Event-Counter-Based techniques. In *HPCA*, volume 3, pages 121–132, 2006.

[78] N. E. Jerger, A. Kannan, Z. Li, and G. H. Loh. NoC Architectures for Silicon Interposer Systems: Why pay for more wires when you can get them (from your interposer) for free? In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 458–470. IEEE, 2014.

[79] D. Joseph and D. Grunwald. Prefetching using Markov Predictors. *IEEE transactions on computers*, 48(2):121–133, 1999.

[80] A. B. Kahng, B. Li, L.-S. Peh, and K. Samadi. ORION 2.0: A Fast and Accurate NoC Power and Area Model for Early-Stage Design Space Exploration. In *Proc. of the Conf. on Design, Automation and Test in Europe (DATE)*, 2009.

[81] A. B. Kahng, B. Lin, and S. Nath. ORION3.0: A Comprehensive NoC Router Estimation Tool. *IEEE Embedded Systems Letters*, 7(2):41–45, 2015.

[82] I. Karlin, J. Keasler, and R. Neely. LULESH 2.0 Updates and Changes. Technical Report LLNL-TR-641973, LLNL, August 2013.

[83] I. Karlin, J. Keasler, and R. Neely. LULESH 2.0 Updates and Changes. Technical Report LLNL-TR-641973, LLNL, Aug. 2013.

[84] K. Kasichayanula, D. Terpstra, P. Luszczek, S. Tomov, S. Moore, and G. D. Peterson. Power Aware Computing on GPUs. In *Proceedings of the 2012 Symposium on Application Accelerators in High Performance Computing (SAAHPC)*, pages 64–73. IEEE, July 2012.

[85] S. W. Keckler, W. J. Dally, B. Khailany, M. Garland, and D. Glasco. GPUs and the Future of Parallel Computing. *IEEE Micro*, 1(5):7–17, 2011.

[86] G. Kestor, R. Gioiosa, D. J. Kerbyson, and A. Hoisie. Enabling Accurate Power Profiling of HPC Applications on Exascale Systems. In *Proc. of the Int'l Workshop on Runtime and Operating Systems for Supercomputers (ROSS@ICS)*, 2013.

[87] G. Kestor, R. Gioiosa, D. J. Kerbyson, and A. Hoisie. Quantifying the Energy Cost of Data Movement in Scientific Applications. In *Proc. of the IEEE Int'l Symp. on Workload Characterization (IISWC)*, 2013.

[88] P. Kogge et al. ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems. Technical report, DARPA IPTO, Sept. 2008.

[89] K. Krommydas and W. Feng. Telescoping Architectures: Evaluating Next-Generation Heterogeneous Computing. In *2016 IEEE 23rd International Conference on High Performance Computing (HiPC)*, pages 162–171. IEEE, 2016.

[90] M. Larabel and M. Tippett. Phoronix Test Suite. `http://www.phoronix-test-suite.com/`, 2011.

[91] J. H. Laros, P. Pokorny, and D. DeBonis. PowerInsight - A Commodity Power Measurement Capability. In *Proc. of the Int'l Green Computing Conf. (IGCC)*, 2013.

[92] M. LeBeane, K. Hamidouche, B. Benton, M. Breternitz, S. K. Reinhardt, and L. K. John. GPU triggered networking for intra-kernel communications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 22. ACM, 2017.

[93] C. Lefurgy, X. Wang, and M. Ware. Power Capping: A Prelude to Power Shifting. *Cluster Computing*, 11(2):183–195, 2008.

[94] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi. GPUWattch: Enabling Energy Optimizations in GPGPUs. In *Proc. of the Int'l Symp. on Computer Architecture (ISCA)*, 2013.

[95] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures. In *Proc. of the Int'l Symp. on Microarchitecture (MICRO)*, 2009.

[96] D. Lo, L. Cheng, R. Govindaraju, L. A. Barroso, and C. Kozyrakis. Towards energy proportionality for large-scale latency-critical workloads. In *2014 ACM/IEEE 41st*

International Symposium on Computer Architecture (ISCA), pages 301–312. IEEE, 2014.

[97] G. H. Loh, N. E. Jerger, A. Kannan, and Y. Eckert. Interconnect-Memory Challenges for Multi-Chip, Silicon Interposer Systems. In *Proceedings of the 2015 International Symposium on Memory Systems (MEMSYS)*, pages 3–10. ACM, 2015.

[98] R. Lucas, Ed. Top Ten Exascale Research Challenges. *DOE ASCAC Subcommittee Report*, pages 1–86, 2014.

[99] X. Ma, M. Dong, L. Zhong, and Z. Deng. Statistical Power Consumption Analysis and Modeling for GPU-based Computing. In *Proceedings of the Workshop on Power Aware Computing and Systems (HotPower)*. ACM, 2009.

[100] N. Magen, A. Kolodny, U. Weiser, and N. Shamir. Interconnect-Power Dissipation in a Microprocessor. In *Proceedings of the 2004 international workshop on System level interconnect prediction*, pages 7–13. ACM, 2004.

[101] A. Majumdar, L. Piga, I. Paul, J. L. Greathouse, W. Huang, and D. H. Albonesi. Dynamic GPGPU Power Management using Adaptive Model Predictive Control. In *Proceedings of the 2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 613–624. IEEE, 2017.

[102] A. Majumdar, L. Piga, I. Paul, J. L. Greathouse, W. Huang, and D. H. Albonesi. Dynamic GPGPU Power Management using Adaptive Model Predictive Control. In *2017 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 613–624. IEEE, 2017.

[103] I. Manousakis and D. S. Nikolopoulos. BTL: A Framework for Measuring and Modeling

Energy in Memory Hierarchies. In *Proc. of the Int'l Symp. on Computer Architecture and High Performance Computing (SBAC-PAD)*, 2012.

[104] A. Marathe, P. E. Bailey, D. K. Lowenthal, B. Rountree, M. Schulz, and B. R. de Supinski. A Run-Time System for Power-Constrained HPC Applications. In *International conference on high performance computing*, pages 394–408. Springer, 2015.

[105] A. K. Mishra, R. Das, S. Eachempati, R. Iyer, N. Vijaykrishnan, and C. R. Das. A case for dynamic frequency tuning in on-chip networks. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 292–303. IEEE, 2009.

[106] J. Mohd-Yusof. Codesign of Molecular Dynamics (CoMD) Proxy App. Technical report, LA-UR-12-21782, Los-Alamos National Lab, 2012.

[107] D. Molka, D. Hackenberg, R. Schöne, and M. S. Müller. Characterizing the Energy Consumption of Data Transfers and Arithmetic Operations on x86- 64 Processors. In *Proc. of the Int'l Green Computing Conf. (IGCC)*, 2010.

[108] R. C. Murphy, K. B. Wheeler, B. W. Barrett, and J. A. Ang. Introducing the Graph 500. *Cray Users Group (CUG)*, 2010.

[109] H. Nagasaka, N. Maruyama, A. Nukada, T. Endo, and S. Matsuoka. Statistical Power Modeling of GPU Kernels Using Performance Counters. In *Proceedings of the 2010 International Green Computing Conference (IGCC)*, pages 115–122. IEEE, Aug. 2010.

[110] S. G. Narendra, L. C. Fujino, and K. C. Smith. Through the looking glass? The 2015 edition: Trends in solid-state circuits from ISSCC. *IEEE Solid-State Circuits Magazine*, 7(1):14–24, 2015.

[111] Nvidia Corporation. *CUPTI: User's Guide*, July 2013.

[112] Nvidia Corporation. *NVML API Reference Manual*, Aug. 2013.

[113] U. Y. Ogras, R. Marculescu, and D. Marculescu. Variation-Adaptive Feedback Control for Networks-on-Chip with Multiple Clock Domains. In *Proceedings of the 45th annual Design Automation Conference*, pages 614–619. ACM, 2008.

[114] S. Pakin, C. Storlie, M. Lang, R. E. Fields, E. E. Romero, C. Idler, S. Michalak, H. Greenberg, J. Loncaric, R. Rheinheimer, et al. Power usage of production supercomputers and production workloads. *Concurrency and Computation: Practice and Experience*, 28(2):274–290, 2016.

[115] D. Pandiyan and C.-J. Wu. Quantifying the Energy Cost of Data Movement for Emerging Smart Phone Workloads on Mobile Platforms. In *Proc. of the IEEE Int'l Symp. on Workload Characterization (IISWC)*, pages 171–180. IEEE, 2014.

[116] PathForward Program. `https://asc.llnl.gov/pathforward/`.

[117] T. Patki, D. K. Lowenthal, B. Rountree, M. Schulz, and B. R. De Supinski. Exploring Hardware Overprovisioning in Power-Constrained, High Performance Computing. In *Proceedings of the 27th international ACM conference on International conference on supercomputing*, pages 173–182. ACM, 2013.

[118] T. Patki, D. K. Lowenthal, A. Sasidharan, M. Maiterth, B. L. Rountree, M. Schulz, and B. R. De Supinski. Practical Resource Management in Power-Constrained, High Performance Computing. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, pages 121–132. ACM, 2015.

[119] I. Paul, W. Huang, M. Arora, and S. Yalamanchili. Harmonia: Balancing compute and memory power in high-performance GPUs. In *ACM SIGARCH Computer Architecture News*, volume 43(3), pages 54–65. ACM, 2015.

[120] I. Paul, S. Manne, M. Arora, W. L. Bircher, and S. Yalamanchili. Cooperative boosting: needy versus greedy power management. In *ACM SIGARCH Computer Architecture News*, volume 41 (3), pages 285–296. ACM, 2013.

[121] I. Paul, V. Ravi, S. Manne, M. Arora, and S. Yalamanchili. Coordinated energy management in heterogeneous processors. *Scientific Programming*, 22(2):93–108, 2014.

[122] G. Pekhimenko, E. Bolotin, M. O'Connor, O. Mutlu, T. C. Mowry, and S. W. Keckler. Toggle-Aware Compression for GPUs. *IEEE Computer Architecture Letters*, 14(2):164–168, July 2015.

[123] L.-N. Pouchet. Polybench: The Polyhedral Benchmark Suite. *URL: http://www. cs. ucla. edu/pouchet/software/polybench*, 2012.

[124] M. D. Powell, A. Biswas, J. S. Emer, S. S. Mukherjee, B. R. Sheikh, and S. Yardi. CAMP: A Technique to Estimate Per-Structure Power at Run-time using a Few Simple Parameters. In *Proc. of the Int'l Symp. on High Performance Computer Architecture (HPCA)*, 2009.

[125] ECP Proxy Apps. `https://proxyapps.exascaleproject.org/ecp-proxy-apps-suite/`.

[126] V. S. Ramachandran. Filling in gaps in perception: Part I. *Current Directions in Psychological Science*, 1(6):199–205, 1992.

[127] K. Ramani et al. PowerRed: A Flexible Modeling Framework for Power Efficiency Exploration in GPUs. In *Workshop on GPGPU*, 2007.

[128] T. Ringler, M. Petersen, R. L. Higdon, D. Jacobsen, P. W. Jones, and M. Maltrud. A Multi-Resolution Approach to Global Ocean Modeling. *Ocean Modelling*, 69(0):211 – 232, 2013.

[129] S. Rivoire, P. Ranganathan, and C. Kozyrakis. A Comparison of High-level Full-system Power Models. In *Proceedings of the 2008 Workshop on Power Aware Computing and Systems (HotPower)*. USENIX, 2008.

[130] B. Rountree, D. H. Ahn, B. R. De Supinski, D. K. Lowenthal, and M. Schulz. Beyond DVFS: A First Look at Performance Under a Hardware-Enforced Power Bound. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International*, pages 947–953. IEEE, 2012.

[131] B. Rountree, D. K. Lownenthal, B. R. de Supinski, M. Schulz, V. W. Freeh, and T. Bletsch. Adagio: Making DVS Practical for Complex HPC Applications. In *Proceedings of the 23rd International Conference on Supercomputing (ICS)*, pages 460–469. ACM, June 2009.

[132] R. Sakamoto, T. Cao, M. Kondo, K. Inoue, M. Ueda, T. Patki, D. Ellsworth, B. Rountree, and M. Schulz. Production Hardware Overprovisioning: Real-World Performance Optimization Using an Extensible Power-Aware Resource Management Framework. In *Parallel and Distributed Processing Symposium (IPDPS), 2017 IEEE International*, pages 957–966. IEEE, 2017.

[133] P. Salihundam, S. Jain, T. Jacob, S. Kumar, V. Erraguntla, Y. Hoskote, S. Vangal, G. Ruhl, and N. Borkar. A 2 Tb/s 6x4 Mesh Network for a Single-Chip Cloud Computer With DVFS in 45 nm CMOS. *IEEE Journal of Solid-State Circuits*, 46(4):757–766, 2011.

[134] R. Sarikaya, C. Isci, and A. Buyuktosunoglu. Runtime Workload Behavior Prediction Using Statistical Metric Modeling with Application to Dynamic Power Management. In *2010 IEEE International Symposium on Workload Characterization (IISWC)*, pages 1–10. IEEE, 2010.

[135] R. Sarikaya, C. Isci, and A. Buyuktosunoglu. Runtime workload behavior prediction using statistical metric modeling with application to dynamic power management. In *2010 IEEE International Symposium on Workload Characterization (IISWC)* , pages 1–10, Dec 2010.

[136] O. Sarood, A. Langer, A. Gupta, and L. Kale. Maximizing Throughput of Overprovisioned HPC Data Centers under a Strict Power Budget. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 807–818. IEEE Press, 2014.

[137] 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. IEEE, 2013.

[138] L. Savoie, D. K. Lowenthal, B. R. De Supinski, T. Islam, K. Mohror, B. Rountree, and M. Schulz. I/O Aware Power Shifting. In *Parallel and Distributed Processing Symposium, 2016 IEEE International*, pages 740–749. IEEE, 2016.

[139] M. J. Schulte, M. Ignatowski, G. H. Loh, B. M. Beckmann, W. C. Brantley, S. Gurumurthi, N. Jayasena, I. Paul, S. K. Reinhardt, and G. Rodgers. Achieving Exascale Capabilities through Heterogeneous Computing. *IEEE Micro*, 35(4):26–36, 2015.

[140] T. R. Scogland, W. Feng, B. Rountree, and B. R. de Supinski. CoreTSAR: core task-size adapting runtime. *IEEE transactions on parallel and distributed systems*, 26(11):2970–2983, 2015.

[141] L. Shang, L.-S. Peh, and N. K. Jha. Power-Efficient Interconnection Networks: Dynamic Voltage Scaling with Links. *IEEE Computer Architecture Letters*, 1(1):6–6, 2002.

[142] Y. S. Shao and D. Brooks. Energy Characterization and Instruction-Level Energy Model of Intel's Xeon Phi Processor. In *Proc. of the Int'l Symp. on Low Power Electronics and Design (ISLPED)*, 2013.

[143] T. Sherwood, S. Sair, and B. Calder. Phase Tracking and Prediction. In *Proceedings of the 30th annual international symposium on Computer architecture (ISCA)*, pages 336–349. ACM, 2003.

[144] K. Singh, M. Bhadauria, and S. A. McKee. Real Time Power Estimation and Thread Scheduling via Performance Counters. *SIGARCH Computer Architecture News*, 37(2):46–55, July 2009.

[145] S. Song, C. Su, B. Rountree, and K. W. Cameron. A Simplified and Accurate Model of Power-Performance Efficiency on Emergent GPU Architectures. In *Proceedings of the 27th IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, pages 673–686. IEEE, May 2013.

[146] V. Soteriou and L.-S. Peh. Dynamic Power Management for Power Optimization of Interconnection Networks using On/Off Links. In *Proceedings of the 11th Symposium on High-Performance Interconnects (HotI)*, pages 15–20. IEEE, 2003.

[147] SPEC ACCEL benchmark suite. `http://www.spec.org/accel/`.

[148] J. A. Stratton, C. Rodrigues, I.-J. Sung, N. Obeid, L.-W. Chang, N. Anssari, G. D. Liu, and W.-m. W. Hwu. Parboil: A Revised Benchmark Suite for Scientific and Commercial Throughput Computing. *Center for Reliable and High-Performance Computing*, 127, 2012.

[149] J. Stuecheli. POWER8. In *Hot Chips Symposium*, pages 1–20, 2013.

[150] 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 *47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 445–457. IEEE, 2014.

[151] L. T. Su, S. Naffziger, and M. Papermaster. Multi-chip technologies to unleash computing performance gains over the next decade. In *2017 IEEE International Electron Devices Meeting (IEDM)*, pages 1–1. IEEE, 2017.

[152] C. Sun, C.-H. O. Chen, G. Kurian, L. Wei, J. Miller, A. Agarwal, L.-S. Peh, and V. Stojanovic. DSENT-a tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling. In *2012 Sixth IEEE/ACM International Symposium on Networks on Chip (NoCS)*, pages 201–210. IEEE, 2012.

[153] J. R. Tramm, A. R. Siegel, T. Islam, and M. Schulz. XSBench-The Development and Verification of a Performance Abstraction for Monte Carlo Reactor Analysis. *The Role of Reactor Physics toward a Sustainable Future (PHYSOR)*, 2014.

[154] A. N. Udipi, N. Muralimanohar, and R. Balasubramonian. Non-Uniform Power Access in Large Caches with Low-Swing Wires. In *Proc. of the Int'l Conf. on High Performance Computing (HiPC)*, 2009.

[155] T. Vijayaraghavan, Y. Eckert, G. H. Loh, M. J. Schulte, M. Ignatowski, B. M. Beckmann, W. C. Brantley, J. L. Greathouse, W. Huang, A. Karunanithi, et al. Design and Analysis of an APU for Exascale Computing. In *Proceedings of the 2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 85–96. IEEE, 2017.

[156] O. Villa, D. R. Johnson, M. O'Connor, E. Bolotin, D. Nellans, J. Luitjens, N. Sakharnykh, P. Wang, P. Micikevicius, A. Scudiero, et al. Scaling the Power Wall:

A Path to Exascale. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 830–841. IEEE Press, 2014.

[157] H.-S. Wang, X. Zhu, L.-S. Peh, and S. Malik. Orion: A Power-Performance Simulator for Interconnection Networks. In *Proc. of the Int'l Symp. on Microarchitecture (MICRO)*, 2002.

[158] WCCFTech. Dieshots of Pitcairn, Tahiti, and Hawaii GPUs. `http://cdn.wccftech.com/wp-content/uploads/2013/12/AMD-Hawaii-GPU.jpg`. Accessed: 2016-03-18.

[159] J.-Y. Won, X. Chen, P. Gratz, J. Hu, and V. Soteriou. Up by their Bootstraps: Online Learning in Artificial Neural Networks for CMP Uncore Power Management. In *Proceedings of the 20th International Symposium on High Performance Computer Architecture (HPCA)*, pages 308–319. IEEE, 2014.

[160] J. Woodring, M. Petersen, A. Schmeiber, J. Patchett, J. Ahrens, and H. Hagen. In Situ Eddy Analysis in a High-Resolution Ocean Climate Model. *IEEE Transactions on Visualization Computer Graphics*, 22(1):857–866, 2015.

[161] G. Wu, J. L. Greathouse, A. Lyashevsky, N. Jayasena, and D. Chiou. GPGPU Performance and Power Estimation using Machine Learning. In *21st International Symposium on High Performance Computer Architecture (HPCA)*, pages 564–576. IEEE, 2015.

[162] W. Wu, L. Jin, J. Yang, P. Liu, and S. X.-D. Tan. A Systematic Method for Functional Unit Power Estimation in Microprocessors. In *Proc. of the Design Automation Conf. (DAC)*, 2006.

[163] Y. Yao and Z. Lu. DVFS for NoCs in CMPs: A Thread Voting Approach. In *Proceedings of the 2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 309–320. IEEE, 2016.

[164] H. Zhang and H. Hoffmann. Maximizing Performance under a Power Cap: A Comparison of Hardware, Software, and Hybrid techniques. *ACM SIGARCH Computer Architecture News*, 44(2):545–559, 2016.