Power, Performance and Energy Models and Systems for Emergent Architectures

Shuaiwen Song

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 Applications

Kirk W. Cameron, Chair
Calvin J. Ribbens
Ali R. Butt
Rong Ge
Abhinav Vishnu
Barry L. Rountree

March 7, 2013
Blacksburg, Virginia

Keywords: High Performance Computing, Performance/Energy Modeling Techniques for Large Scaled Systems, Power-Performance Efficiency and Management, Runtime System

Power, Performance and Energy Models and Systems for Emergent Architectures

Shuaiwen Song

ABSTRACT

Massive parallelism combined with complex memory hierarchies and heterogeneity in high-performance computing (HPC) systems form a barrier to efficient application and architecture design. The performance achievements of the past must continue over the next decade to address the needs of scientific simulations. However, building an exascale system by 2022 that uses less than 20 megawatts will require significant innovations in power and performance efficiency.

A key limitation of past approaches is a lack of power-performance policies allowing users to quantitatively bound the effects of power management on the performance of their applications and systems. Existing controllers and predictors use policies fixed by a knowledgeable user to opportunistically save energy and minimize performance impact. While the qualitative effects are often good and the aggressiveness of a controller can be tuned to try to save more or less energy, the quantitative effects of tuning and setting opportunistic policies on performance and power are unknown. In other words, the controller will save energy and minimize performance loss in many cases but we have little understanding of the quantitative effects of controller tuning. This makes setting power-performance policies a manual trial and error process for domain experts and a black art for practitioners. To improve upon past approaches to high-

performance power management, we need to quantitatively understand the effects of power and performance at scale.

In this work, I have developed theories and techniques to quantitatively understand the relationship between power and performance for high performance systems at scale. For instance, our system-level, *iso-energy-efficiency* model analyzes, evaluates and predicts the performance and energy use of data intensive parallel applications on multi-core systems. This model allows users to study the effects of machine and application dependent characteristics on system energy efficiency. Furthermore, this model helps users isolate root causes of energy or performance inefficiencies and develop strategies for scaling systems to maintain or improve efficiency. I have also developed methodologies which can be extended and applied to model modern heterogeneous architectures such as GPU-based clusters to improve their efficiency at scale.

Acknowledgements

First and foremost, I would like to my graduate advisor Professor Kirk W. Cameron for his extremely valuable guidance and advice, constant support, and tremendous encouragement over the years. He has inspired me greatly to work in high performance computing (HPC) field. His willingness to motivate me contributes greatly to this thesis work. Besides, I want to give special thanks to my thesis committee members: Professor Calvin Ribbens, Professor Ali Butt, Dr. Barry Rountree, Dr. Abhinav Vishnu, and Professor Rong Ge. Each of them has devoted significant time and efforts to this thesis work and given me many insightful suggestions on how to improve this work. Also, I would like to take this opportunity to thank all my colleagues at SCAPE lab for their wonderful discussions on my thesis research and very useful ideas. Finally, an honorable mention goes to all my families and friends for their understandings and supports in completing this thesis.

Table of Contents

List of Figures

PAGE

List of Tables

PAGE

Chapter 1

Introduction

The demands of exascale computing systems and applications require improvements in our fundamental understanding of the relationship between power and performance. My thesis work explores the development, implementation, and evaluation of techniques that model the interactive effects of energy efficiency and performance. For example, I propose a new concept called "*iso-energy efficiency*" which captures the effects of changes in system and application configuration on execution time, power consumption, and energy efficiency. In this thesis work, I propose my research on a predictive model of system and application performance and energy use. I will describe herein detailed background and related work, my modeling approach in the context of previous work, the experimental evaluation methodologies (including profiling and evaluating, modeling and

analysis framework, and runtime system efficiency optimization), and experimental results. This thesis will also provide insights on how to identify and eliminate potential performance/energy bottlenecks for current mainstream heterogeneous systems in HPC environments.

1.1 Motivation

1.1.1 Need to Improve Energy Efficiency in HPC

As we step into the exascale computing era, High Performance Computing (HPC) systems and technologies have provided an unprecedented level of computational capability for solving traditionally insolvable or extremely challenging social and scientific problems where experiments are impossible, dangerous, or inordinately costly. New technology innovations and breakthroughs facilitated or accelerated by the newest HPC technologies have been widely seen in the scientific fields of aerospace, astrophysics, climate modeling and combustion, fusion energy, nuclear engineering, nanoscience, and computational biology [1]. 'Going to the exascale' will also mean a radical change in computing architecture – basically, vastly increasing the levels of parallelism to the point of millions of processors working in tandem – which will force radical changes in how hardware is designed, in how we go about solving problems, and in how we map application codes to the underlying hardware (e.g., the compilers, I/O, middleware, and related software tools [2, 3]).

a) "Race to the moon": DOE goal of building exascale supercomputer within 20 MW by 2022:

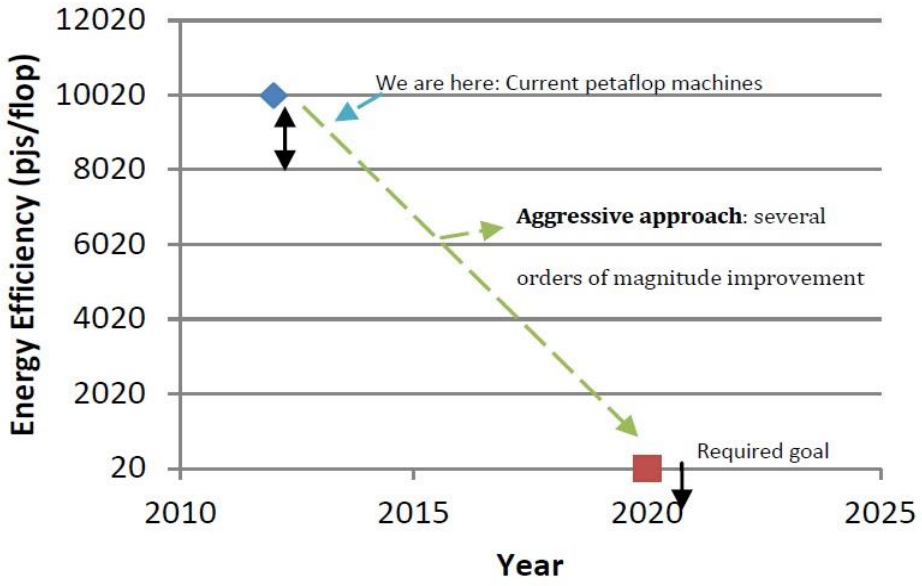


Figure 1.1**:** Energy efficiency: where we are now and where we want to be.

Today, one of the most significant hindrances for designing highly scalable data intensive applications and larger HPC systems to achieve exascale is the enormous gap between the demand for performance and the limit of the power envelope [4-6]. The US department of Energy has challenged the community to build a supercomputer capable of exascale computations using fewer than 20 MW of power by the year 2022 [3]. Fig 1.1 shows where we are now and where our future goal is in terms of energy consumption per flop. Current Top 500 petaflop supercomputers demand 2,000 to 10,000 pJs/flop. However, in order to achieve exaflop performance under 20 MW power, future systems can only consume 20 pJs/flop. This means that the energy efficiency of future

supercomputers has to be improved dramatically by several orders of magnitude over the most energy efficient systems of today.

To reach the 20 MW goal, we must achieve a 1000-fold performance increase over current petaflop systems with only a 10-fold increase in power consumption [7] by 2022. Figure 1.2 shows the maximum performance, power consumption and power efficiency of the top 5 fastest supercomputers (in descending order: RIKEN1*, NUDT2, Oak Ridge Cray XT5-HE3, Dawning4 and GSIC HP Proliant5. * represents current ranking of individual supercomputer) in the world from the November 2011 TOP 500 list [8] and also two highly ranked energy efficient clusters (IBM Rochester1 and Hokie Speed11) from the Green500 list [9]. RIKEN-K[10], the first supercomputer ever to achieve a performance level of 10 Petaflop/s, equipped with 705,024 2.0 GHz SPARC64 processing cores, is approximately 4 times faster than the 2nd place Chinese Tianhe-1A system and 31 times faster than the No.1 energy efficient supercomputer IBM Rochester. However, the total system power consumption of RIKEN is already 3 times that of Tianhe-1-A's and 74 times that of IBM Rochester. For power efficiency expressed as MFLOPS/W, "Green" supercomputers Hokie Speed and IBM Rochester [9] are 1.12 and 2.44 times more efficient than RIKEN-K, respectively. Based on current trends in performance and energy efficiency observed in the top supercomputers above, without more effective and aggressive power/energy conservation techniques, the goal of designing future systems under 20 MW is almost impossible to achieve due to exponentially increasing power demands. Therefore, identifying the root cause of the gap between increasing peak performance and energy, and developing corresponding optimization strategies are critical open problems in HPC.

b) Reliability:

Increasing the scale of HPC systems to achieve better performance has the unwelcome consequence of reduced system reliability due to heat emissions caused by high power consumption.

Increased power consumption can result in higher operating temperatures for parallel systems and dramatically reduce overall system reliability and availability. For instance, at the architecture level, thermal issues have become especially severe as the ability to reduce supply voltage has slowed [11]. As a result, the number of devices per unit area is scaling up faster than power density is scaling down (e.g. 3-D integration designs [11] make the challenge even more acute). Furthermore, due to degraded carrier mobility and temperature-dependent wear out rate, high temperatures can result in slower integrated circuits and accelerated multi-chip failures such as electromigration [12] and NBTI [13]. Despite these issues, current cooling solutions at the chip level are very limited [11].

For larger HPC systems and datacenters, overall system reliability can also be compromised due to the high system power and temperature. For example, a Google cluster with 450,000 processors has to be rebooted 60 times per day and experiences a 2 to 3 percent annual replacement rate [14]. According to an estimation method from D.A. Patterson's computer architecture book [15], the current No.1 supercomputer RIKEN-K may sustain hardware failures 50 times every 24 hours. In addition to installing expensive and complicated cooling systems, there have been a number of attempts to improve reliability by reducing system power consumption and operating temperature. For

instance, thermal management techniques [16-20] have used global frequency switching DVFS (Dynamic Voltage and Frequency Scaling) [21] or scheduling processes (or threads) running on specific cores in SMTs and CMPs to reduce "hotspots". Some use a combination of DVFS and temperature-aware load balancing to constrain core temperatures as well as save cooling energy [22, 23]. Others use a monitoring strategy based on hardware counters and profiling-driven techniques [24]. There are also a number of architecture-level simulation models for low level thermal-aware design

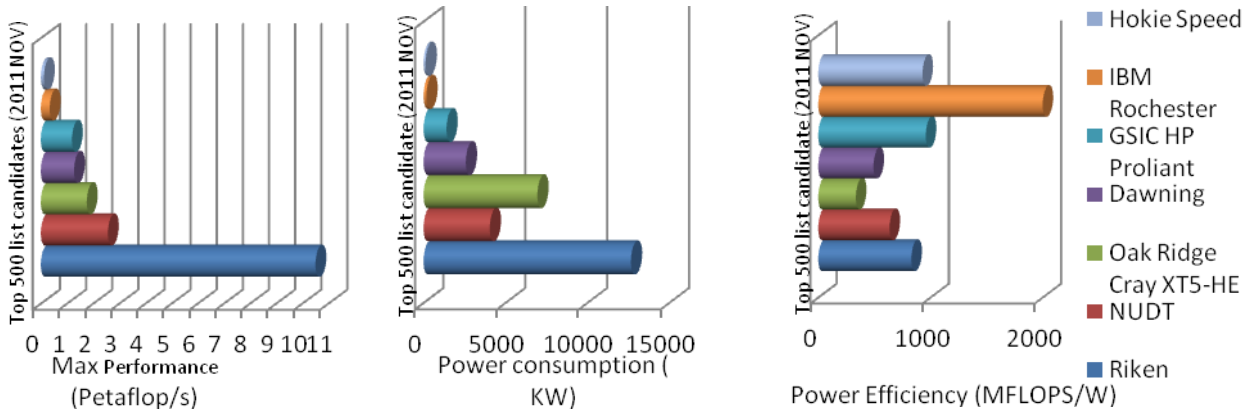


Figure 1.2: Maximum performance, power consumption and power efficiency comparisons for highly ranked supercomputers from the TOP500 and the Green 500 lists. Hokie Speed and IBM Rochester, are not among the top 5 in TOP500 list, but are highly ranked supercomputers for power efficiency (for example, IBM Rochester ranks No.1 on the Green500 list).

purposes such as PowerTime [25] and HotSpot [26].

c) Cost:

Increased power consumption directly produces a substantial amount of operating cost, including electricity bills, expenses for cooling facilities and extra space [23, 27]. For instance, the Jaguar supercomputer in Oak Ridge National Lab [8] requires 6.95MW

power to operate. With a utility rate of 12 cents per KW/hour, the annual electricity bill could reach as high as $7.4 million. This rough estimation doesn't include cooling expenses which can easily cost up to 40% of total system operating expenses [23, 27-29]. If we scale Jaguar to an even larger and more powerful petaflop machine which consumes 100 megawatts, it could cost $10,000 per hour and approximately $85 million a year [14].

1.1.2 Need to Understand Energy Efficiency in HPC

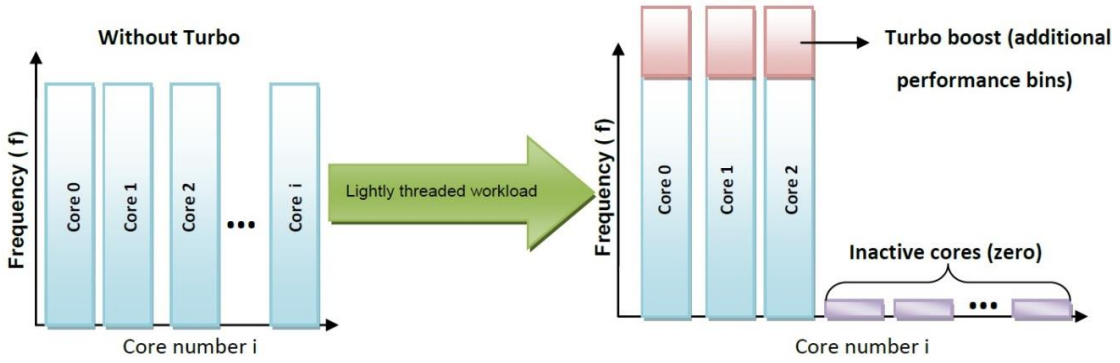


Figure 1.3: Intel Nehalem Turbo Boost Technology.

High operational costs and system failure rates associated with many-megawatt computing resources have increased the need to consider the entangled effects of power and performance in emergent exascale systems and applications [4, 30].

Raw performance/energy data is too abstract:

Despite the increasing importance of investigating system-wide energy efficiency, many techniques focus on measuring the total energy consumption and performance and then attempt to conclude how well the performance and energy efficiency of a specific application scales on a certain architecture. However, these state of the art approaches provide little or no insights to the factors causing the overall efficiency, how these factors scale, and how close we are to an optimal solution.

For example, Fig. 1.2 shows the power efficiency of several supercomputers. IBM Rochester clearly has better power efficiency than Hokie Speed and Riken-K. But, why does IBM Rochester have better power efficiency compared to the others in the graph? Where does the increased efficiency come from? Are there bottlenecks on one platform that are not present on another platform? We cannot answer these questions using only the raw performance/energy data. Understanding where and how power is consumed is the first step towards tracing energy efficiency in high performance systems and ultimately determining causal effects.

1.1.3 Opportunities and Challenges to Study Energy-Performance Tradeoffs for Emerging Architectures

a) Relationships between performance and energy efficiency are complicated:

Fig 1.4 shows the performance and energy efficiency curves for Fast Fourier Transform (FT). *Since the performance and energy efficiency curves do not track exactly with one another*, the performance efficiency cannot always predict system energy efficiency and vice versa. This phenomenon results from the interactive effects of both power and performance. If we could identify the root cause of energy inefficiency we

may be able to improve system and application efficiency. However, analyzing and potentially predicting energy efficiency is exceedingly difficult since we must identify and isolate the interactive effects of power and performance. For example, changing the power settings on a processor using dynamic voltage and frequency scaling (DVFS) affects performance which in turn potentially affects the length of time an application takes to complete which is key to its overall energy usage. Therefore, a system-level, accurate, energy efficiency model must capture these effects if it is to be used to explore the impact of application and system design choices on energy efficiency.

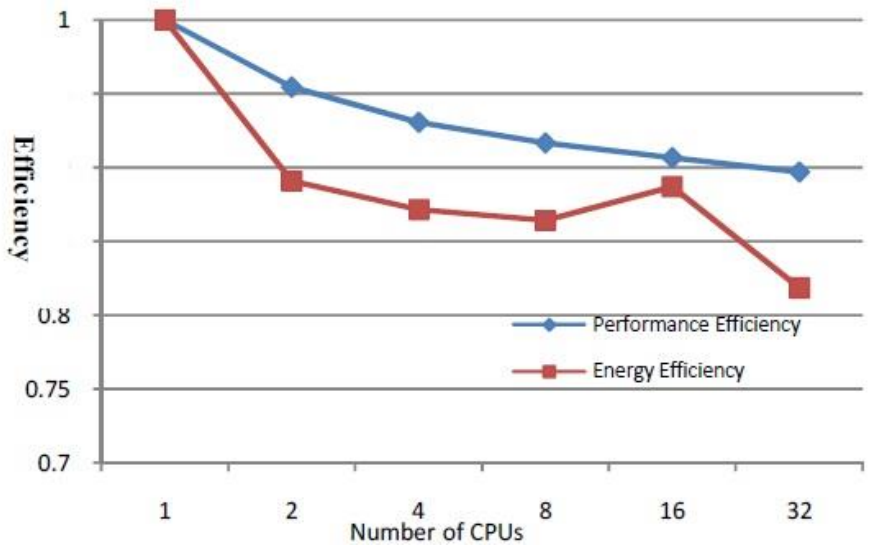


Figure 1.4: Performance and energy efficiency for 3D Fourier Transform from NAS Parallel Benchmark.

In addition to challenges associated with explaining the interactive effects of power and performance, modern architectures now employ more complex devices than ever before. In particular, the emergence of multi-core, heterogeneous (e.g. GPGPU-based) architectures complicates model development.

b) Multi-core architecture:

Multi-core architectures bring exciting opportunities and challenges for power and performance analysis at both micro-architecture and system levels. Traditional power-aware techniques have been applied to multi-core architectures to reduce overall system energy consumption. Examples include: a) architecture and circuit level power-aware design including simulation and modeling [31-33]; b) processor frequency scaling using DVS (Dynamic Voltage Scaling) or DVFS (Dynamic Voltage and Frequency Scaling) [34-36] based on workload characteristics (both balanced and imbalanced [35, 37]) and execution patterns under certain energy budget or performance thresholds; c) concurrency throttling and thread-core mapping [38-40] based on workload demand; d) dynamic task scheduling and process migration [41, 42]; e) low power approaches for embedded [43-47] and mobile systems. Other than example (e), the ultimate goal of these approaches is to reduce the maximum amount of power consumed without sacrificing the performance.

Recently developed architectures show promise for designing new power/energy optimization approaches for homogenous multi-core systems. For example, the Intel Core i7 (Nehalem) provides new package level power management including a dedicated power control unit (PCU) and Intel Turbo Boost Technology [48] (shown in Fig. 1.3). However, even with the advance of the recent multi-core architectures like Nehalem and their related new power optimization tools, the challenges still remain: how can we quantitatively understand the effects of power and performance at scale? What are the major parameters that impact the overall system performance and energy efficiency? How do they interact with each other? Can we scale them effectively in order to achieve

better efficiency?

c) GPGPU architecture:

Recent heterogeneous GPGPU accelerated systems have drawn the attention of both researchers and engineers due to their abundant cores and arithmetic computational units for massive parallel and computation intensive workloads (3 out of 5 top supercomputers in the world are built with NVIDIA C2050 Fermi GPUs [8]). However, integrating the GPUs into the already power-consuming HPC systems needs to be carefully assessed with respect to the impacts on system energy efficiency [49]. For example, Figure 1.5 (a) shows a power profiling of naïve Matrix Multiplication (MM) running on both GPU and CPU systems. The GPU consumes 45 watts more power than

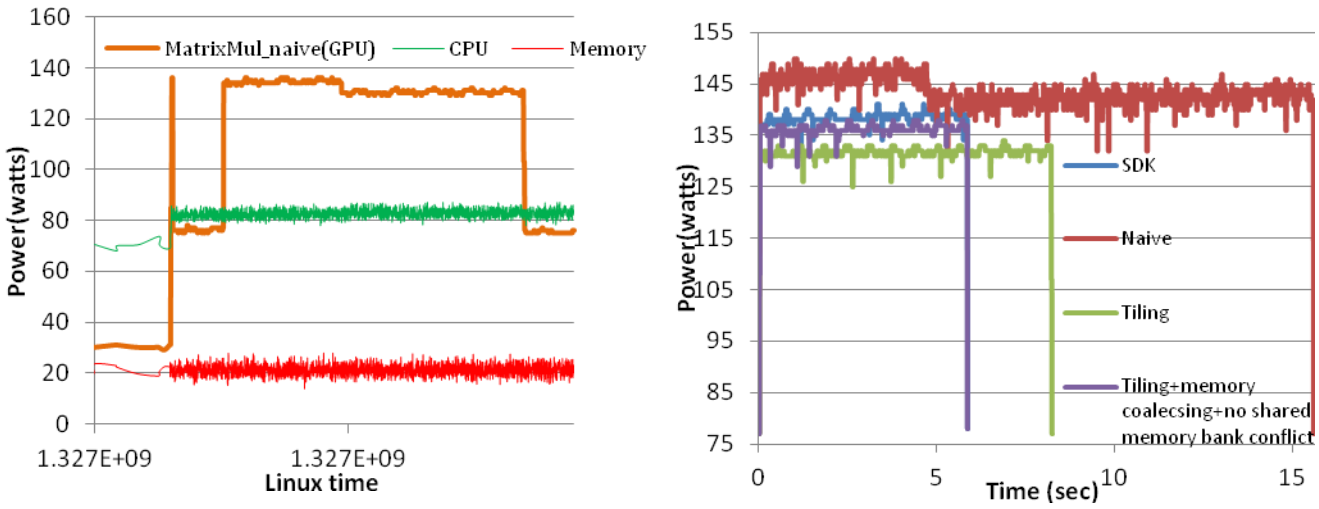


Figure 1.5: (a) Power profiling of the naïve implementation of Matrix Multiplication running on both GPU and multi-core CPU devices. (b) Power profiling of three different implementations of Matrix Multiplication running on GPU. Both (a) and (b) use NVIDIA C2075 as running GPU device. In 1.5 (b), Tiling + Memory optimization (purple line) finishes first, followed by SDK, Tiling and then naïve. Naïve also consumes the highest power among the four.

the CPU on average and 27 watts more than the CPU and main memory combined. The peak power for NVIDIA C2075 is 250 Watts or twice of the peak power for the host CPUs. This indicates that performance gains from utilizing GPUs may not offset the large increase in power consumption; users need to evaluate the power efficiency (e.g. performance/watt) of an individual application before selecting the GPU as the primary device to launch compute kernels.

Fig. 1.5(b) shows the power consumption of four different implementations of MM running with the same workload and it suggests that intensive global memory access and shared memory bank conflicts can affect the GPU power consumption. But what are the other influential factors for GPU power consumption and how do they impact the energy efficiency of GPU devices? In order to answer these questions, we need to quantitatively identify the performance and power bottlenecks and their causes, so that we can predict the benefits of potential program optimizations and architectural improvements [50].

1.2 Limitations of Past Approaches

At both system and architecture level, there have been studies on how to apply various hardware and software strategies to reduce power consumption without sacrificing performance. These techniques include DVS and DVFS on CPU cores, power-aware cache and memory management, optimized storage-bound I/O accesses, traffic-aware and power-efficient network interfaces, concurrency throttling and thread-core mapping, dynamic task scheduling and power-aware process migration, thermal-aware scheduling, and low-power hardware design. The focus in previous work has been

developing a controller that uses observational data and (in later techniques) predictive data to schedule power states and balance performance. Fig. 1.6 depicts the types of controllers used in these techniques to build sophisticated power management software.

A key limitation of past approaches is a lack of interactive power-performance strategies allowing users to quantitatively bound the effects of power management on the performance of their applications and systems. Existing controllers and predictors use policies fixed by a knowledgeable user to opportunistically save energy and minimize performance impact. While the qualitative effects are often good and the aggressiveness of a controller can be tuned to try to save more or less energy, the quantitative effects of tuning and setting opportunistic policies on performance and power are unknown. In other words, the controller will save energy and minimize performance loss in many

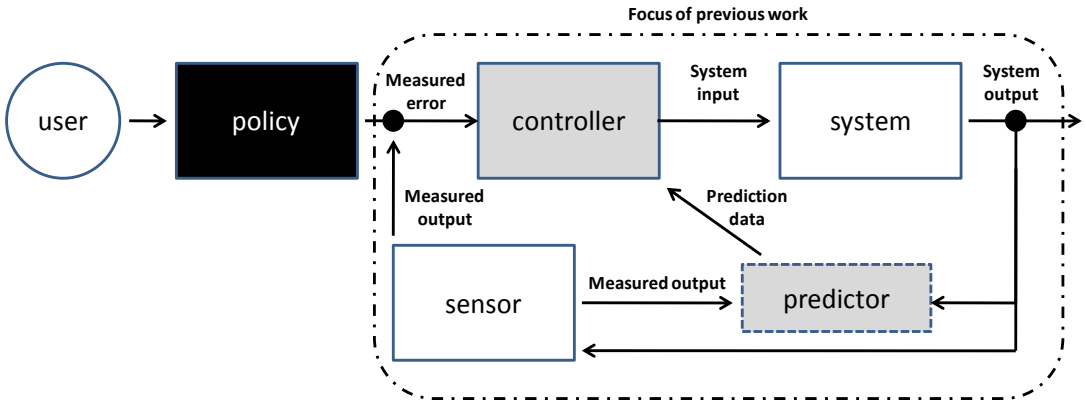


Figure 1.6: Past and current approaches to power management in high-performance systems.

cases but we have little understanding of the quantitative effects of controller tuning. This makes setting power-performance policies a manual trial and error process for domain experts and a black art for practitioners. To improve upon past approaches to high-

performance power management, we need to quantitatively understand the effects of power and performance at scale.

1.3 Research Objectives and Components

The objective of the proposed research is to develop detailed, sophisticated and accurate system-level iso-energy-efficiency models [4, 30, 51] to analyze, evaluate and predict energy-performance of parallel applications with various execution patterns running on large scale power-aware multi-core and accelerators-based clusters. Our analytical models and their related runtime tools can help users explore the effects of machine and application dependent characteristics on system energy efficiency and

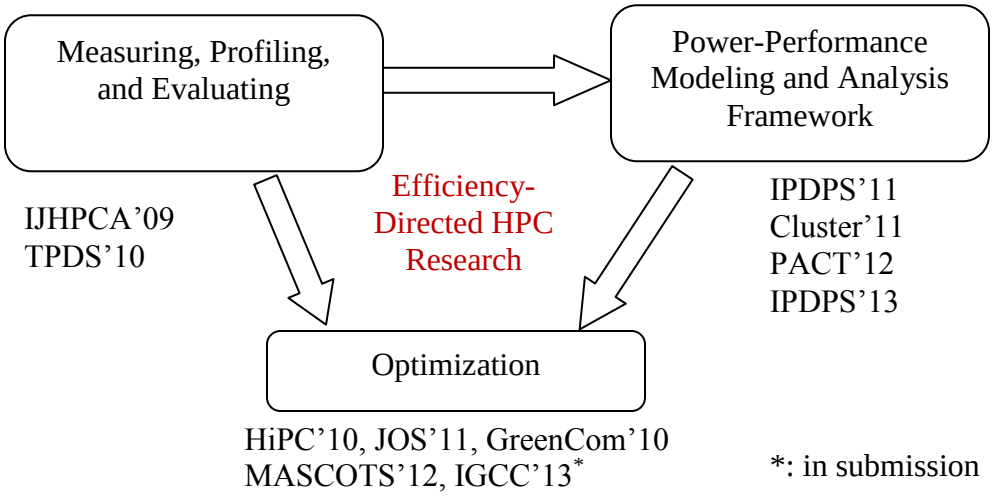


Figure 1.7: Research components of this thesis and the related publications for each category.

isolate efficient ways to scale essential system parameters (e.g. processor count, CPU power/frequency, workload size and network bandwidth for multi-core systems; grid size, number of threads per block, number of kernels, and problem size for GPGPU-based clusters) to balance energy use and performance.

My thesis research is composed of three major components: 1) profiling, measuring and evaluating the power behaviors of various parallel applications to identify underlying correlations between performance/energy and execution/workload patterns; 2) developing a power-performance modeling and analysis framework for current HPC applications; and 3) power-performance optimizations based on (1) and (2). All three components focus on improving HPC efficiency and improving our understanding of the relationship between power and performance? Figure 1.7 shows these components and my respective work under each component.

1.4 Research Contributions

In summary, this thesis provides methodologies and techniques to enable HPC users to improve system-wide efficiency as well as identify potential energy/performance bottlenecks at scale.

- I have applied our PowerPack toolkit [52, 53] to conduct detailed studies of the energy efficiency of various mainstream parallel benchmarks (including the NAS Parallel Benchmarka [54], the HPC Challenge Benchmarks [55], etc.), which are crucial to understanding the energy boundaries and trends of HPC systems. I have leveraged the methodology of profiling, measuring and evaluating an individual

application's unique power profiles by function and component in a high performance configuration [52, 53]. I also discovered that an applications' memory access rate and locality strongly correlate to their power consumption. Higher memory access locality is indicative of higher power consumption. Through correlating applications' memory access patterns to their power consumption, I was able to substantially optimize their performance and power efficiency at scale.

- I proposed a system-level *iso-energy-efficiency* model [4, 30] to analyze, evaluate and predict energy-performance of data intensive parallel applications with various execution patterns running on large scale power-aware clusters. My analytical model can help users explore the effects of machine and application dependent characteristics on system energy efficiency and isolate efficient ways to scale system parameters to balance energy use and performance. By applying the model and its related software framework to various parallel applications (i.e. NPB benchmarks, High Performance LINPACK [56], Dense Matrix Multiplication, etc.), my results indicate that the model accurately predicts total system energy consumption within 5% error on average. I also demonstrated how to effectively use our model framework for various application contexts and in scalability decision-making.
- Emergent heterogeneous systems must be optimized for both power and performance at exascale. Massive parallelism combined with complex memory hierarchies form a barrier to efficient application and architecture design. In order to address these challenges, I combined GPU hardware performance counter data with machine learning and advanced analytics to model power-performance efficiency for modern GPU-based systems [57, 58]. My performance counter based approach is simpler

than previous approaches and does not require detailed understanding of the underlying architecture. The resulting model is accurate for predicting power (within 2.1%) and performance (within 6.7%) for application kernels on modern GPUs. Mymodel can also identify power-performance bottlenecks and their root causes for various complex computation and memory access patterns (e.g. global, shared, texture). I also applied and extended the *iso-energy-efficiency* model to address heterogeneous HPC systems (e.g. GPGPUs and multi-core).

- Leveraging the concepts and infrastructure of our model-based framework, I designed and implemented PASCoL [59, 60], a high performance power-aware one-sided communication library using Aggregate Remote Memory Copy Interface (ARMCI), the communication runtime systems of Global Arrays [61]. For various communication primitives provided by ARMCI, we studied the impact of Dynamic Voltage/Frequency Scaling (DVFS) and a combination of interrupt (blocking)/polling based mechanisms provided by most modern interconnects. I implement our design and evaluate it with synthetic benchmarks using an InfiniBand Cluster. My results indicate that PASCoL can achieve significant reduction in energy consumed per byte transfer without additional penalty for various one-sided communication primitives and various message sizes and data transfer patterns.

1.5 Organization of This Dissertation

The organization for the rest of this thesis is as follows. Chapter 2 discusses the background and literature survey. Chapter 3 presents the first research component: measuring, profiling and evaluation techniques for power/performance of large scale

HPC systems. Chapters 4 and 5 describe our modeling and analysis framework: iso-energy-efficiency for both multicore- and accelerator-based HPC systems. Chapter 6 covers our runtime energy optimization techniques. We summarize our conclusions and future work in Chapter 7.

Chapter 2

Background and Literature Survey

We begin this section with a general review of the terms, metrics, and models related to this thesis for performance/energy efficiency research. Then, we will give a thorough discussion of other related work.

2.1 HPC Related Performance and Power-Aware Metrics and Models

In this section, we will show several important metrics and models which serve as the foundation for understanding our research methodologies described in the next chapters. A more thorough literature survey is presented in the subsection 2.2.

2.1.1 Power-Centric Metrics and Models

Various power-management techniques have been implemented to save energy. Dynamic Voltage and Frequency Scaling (DVFS) is the most commonly used dynamic power-management technique that allows the system to explicitly tradeoff performance for energy savings, by providing a range of voltage and frequency operating points [62]. For instance, DVFS has been widely applied to CPU [34-36, 63-65] and memory [43, 66] (simulation or emulation stage) to reduce power cost. Systems with higher/lower frequency levels have better/worse performance, however, at the same time, consume more/less power. Also, increasing/decreasing the supply voltage on a fixed system configuration has impacts on both operating frequency and power consumption. In CMOS technology [67, 68], power consumption can be divided into two parts: dynamic power and static power, shown in the following equations:

$$Power = P_{dynamic} + P_{static} \tag{2.1}$$

$$P_{dynamic} = ACV_{supply}^{2} f \tag{2.2}$$

$$Power \propto V_{supply}^{2} f$$

_/ 〖 __() 〗 ^ _D_Dd_____ ◌oA22_____ _ D_Dd_____ÿ22_____ _()

$$delay \propto V_{supply} / (V_{supply} - V_{sub_{th}})^{\alpha} \tag{2.4}$$

Dynamic power is the switching overhead in transistors and is determined by runtime events [69]. Static power is primarily determined by circuit technology, operating temperature, and chip layout. V_{supply} is the supply voltage and $V_{sub_{th}}$ is the sub-threshold voltage; A is the gate activity factor, C is the total capacitance and f is the operating frequency. In equation (2.1), $P_{dynamic}$ dominates over 70% of the total power.

Therefore, if we ignore the effect of static power in (2.1), power will be approximately proportional to operating frequency and supply voltage squared, shown in equation (2.3). Also equation (2.4) shows that reducing voltage will result in performance degradation as the clock frequency needs to be decreased to account for the increased circuit delay. Consequently, DVFS techniques need to make sure that the performance needs continue to be met as voltage and frequency are scaled. We actually use equation (2.1) to (2.4) to model CPU power in our global energy prediction function demonstrated in Section 5.

Based on this technique, substantial research has been done on power management at the hardware and architectural levels for microprocessor-based systems [31-33, 70, 71], and system level for global energy reduction. Extensive literature survey on this subject will be conducted in Section 3. Here we demonstrate two of the most commonly used power management (PM) algorithms on processor architecture level using DVFS. They are the MaxBIPS algorithm from [71] which uses a discrete set of possible voltage and frequency pairs, and a continuous algorithm [70] based on non-linear programming which finds the best possible voltage and frequency pair within a continuous range of values. Both of these two algorithms can be applied on a per-core basis (i.e., each core is allowed a unique voltage / frequency setting) and chip-wide (i.e., all cores operate at the same voltage / frequency setting). Unlike many system level approaches [34, 35] which consider all the system components when making frequency switching decisions (e.g., their workload demand changes due to execution patterns), these two techniques are mainly used to address the multi-core CPU power management under relatively stable workload characteristics. They are the fundamental techniques for chip-level power-aware computing and served as examples of applying DVFS on CPUs

to maximize performance under power budget.

MaxBIPS [71] assumes a set of discrete power modes ($V_{supply} - frequency$ pairs) for each core which the PM can control individually. The goal is to maximize the overall chip performance, as measured by the total number of completed instructions (I) by all cores per time period, under a given power budget. When a given core switches from power mode A ($V_{supply-A}, f_A$) in observation window N to power mode B ($V_{supply-B}, f_B$) in observation window $N+1$, the future performance and power is predictable using simple formulas, as shown in Table 1. Since both number of power modes and cores constrain the searching performance for optimal solution, MaxBIPS is mostly used for a simple runtime search. To compare with the discrete MaxBIPS solution, the Continuous Power Modes (CPM) algorithm [70] has been proposed using non-linear programming to model the same DVFS problem with continuous power modes. Under this algorithm, cores can run at any frequency and voltage within predefined upper and lower bounds. The objective function of the CPM is to maximize the overall chip performance as measured by the sum of the predicted performance for the next interval as long as the sum of the predicted power (dynamic + static) do not exceed the power budget. Work [70] states that the CPM algorithm is too complex for a hardware implementation, but it is useful in a design exploration methodology since it produces optimal results which allow us to evaluate heuristic algorithms with practical implementations.

Memory power management using DVFS is also proposed by works [43, 66]. However, current hardware cannot dynamically scale memory frequency and voltage. For now, the best users can do is to emulate the performance effects of frequency scaling by changing timing parameters of the DRAM [66] . We expect architecture in near future

will provide such memory power management interface.

2.1.2 Performance-Centric Metrics and Models

Performance modeling is vital for power-aware computing because its close relationship with energy makes it impossible to discuss the two separately. Accurate modeling of

Table 2.1: Simple illustration of MaxBIPS algorithm.

Observation Window	N	N+1
Mode	(v, f)	(v', f')
Performance	I	$I * (f'/f)$
Dynamic Power	$P_{dynamic}$	$P_{dynamic} * (v'/v)^2 * (f'/f)^2$
Static Power	P_{static}	$P_{static} * (v'/v)^3$ (estimation)

performance will greatly improve the accuracy of modeling total system energy consumption based on equation (2.11). Also, for the architectures which have not offered system-level power management techniques such as DVFS (e.g. most of the current GPGPU chips), improving performance may gain better total energy consumption. In this subsection, we will introduce: (a) general performance models for both sequential [4, 21] and parallel execution [4]; (b) performance isoefficiency relation [72, 73]; and (c) power-aware speedup [74]. Methods listed in this section are the ones closely related to our proposed research in the Chapter 3 to 6. More detailed related work on the subject of performance modeling will be discussed in the subsection 2.2.

a) Sequential time modeling based on workload decomposition of single processor system:

At the system level, works [4, 21, 74] provide a sequential execution time model for describing on-chip/off-chip workload. The model assumes in-order processors and only main memory as off-chip target. The basic model is described as follow:

$$W^{on} = N * CPI_{on}^{avg} = N * \left(CPI_{0} + CPI_{branch-miss}^{avg} + CPI_{stall_{onchip}}^{avg}\right) \quad (2.5)$$

$$W^{OFF} = N * CPI_{stall_{offchip}}^{avg} = M * CPI_{off}^{avg} \quad (2.6)$$

$$T = T^{ON} + T^{OFF} = \frac{N * CPI_{on}^{avg}}{f^{cpu}} + \frac{M * CPI_{off}^{avg}}{f^{EXT}} \quad (2.7)$$

Where W^{on} is defined as on-chip workload, which is the number of CPU clock cycles required to perform the set of on-chip instructions. They are executed inside the CPU only. N is the number of instructions, CPI_{0} is the ideal CPI which is 1 for a single-issue general-purpose microprocessor, $CPI_{branch-miss}^{avg}$ denotes the number of CPU clock cycles due to branch misprediction overhead, and $CPI_{stall_{onchip}}^{avg}$ and $CPI_{stall_{offchip}}^{avg}$ denote the numbers of CPU clock cycles due to on-chip stalls and off-chip stalls, respectively. M is number of the main memory accesses. T^{ON} and T^{OFF} are on and off – chip execution time. f^{cpu} is on-chip frequency and f^{EXT} is external memory clock frequency.

The major contribution of this model is to separate on and off-chip effects impacted by CPU and memory frequency. For instance, using equation (2.6) combined with performance counters, users can easily calculate the expensive main memory access

time, which has been proved to be two orders of magnitude greater than the instruction execution time. However, this model ignored two factors for modeling performance on a single core system: 1) for the out-of-order execution, the performance model requires certain kind of overlapping or corrector factor for the execution time of individual system component. Otherwise, equation (2.7) may not be accurate; and 2) this model neglects other major system components of the modern architecture, which may very likely be the bottleneck for the overall performance. For instance, the I/O access time, which includes the accesses of network and all kinds of local storage devices, has been a major bottleneck for HPC system performance [75-82]. The drawbacks of (1) and (2) will be addressed in our proposed energy model presented in Chapter 4.

b) General parallel execution time decomposition:

Equation (2.8) demonstrates a simple scenario when application running on n processors. Parallel execution time $T_{parallel}$ is composed of parallel computation time $T_{Para-Computation}$, parallel main memory access time $T_{Para-Memory-access}$, parallel I/O time $T_{Para-IO}$, and parallel overhead on each processor $T_{overhead}$. α is the corrector factor as we discussed in the previous section.

$$T_1 parallel = \alpha\,(T_1(Para - Computation) + T_1(Para - Memory - access) + T_1(Para - IO) + T_1 overhead) \quad (2.8)$$

The component $T_{overhead}$ consists of several factors, including communication overhead for data transmission between nodes, synchronization overhead due to load

imbalance and serialization, and also the unexpected extra computation overhead caused by parallel scheme and task assignment [4]. Parallel overhead is one of the most essential design constraints for HPC applications and systems, and minimizing it may dramatically improve system energy efficiency. We will demonstrate how to derive each component in equation (2.8) using various system tools in Chapter 4.

c) Performance isoefficiency:

Grama et al [72] and Michael J. Quinn [73] proposed the performance isoefficiency metric to relate problem size with the number of processors required to maintain speedup. Under Amdahl effect [83, 84], speedup (and hence efficiency) is typically an increasing function of the problem size, because the communication complexity is usually lower than computational complexity. So in order to main the same level of efficiency when processors are added, we can increase the problem size. This performance scalability focused metric is the inspiration for our iso-energy-efficiency model on parallel multi-core systems [4, 30] discussed in Section 5. It is defined as follows [73]:

Suppose a parallel system exhibits efficiency $\varepsilon(n,p)$. Problem size is denoted as n . $T(n,1)$ is the sequential execution time. $T_o(n,p)$ is the total amount of time spent by all processes doing work not done by sequential algorithm. One component of this time is the time $p-1$ processes spend executing inherently sequential code. The other part of this time is the time all p processes spend performing inter-processor communications and redundant computations. So in order to maintain the same level of efficiency as number of processors increases, problem size n must be increased so that the following

inequality is satisfied:

$$T(n,1) \geq CT_o(n,p) \tag{2.9}$$

c) Power-aware speedup:

Power-aware speedup, proposed Ge et al [74] , is one of the most related works to our proposed research described in Section 5. The basic idea of this analytical model is to take into accounts the effects of both parallelism and power-aware techniques on ***speedup*** based on the idea of decomposing the workload with ON-/OFF- chip characteristics (similar to what we discussed previously in Chapter 2.1.2(a)) and degree of parallelism (DOP). The model itself involves both ON-/-OFF chip frequency and workload as major parameters for predicting performance speedup. The general model is defined as follows:

$$S_N(\omega,f) = (T_1(\omega,f))/(T_N(\omega,f)) = [\omega^{ON} \cdot [CPI]^{ON}/(f_0^{ON}) + \omega^{OFF} [CPI]^{OFF}/f^{OFF}]/([\sum_i (i=1)^m [((\omega_i^{ON})/i \cdot [CPI]^{ON}/f^{ON} + (\omega_i^{OFF})/i \cdot [CPI]^{OFF}/f^{OFF}) + (T(\omega_i^{PO ON}, f^{ON}) + T(\omega \tag{2.10}$$

Where $T_1(\omega, f)$ and $T_N(\omega, f)$ are sequential and parallel execution time, respectively. ω^{ON} and ω^{OFF} are on- and off-chip workload. m is the total number of processors in the system. $T(\omega_{PO}^{ON}, f^{ON})$ and $T(\omega_{PO}^{OFF}, f^{OFF})$ are execution time for calculating parallel overhead for on- and off- chip. ω_{PO} is the parallel overhead workload due to communication, synchronization, etc. Definitions of the other parameters in (2.10) are as same as those in Chapter 2.1.2 (a).

Power-aware speedup model shown in equation (2.10) can be converted into simpler forms depending on the workload characteristics, such as computation intensive

workload with minimum communication (e.g. embarrassingly parallel case [54]) and communication intensive workload (e.g. FT in NPB benchmark suites [54]). This feature will benefit users with only significant and necessary parameters for specific application and a clearer picture of how to scale these parameters such as frequency and DOP in order to achieve better speedup. Similar to the [21] in Chapter 2.1.1, current power-aware speedup model only assumes in-order execution for simplicity.

2.1.3 Estimation of energy consumption

Metrics in equation (2. 11) employs average power and delay (or execution time). While power P_i ($i \epsilon$ *set of all nodes* Ω) describes the rate of energy consumption at a discrete point in time on node i , energy E specifies the total number of joules in the time interval ($t_1 1$, t_2) , as a sum of products of average power $\overline{P_i}(i \epsilon$ *set of all nodes* $\Omega)$ and delay ($Delay = t_2 - t_1$) over all the computing nodes:

$$E = \sum_{i(i \epsilon \Omega)}^{\Omega} \int_{t_1}^{t_2} P_i(t) dt = \sum_{i(i \epsilon \Omega)}^{\Omega} \overline{P_i} \times Delay \qquad (2.11)$$

We can conclude from equation (2.11) that strategies for reducing average power, delay (execution time) or both should be applied in the power-aware design in order to achieve minimum total energy cost. Maximally and more efficiently parallelizing the application with less memory communication and synchronization overhead, better deploying data in local and global shared memory and optimizing communication will help boost the performance and reduce the delay. For average power consumption $\overline{P_i}(i \epsilon$ *set of all nodes* $\Omega)$, we can use strategies such as: (1) DVFS to scale down CPU

frequency, reducing CPU power during CPU non-intensive phases [34, 85, 86]; (2) concurrency throttling [38] to decrease the number of cores based on previous training experience and bind specific threads to corresponding cores in order to reduce both CPU and memory power (memory power reduced due to less access contention); and (3) smart power management schemes for major system components such as CPU, memory, hard disk and power supply. For parallel computing, scaling up the system size also boosts the performance and reduces delay. At the same time, more overhead and power cost occur. Automatic, runtime and intelligent power-aware scheduling systems are highly demanded by the HPC and the growing data center market [87, 88].

We can also use (2.11) to derive a simple energy model specifically for GPGPU-based cluster. Let's assume we have a set of hybrid nodes with multi-core and multi-GPUs on it, denoted as Ω . Also on each of the node, we have a set of GPUs, denoted as Ψ. We also have $i \in \Psi$ and $k \in \Omega$. So we can estimate the energy consumption of this hybrid system as:

$$\forall i,k \ , \ i \in \Psi \ , \ k \in \Omega \ , \ E_{CPU-GPU} = \sum_{k \in \Omega} \sum_{i \in \Psi} \left[\left(\overline{P_G}^{k_i} + P_{C_{idle}}^{k} \right) * t_{GPU} + E_{transfer} \right] + E_{para-overhead} \quad (2.12)$$

Where $\overline{P_G}^{k_i}$ is the average power consumption of i th GPU device on node k . $P_{C_{idle}}^{k}$ is denoted as the idle power consumption of the multi-core CPUs on node k . We make the assumption that the kernels running on this system only contain GPU workload. t_{GPU} is the execution time for this GPU kernel. $E_{transfer}$ is the total energy consumption for data transformation time between host memory and GPU global memory after GPU

kernel launches and before it finishes. $E_{para-overhead}$ is the energy consumption for the parallel overhead. This simple model illustrates some influential factors for global energy consumption of such CPU-GPU hybrid parallel systems such as $\overline{P_G}^{k_i}$, $E_{transfer}$ and $E_{para-overhead}$. Detailed GPU cluster energy modeling will be shown in Chapter 5.

2.1.4 Commonly used performance-energy tradeoff metrics

For parallel systems, energy scaling is dependent on performance efficiency or speedup and there exist tradeoffs between power, energy and performance. Searching for the best "operating point" based on users' demand is sometimes non-trivial. The energy-delay product ED^{α} ($\alpha \geq 1$ *is a real number*) evaluates energy-power-performance tradeoffs and users' priorities [52, 85]. The smaller the value of ED^{α}, the better energy-performance efficiency the system configuration can achieve for the application. For performance-at-any-cost systems, $E = \mathbf{1}$ and minimizing *D* is the best solution for finding the optimal operating point. For energy- or power-constrained systems, we need to minimize the energy *E* or power *P* and ignore the *D* effect in order to get the best energy number. However, the strategies above are two extremes and not optimal for current systems. ED^{α} defines a scenario for evaluating energy-performance efficiency and their tradeoffs for different types of systems. EDP (energy product when $\alpha = \mathbf{1}$) is primarily used for evaluating low power portable systems such as smart phones, *PDAs*, and laptops, where battery life is the major design concern for these devices and energy weighs more than performance [89]. *ED2P* (when $\alpha = \mathbf{2}$) emphasizes both performance and energy [25]. *ED3P* (when $\alpha = 3$) emphasizes delay *D* more than energy and is

appropriate for evaluating high performance systems where performance is the most important design constraint but energy efficiency is needed [25]. For instance, running the highly computational chemistry application *NWCHEM* [90] from Pacific Northwest National Lab requires more than 2,000 nodes and using *ED3P* can help choose the system configurations that favor the best performance while keeping energy costs down.

In addition to the *EDP* products, **Performance/Watt** becomes a very popular metric to measure system energy-efficiency for GPGPU devices [91, 92]. It is because high performance is still the primary target for GPU designers and programmers. More performance-energy tradeoff metrics will be discussed in the next subsection (related work).

2.2 Other Related Work

In this subsection, for the purpose of supporting the rest chapters and being cohesive, I summarize the related work into two big categories: 1) power-aware technologies on various subjects including multi-core processors, main memory, I/O devices, accelerators, HPC Servers and Cloud; (2) measuring, profiling and modeling of power, performance and overall energy for both multi-core and accelerator based systems.

2.2.1 Power-aware and energy conservation approaches

a) Power-aware techniques for Multi-core processor:

In power-aware computing, one of the most common approaches to reduce system power consumption is to apply Dynamic Voltage Frequency Scaling (DVFS) to CPU

cores based on workload demand (trying to maximize energy proportionality [76]), since traditionally CPU consumes the most power for common architectures [52, 53]:

DVFS for microprocessor-based chip multiprocessors (CMP)*:* As the power / performance curve for microprocessor-based systems is flattening across processor generations [93], techniques for managing power and performance are needed both at the low-end battery-constrained embedded system and high-end datacenters. Nowka et al. [31] uses software control to make 36 mm2 processor go to either a low-leakage sleep state or a state-preserving deep-sleep state to minimize standby power consumption. Magklis et al.[32] present Multiple Clock Domain (MCD) approach to alleviate the bottlenecks of fully synchronous systems and combined with on-chip voltage scaling technology to reduce chip energy consumption. McGowen et al. [33] describe an embedded feedback and control system on a 90-nm Itanium family processor "Montecito" that maximizes performance while staying within a target power and temperature (PT) envelope. Isci et al. [71] propose a chip-level, per-core DVFS based, global monitoring and controlling system for CMP systems under power and temperature budget constraint. To further optimize the power management for MCD architectures, Mishra et al. [94] propose two-tier power management architecture where, the first-tier allocates a target budget to each local island in the CMP and the second-tier controls the power at the target budget. The goal of this work is to maximally reduce performance degradation while still effectively control chip power under budget. Bergamaschi et al. [70] apply two PM algorithms on CMP simulation system SLATE (System-Level Analysis Tool for Early Exploration) [95] in order to study how DVFS impacts CMP

level power management on both per core and per chip level.

System-level DVFS on multi-core based parallel systems: In addition to chip level power management using DVFS, power and energy consumption of large scale HPC systems have become a critical design constraint. Most of the proposed energy efficiency mechanisms (online or offline) based on DVFS will observe the system workload or behavior and then adjust the system's operating point periodically based on certain prediction [34-36, 65, 96], moving on the performance/power curve to achieve the best efficiency. Generally speaking, there are three types of bottlenecks DVFS strategies can be used to attack: (1) inter-node bottlenecks, where the CPU waits for data transmission from foreign nodes, (2) memory bottlenecks, where the CPU waits for data from the memory system, and (3) communication bottlenecks, where power/energy can be saved by applying DVFS to non-CPU intensive operations. For instance, in Fig 3.6 PTRANS and MPI_FFT [53] show those CPU power "valleys" due to non-computation intensive phases to which DVFS could be applied.

Some researchers have discussed the tradeoff between performance and energy consumption for scientific applications [63, 85, 86]. Ge et al.[85] have studied the DVS scheduling on a power-aware cluster and compare several DVS strategies to explore the possibility of saving system-wide energy without losing performance. Springer et al. [63] and Freeh et al [86] studied the energy-performance tradeoff for power-aware clusters using NAS parallel benchmark [54] through performance modeling and prediction. They use this offline analytical approach to manually choose the number of nodes and processor frequencies to minimize execution time under maximum allowed power

budget. These preliminary studies have provided researchers some ground rules for further designing energy-efficient DVFS scheduling techniques for large scale clusters.

Offline trace-based DVFS techniques and online workload demand based techniques have been proposed to maximize energy saving without sacrificing performance. Freeh et al [64] proposed to use a heuristic to attack intra-node memory bottlenecks by choosing frequencies for execution phases based on previous experience. This approach requires offline performance profiling and manual insertion of DVFS calls to the execution phases. Another offline approach [97] proposed by Hsu et al. focuses on using compiler instrumentation to profile and insert DVFS scheduling functions for sequential codes. These offline approaches above make setting power-performance policies a manual trial and error process for domain experts and a black art for practitioners. To improve the usability and prediction accuracy, several runtime DVFS approaches have been proposed. Lim et al. [65] implement a run-time scheduler which intercepts MPI calls to identify communication-bound phases in MPI programs. "Adagio" [35], using a critical path based online algorithm, applies simple predictions of execution times for program regions based on prior executions of the regions. It assumes execution slowdown is proportional to the frequency change. Ge et al. [34] propose "CPU MISER", a performance driven, system-wide, runtime DVFS scheduler for high performance computing. It achieves performance counters at runtime and feeds them into the RELAX model in order to predict what is the best frequency choice for next time interval. The approach is based on system workload demand and runs as a background demon on each parallel node. Spiliopoulos et al [36] propose a Continuously Adaptive DVFS framework called "Green Governors" in Linux systems running on Intel i7 and

AMD Phenom II processors. Unlike the on-chip workload driven performance prediction models in the previous works, the underlying methodology of Green Governors is based on a simple first-order processor performance model in which frequency scaling is expressed as a change (in cycles) of the main memory latency. Combining with exploiting slack in memory-bound programs, Green Governors can improve energy efficiency when the CPU is not in slack. Hsu and Feng [96] propose the β -adaption algorithm to automatically adapt the voltage and frequency at run-time in order to gain better energy savings. The core of all above approaches is to find an accurate performance prediction model to drive the automation process in order to avoid severe performance degradation caused by runtime mispredictions. In work [98], Rountree et al. provide a wide-range survey covering many counters which have been used in previous performance modeling works, and then compare both prediction error and standard deviation of the proposed new performance counter "Leading Loads" with the current best practice linear regression model. They demonstrate "Leading Loads" has an order of magnitude improvement over the best existing approaches.

There also have been some works focusing on addressing energy reduction for load-imbalanced parallel applications executing on parallel systems using DVFS. Rountree et al. [99] developed an offline method that uses linear programming to estimate the maximum energy saving possible for MPI programs based on critical path analysis. The basic idea is to let cores on the critical path run as fast as possible while exploring energy saving opportunities on the rest of the cores by slowing them down proportionally. The output schedule generated through the linear programming will direct the applications to run more energy efficiently without losing performance. Kappiah et al.

[37] monitor the synchronization points in MPI in order to determine which nodes are the non-bottleneck at each communication point and then run them slower to save energy without slowing down the nodes on the critical path.

DVFS critical phase detection: For many of the DVFS approaches demonstrated above, users need to first identify the DVFS critical regions. Several approaches discuss how to manually or automatically detect the critical phases to which power-aware approaches can be applied. Freeh et al. [64, 86] first manually divide programs into phases using MPI hijack, PMPI mechanisms, and then ad post processing hooks. A series of experiments are then executed, with each phase assigned a prescribed frequency. Balasubramonian et al.[100] use a *conditional branch counter* to detect program phase changes. The counter keeps track of the number of *dynamic* conditional branches executed over a fixed execution interval (measured in terms of the dynamic instruction count). Phase changes are detected when the difference in branch counts of consecutive intervals exceeds a threshold. Ashutosh S. Dhodapkar and James E. Smith [101] define a program phase to be the instruction working set of the program, i.e. the set of instructions touched in a fixed interval of time. Program phase changes are detected by comparing consecutive instruction working sets using a similarity metric called the *relative working set distance*. Work [102] proposes an algorithm to infer MPI communication patterns by looking at both local and global sequences to detect MPI event patterns. Hsu and Kremer [103] explore a compile-time technique to detect the critical phases for frequency scaling. However, the compiler approach for auto detection doesn't have high accuracy because it requires knowing the exact execution duration of the regions. Bertran et al [104] propose

performance monitoring counter (PMC) based Decomposable and Responsive Power Models for Multicore Processors and the responsiveness validation shows almost 100% accuracy in detecting phase variations above 0.5 watts.

Dynamic concurrency throttling and its related techniques on power reduction: Dynamic concurrency throttling (DCT) for multi-core systems has become another potential approach to reduce system energy at runtime. The basic idea to save energy in case is that: 1) not any workload requires the full set of cores in the node; and 2) there is a performance difference among different thread-core mappings (e.g. cores share the same socket will potentially encounter more memory contention than if two cores are on different sockets). There have been several DCT related approaches proposed for saving runtime energy consumption. Curtis-Maury et al. [38] combine DVFS with concurrency throttling techniques on multi-core systems at runtime to explore the right combination of "switches" (frequency level and number of cores being utilized) to reduce total energy consumption. Li et al. [67] apply concurrency throttling and DVFS on single chip multiprocessors to maintain performance while reducing power consumption. Unlike Curtis' approach, this approach is based on simulation and ignores the overhead when switching configuration. Li et al. [39] propose algorithms using DCT and DVFS to leverage energy-saving opportunities without performance loss for hybrid MPI/OpenMP applications. Su et al. [40] propose DCT based critical path-based thread placement for NUMA systems to avoid performance degradation caused by remote memory access and memory resource contention.

b) Power-aware approaches for memory devices:

Traditionally, multicore processors dominate the entire server energy consumption [52, 53] and attract a lot of attention in the HPC field. Since then, various hardware and software techniques have been developed and applied to multicore processors to improve their power efficiency as well as maximizing the system performance, as we discussed in the previous section. With the decreasing power share in HPC server node for power-efficient multicores, the memory power consumption system share is increasing dramatically due to growing high demand [105-107] for memory capacity and bandwidth. From a recent report on warehouse-scale datacenter design [105], the main memory system accounts for almost 40% of total datacenter power consumption, compared to the old 15% to 20% from several HPC systems [52, 53]. For the GPGPU-based system, memory system (including global memory, shared/local memory, texture memory, and constant memory) also accounts for a major share of the total power budget [49], which is comparable to computation device. In this subsection, we are going to give a literature survey covering both hardware and software level approaches of memory power conservation techniques. The substantial amounts of works has been focused on memory system for multicores and specifically on DRAM and DIMMs [43].

1) Hardware-level approaches for power-aware memory

DRAM low power states: Some works have studied the tradeoffs inherent in DRAM low power states (DRAM idle states or memory sleep states). During these idle periods, power consumption of DRAM can be reduced. However, memory access is not

permitted during the DRAM low power states. Research has been done on how to manage the memory system in order to increase the opportunity of DRAM idleness through layout transformations, batching, and scheduling [43]. Diniz et al. [44] propose algorithms to limit the power consumption by smartly turning off a subset of the memory devices in a system that can control each device individually. Fan et al. [45] give an analytical model for sleep-state benefit. Huang et al. [108] proposed a hardware-software combined approach integrating awareness of DRAM power states into the OS virtual memory system, allocating pages intelligently from different banks. Lebeck et al. [46] explored the interaction between memory sleep states and page placement. Li et al. [109] proposed several performance-guaranteeing control algorithms to bound performance losses resultant to fluctuations in workload memory accesses and improve the performance of power control algorithms proposed by Lebeck et al [46, 47, 109]. Aggarwal et al [110] presents methodologies to determine when speculative DRAM accesses are power-efficient. Work [111] proposed by Hur and Lin presents techniques to increase opportunity for DRAM power-down states. However, the challenge we have for above techniques is that modern DRAM devices have to be designed with rich chip-level power management techniques permitted in the older module, such as RDRAM [112]. Without these techniques, exploring these idleness states becomes very difficult with coarse power management granularity such as a rank of DRAM chips [43].

Memory throttling under peak power or temperature limit: Works belonging to this category tend to take power or temperature limits as the primary constraint regardless of performance degradation caused by memory access latency. “Memory throttling” is a basic technique used here to limit number of memory access requests in order to achieve

better dynamic power consumption. However, this technique suffers from long latency delays when DRAM is throttled and it is also less energy-proportional [76]. Felter et al. [113] proposed "power shifting", in which memory throttling is used to manage power consumption across the system (including the memory subsystem). Hanson and Rajimani [114] provide measured power and performance data with memory throttling on a commercial blade system, and discuss key issues for power management with memory throttling mechanisms. David et al. [115] proposed a power model called "RAPL" (Running Average Power Limit), in which memory power is explicitly modeled and throttled. Lin et al. use a temperature upper bound to either adjust amount of memory traffic [116] or clock-gating processors [117] when memory temperature is near the limit.

Ensemble-level techniques for memory energy reduction: There are several approaches proposed by using ensemble-level techniques to reduce larger scale server memory energy. Meisner et al. proposed PowerNap [118] which provides a full-system sleep state in lieu of finer-grain power management. But this approach only assumes workload with amount of idle time between requests so that serving requests faster will be more energy-efficient than entering low-power sleep state. It is not a "under workload demand" approach. Elnozahy et al. [119] proposed scaling the size of a server cluster according to load in order to improve energy proportionality and reduce memory idle time waste. This work is on higher server level and requires modifying parallel system organization.

DRAM reorganization and Rank Subsetting: Several proposals [120-122] have been presented recently on reducing number of memory chips or bits accessed at a time in order to achieve better memory energy efficiency. These approaches require either re-

architecting the DRAM [120] or DIMMs [121, 122]. Udipi et al. [120] proposed to reorganize and redesign the current memory DRAM by reducing number of chips accessed in order to fit under power constraint. Instead of re-architecting the DRAM, Multicore DIMM [121] and mini-rank [122] have been proposed to reduce power through DIMMs. Both proposals need chip select signals per rank subset [123]. The major difference between these two is that mini-rank has a demux per memory rank while Multicore DIMM has one per memory channel. so that mini-rank is less energy efficient [121]. All of the approaches above will benefit memory system with low *dynamic power*, however, they suffer from high re-architecting or re-design cost which may not be easily adopted by current existing commodity products [43].

Memory voltage and frequency scaling for SRAM and DRAM: Several research groups [43, 66, 124, 125] have studied the subject of memory voltage and frequency scaling for SRAM and DRAM. Puttaswamy et al. [124] examined voltage/frequency scaling on off-chip SRAM in an embedded system. This work doesn't provide dynamic frequency switching mechanism and it is on SRAM which has different performance and power characteristics than DRAM. Zheng et al. proposed Decoupled DIMMs [125] which can run the DRAM device at a lower frequency level than the memory channel. But it requires extra efforts to achieve this such as synchronization buffer between DRAM and memory channel. Also it does not have dynamic frequency scaling ability and only reduce DRAM power. Recently, two studies [43] [66] focused on memory **dynamic** voltage and frequency scaling through simulation [43] and real system emulation [66] have been proposed. Deng et al. proposed the OS driven and workload-directed memory voltage and frequency scaling technique "*MemScale*" [43] in

simulation to reduce background, register/PLL, and memory controller power consumptions. Based on a real system **emulation**, David et al. [66] proposed and evaluated the memory DVFS and its related control algorithm to increase frequency when bandwidth utilization crosses a predefined threshold, bounding the performance impact. Since current hardware does not support dynamically scale memory frequency and voltage [66], both of these approaches are still limited to the analytical modeling stage.

2) Software-level approaches for power-aware memory

Recently, a few studies have been exploring the software-level opportunities and control methods on how memory should be accessed and allocated in order to improve memory energy efficiency. In [126], Delaluze et al. describes how compiler analysis of workloads to direct memory model control can lead to better efficiency and they also proposed a OS scheduler-based DRAM energy management using power states in work [115]. Pandey et al in [127] examine memory accesses due to I/O DMA activity and proposed alignment and data-placement techniques to increase energy efficiency. Lebeck et al. [46, 128] explore the impact of static and dynamic memory controller policies with page allocation algorithms in order to dynamically switch memory to lower power states. Hung et al. [108, 129] integrate DRAM power awareness into the OS virtual memory system and smartly allocate pages from different banks. Also, they propose the “hot” and “cold” ranks concept based on workload demand and migrate frequently accessed pages to “hot” ranks while putting the rest in “cold” ranks into sleep states. Zhou et al. [130] propose a metric for analyzing page demand and applied it to direct system power management. In order to further emphasize the scalability of the memory power

management on parallel systems, Tolentino et al. [131, 132] propose "*Memory-Miser*", a transparent, performance-neutral, and scalable memory management infrastructure. It combines page allocation shaping, allocation prediction, and dynamic control based on runtime memory workload. It takes the advantage of the slack in memory demand and modern server's ability to turn on and off DIMMs at runtime to reduce memory power consumption. Authors claim that, for the multi-user workload, *Memory-Miser* can save up to 67.94% of memory energy with~~in~~ 1% performance degradation [132].

c) Power-aware approaches for I/O devices:

Along with the growing number of CPU and memory level power-aware approaches, emerging I/O level (including all kinds of local storage devices and network) energy conservation techniques have also been proposed due to the increasing importance of I/O component on both performance and power for current and future HPC systems.

At disk level, energy saving techniques such as multi-speed hard disks, turn-off and spin-down, and low disk power mode have been studied. For multi-speed hard disks approach, Gurumurth et al. [79] present DRPM to modulate disk speed (RPM) dynamically and use simulation method with various workloads and hardware parameters to prove its energy saving ability. Carrera et al [77] found multi-speed disks can save up to 23% of network server energy. Zhu [82, 133] et al. combine several techniques of multi-speed disks, data migration, and performance boosts to reduce energy consumption while meeting performance goals. Gniady and Hu used this technique to determine when to power down disks [134]. Other approaches [75, 78, 135-137] focus on detecting inactivity of disk after a certain period of time and then spinning it down for energy

saving. Weddle et al [138] built Power-Aware RAID (PARAID), which reduces energy use of commodity server-class disks without specialized hardware. PARAID uses a skewed striping pattern to adapt to the system load by varying the number of powered disks. By spinning disks down during light loads, PARAID can reduce power consumption while still meeting performance demands. However, PARAID doesn't address intra-disk reliability techniques. In order to improve the reliability issue, Wang et al [80] propose energy-reliability product (ERP) to study distribution of the disk idle periods between reliability and energy management tasks and coordinate both factors into the energy optimization strategy.

At network level, works [139, 140] propose adapting the power states of network interface to reduce network power consumption. Also, power-aware communication protocols have been designed and implemented. Works [141] [142] propose energy saving approaches using DVFS and CPU throttling for collective communication primitives. Liu et al. [143] have provided a detailed empirical study of the benefits of power efficiency of RDMA compared to the traditional communication protocols such as TCP/IP. However, this work does not provide guidance for higher level communication protocols for implementation. In Section 5, to address one-sided communication runtime systems, we have presented PASCoL [59, 60] , which provides a power efficient and high performance communication runtime system for one-sided primitives.

d) Power-aware approaches for GPGPU:

While performance acceleration with GPGPU devices has been widely explored with various applications, power-aware techniques for GPUs have been rarely studied.

The reason is that there has not been any system-level power-aware technique available to users on mainline commercial GPU chips. For instance, NVIDIA has not released how they schedule tasks running on Streaming Multiprocessors (SM) and DVFS for their GPUs is not enabled either. Other options to make GPU based clusters more energy efficient is to study how to scale system parameters such as grid size, number of threads per block, problem size and parallelism. Without an accurate system-level energy model which describes the interactions between those parameters, it is very difficult for suggesting the optimal settings for best system energy efficiency. This is also one of challenges we are facing for the current ongoing work on the energy efficiency of GPU based clusters. There have been only a few works [49, 91, 144] studying power modeling for GPGPUs with linear regression for specific GPU architectures. These works focus more on the statistical process of assuming the linear relationship between power and several counters without correlating performance and energy consumption with essential parameters. Also, their approaches were designed on the older generation of GPUs so do not provide insights on the new GPU hardware components such as texture and constant memory. Hong et al. [69] propose an architecture level analytical power model and then use performance/watts to predict optimal number of cores to run GPU applications. However, this requires a big amount of complicated architecture level parameters to support the model and without the explicit revealing of NVIDIA scheduling methodology and runtime performance counters support, this may only work for some applications and specific GPU test beds. Also, their model does not provide insights for how system level parameters correlate and how they impact system energy consumption or efficiency at scale.

e) Energy saving strategies for server farms and datacenters in a cloud:

In addition to large scale HPC systems, cloud computing is rapidly growing in importance as increasing numbers of enterprises and individuals are shifting their workloads to cloud service providers [88]. From a 2006 report [145], U.S datacenters were consuming 61.4 billion KWh per year which equals to the total energy consumption for the entire transportation manufacturing industry (including airplanes, ship, cars, etc.). Large cloud service providers such as Amazon, Microsoft, Google and IBM have implemented on thousands of servers across multiple geographically distributed data centers, which could result in an enormous electricity bill [87]. These companies have also spent billions of dollars to improve the energy efficiency of their multi-megawatt datacenters. However, the majority of the energy cost comes from millions of energy inefficient small and medium-sized datacenters across the continents. Besides the exponentially mounting cost by the datacenters and cloud service business, the environmental drawbacks, such as climate change and global warming, have become escalating concerns for human society. Most electricity produced in the U.S. and around the world is from carbon-intensive approaches (e.g. burning natural gas and coal) [146]. Thus, the rising "brown energy", produced by carbon-intensive means and distributed through the electrical grid [87], has to be aggressively controlled and reduced in order to prevent environment from potential hazardous impact. Besides searching for "Green" alternative resources such as wind and solar energy, smart energy conservation approaches such as cost-aware wide-area load placement, job migrations, thermal management and low-cost efficient cooling, need to be further investigated and

optimized.

Several approaches have been proposed to reduce the energy consumption at the entire HPC server system level. PowerNap [118] proposes a full-system sleep state in lieu of finer-grain power management and demonstrates that for the workload with idle time between requests, it is more efficient to serve requests quickly and then go to low-power sleep state. Elnozahy et al. [119] propose scaling the size of a server cluster according to workload. By concentrating load onto fewer, more highly-utilized servers, this technique approximates energy proportionality by reducing idle-time. Tolia et al [147] optimize a cluster through DVFS and consolidation. Femal et al. [148] propose a power-driven framework to intelligently allocate power for controlling aggregate system throughput under an energy budget.

The recent emergence of clouds with large pools of compute and storage resources raises the possibility of a new compute paradigm for scientific research. Recently, there have been several works targeting energy efficiency techniques for cloud computing. Goiri et al. [87] propose GreenSlot, a parallel batch job scheduler for a datacenter powered by a photovoltaic solar array and the electrical grid (as a backup). GreenSlot predicts the amount of solar energy that will be available in the near future, and schedules the workload to maximize the green energy consumption while meeting the jobs' deadlines. Zhu et al. [149]consider the power management problem of executing scientific workloads in virtualized environments. They develop a *pSciMapper*, a power-aware consolidation algorithm based on hierarchical clustering and an optimization search method. Le et al. [88] design policies that intelligently place and migrate load across the data centers to take advantage of time-based differences in

electricity prices and temperatures.

f) Limitations to prior work on power-aware approaches:

The focus in previous works shown above has been developing a controller that uses observational data and (in later techniques) predictive data to schedule power states and balance performance, shown in Fig 1.6. The limitation of past approaches is a lack of power-performance policies allowing users to quantitatively bound the effects of power management on the performance of their applications and systems. Existing controllers and predictors use policies fixed by a knowledgeable user to opportunistically save energy and minimize performance impact. While the qualitative effects are often good and the aggressiveness of a controller can be tuned to try to save more or less energy, the quantitative effects of tuning and setting opportunistic policies on performance and power are unknown. In other words, the controller will save energy and minimize performance loss in many cases but we have little understanding of the quantitative effects of controller tuning. This makes setting power-performance policies a manual trial and error process for domain experts and a black art for practitioners. To improve upon past approaches to high-performance power management, we need to quantitatively understand the effects of power and performance at scale.

2.2.2 Measuring, Profiling and Modeling

In this subsection, we will provide an overview of power measuring and profiling for both Multi-core and GPGPU systems. Also, we will talk about existing approaches for

modeling power, performance and system wide energy before we step into the mathematical derivation of our iso-energy-efficiency model in Chapter 4. Studies of power measurement, profiling and evaluation can help us explore the root causes of system power consumption, which can be targeted and improved by applying power/energy reduction strategies.

a) Estimating Multi-core system power consumption using simulation approach:

There have been several works studying the power consumption of major system components such as CPU, memory and network based on simulation on micro-architecture level. SimplePower [150, 151]is a cycle-accurate RT level energy estimation microprocessor simulator built on SimpleScalar [152]. It is primarily used for sequential simple-scalar architecture and estimating CPU and memory power consumption. Wattch [153] is a microprocessor simulator based on performance simulator SimpleScalar [152] for analyzing and optimizing microprocessor power dissipation at micro-architecture level for circuit designers, chip architects and compiler writers as an early stage design tool. PowerTimer [154] includes a simpler version of Wattch for PowerPC processors and also provides a web interface for users to study the performance and power tradeoff. Yang Ding et al [155] use a simulation approach to study performance-energy efficiency and scaling for parallel code running on a large number of CPUs. There are also some simulators incorporating temperature models to study system power dissipation such as the Cai-Lim model [156] and TEM^2P^2EST [157]. Orion [158] is an architecture-level internet communication power simulator and provides accurate analytical power simulation and modeling by using architecture-level parameters. SoftWatt [159] is a complete system power simulator built on top of the SimOS infrastructure and models the

CPU, memory hierarchy, and a low-power disk subsystem and quantifies the power behavior of both the application and operating system. PowerScope [160] has been done for mobile computing where battery life and power consumption are very critical design constraints. These techniques address power estimation in the context of computer architecture as opposed to computer systems that include processor, memory, and on-chip and off-chip behaviors of an application. These simulators are also execution-driven using compiled code to execute and evaluate through simulation. Simulation approaches are less scalable and difficult to be applied at runtime.

b) Direct measurement for Multi-core and GPGPU systems:

For measuring Multi-core system power consumption, there are four common ways to do it. First, directly measure an entire system's power consumption. For instance, commodity power meters such as Watts UP? Pro, industrial-strength power meter with high resolution like Yokogawa meters, and also Power Distribution Units (PDU) which are widely used for automated recording of server power consumption [161]. Second, use an ammeter to measure the amperes of current flow of the power supply line. For example, Isci et al. [162] uses an ammeter to measure the power consumption of P4 processors for their event-based power model derivation. Tiwari et al. [163] uses an ammeter to measure current of a processor from an embedded system and estimate the power cost. Third, more intrusive than the previous two, it requires tapping the power supply wires and inserting a precision resistor between power lines to measure voltage change using multi-meters. We can then use the formula $P = UI$ to calculate power. This approach can be used to study the power consumption of lower level system

components individually. Joseph et al. [164] uses this method to measure power cost for a Pentium Pro Processor. Bellosa et al. [165] measures Pentium II CPU power by inserting precision resistor between power lines in order to validate a performance counter-based power model. Similar methods have been used by Ge et al. [52] and they propose PowerPack which is a runtime framework for profiling power and energy consumption at system component level. At last, Bedard et al. [166] provide a fine-grained, integrated power measurement tool called PowerMon2, which is a device that can be inserted between a computer power supply and the computer's main board to measure power usage at each of the DC power rails supplying the board. The device can fit in an internal 3.5" hard disk drive bay, thus allowing it to be used in a 1U server chassis. Both of the last two approaches can measure power consumption at component level with high sample rating.

For GPGPU devices, there have been some out-of-band measuring approaches proposed for the purpose of validating related power models. Nagasaka et al. [49] uses three current clamps connected to a 32-channel NI instrument to monitor and profile power consumption of the Geforce 285 GTX chip. Works [91, 144] use FLUKE Hydra logger 2625A power acquisition system to record power. In our proposed research in Section 5, we use real-time in-band power measurement provided through NVIDIA Management Library (NVML). NVML interfaces offer runtime system measurements for environmental parameters such as power, temperature and system utilization. It measures the power draw of the entire board including GPU device(s) and global memory. Compared to the out-of-band approaches, NVML runtime profiling is less intrusive and

offers high sampling rates (10^5 samples per second) for underlying NVIDIA Fermi C2075.

c) Statistical Power Modeling:

There have been several works modeling power consumption by using statistical power/energy modeling with performance counters on both Multi-core and GPU devices. The basic idea is to assume there is a linear relationship between power consumption and a selected group of runtime performance counters. This simple multi-variable linear regression model (MLR) is often used as power estimation for runtime prediction on multicore systems. M Curtis-Maury et al. [38] proposed such prediction model for multi-dimensional power-performance optimization on many cores by correlating essential hardware counters (such as retired instructions, L1/L2 cache misses, memory accesses and TLB misses, etc.) with system power consumption. Joule Watcher [165] also correlated power consumption with performance events but using single performance events. Isci et al. [71] proposed a runtime power model by correlating hardware counters with CPU subunit power dissipation on real machines. Similar to Isci's work, CASTLE [164] worked on the SimpleScalar simulator. Lewis et al. [167] used a small set of tightly correlated parameters to create a model relating system energy input to subsystem energy consumption. They developed a linear regression model that relates processor power, bus activity, and system ambient temperatures into real-time predictions and to overall thermal envelopes. For GPGPU systems, Nagasaka et al. [49] and Ma et al. [144] proposed a similar approach with performance counters and linear regression to model GPU system power consumption. However, MLR approaches suffer from inaccuracy and high standard deviation, which we will address in Chapter 5.

d) Performance Modeling:

For Multi-core systems, we have discussed how to use workload decomposition method for general sequential and parallel execution in the previous subsections. In order to support our iso-energy-efficiency modeling described in Chapter 4, we improve the performance model shown previously in Section 2.1 to cover I/O and parallel overhead and show how to use runtime system tools to conduct parameterization.

For GPU based systems, there have been some performance models based on architecture level [92], PTX [168], compiler level [169] parameters, or system level simulations [50]. Kerr et al. [168] presented an emulation and translation infrastructure and its use in the characterization of GPU work-loads. Hong et al. [92] proposed an architecture- level memory parallelism aware analytical model to estimate execution cycles for NVIDIA GTX 280 test bed. Baghsorkhi et al. [169] proposed a compiler-based approach to application performance modeling on GPU architectures. The model is equipped with an efficient symbolic evaluation module to determine the effects of the structural conditions and complex memory access expressions on the performance of a GPU kernel. These architectural or compiler level performance model is architecture dependent and inputs for these complicated models are extremely difficult for system level users to achieve. And this is why we need an easy-to-use, runtime performance counter based, architecture independent, system-level performance model for applications running on GPUs. The closest related work to our proposed research in Chapter 5 is the quantitative performance analysis model proposed by Zhang et al. [50]. However, their model is based on simulation results and does not reflect the variation of runtime

performance counters.

In the past decade, many distributed communication performance models have been proposed to help users analyze, predict and evaluate either interconnection performance or memory hierarchy performance [170] [171] due to the huge impact of communication overhead on overall parallel performance. Here, we put all the related models into two categories: hardware-parameterized and software-parameterized models.

Hardware-parameterized performance models ignore the effects of middleware (such as the popular MPI) on communication overhead and application data distribution on memory communication in distributed systems. LogP [172], PRAM [173], and Bulk Synchronous Parallel (BSP) [174] approximate communication latency and overhead and incorporate multiple system characteristics. LogP sets the memory communication overhead to be constant and further incorporates asynchronous behavior. LogGP [175] is an extension of LogP model by introducing large message size with gap/byte parameter G and ignoring parallel data distribution. Other similar or improved models based on LogP have also been proposed such as LoPC [176], LoGPS [177], LoGPC [178], and PLogP [179] by overlooking or adjusting certain parameters.

For achieving high performance for distributed parallel systems, memory communication and overhead should not be simply ignored. Inspired by LogP and to address the above problem, Memory LogP has been proposed to incorporate memory hierarchy performance into communication performance models. It combines system and application characteristics in memory communication cost. Also, it separates memory communication into inherent unavoidable system overhead and additional latency. The Hierarchical Memory Model (HMM) [171] also applies memory hierarchies to network

communication. However, HMM overlooks the effects of essential network attributes common to distributed systems. Deriving from the Memory LogP model, $\mathbf{Log_nP}$ [180] and its reduction $\mathbf{Log_3P}$ [181] integrate vertical memory hierarchy performance with the cost of interconnection to accurately model the parallel performance. Memory LogP, $\mathbf{Log_nP}$ and $\mathbf{Log_3P}$ are all vertical memory level communication models.

With the emergence of multi-core clusters and their three distinct performance levels of communication including intra-CMP, inter-CMP and inter-node, horizontal memory hierarchy performance modeling is getting more attention. Bibo Tu et al [182] proposed a new analytical model $\mathbf{mlog_nP}$ and its reduction $\mathbf{2}\log_{\{2,3\}}\mathbf{P}$ to abstract memory hierarchy in vertical and horizontal levels so as to accurately model the performance of message passing on multi-core clusters.

e) Energy Modeling:

Designing detailed, sophisticated and accurate energy consumption and efficiency modeling for scalable systems is the first step to identify the problems and help users choose the best performance-energy scalable algorithms and power conservation techniques. The power-aware speedup model proposed by Ge and Cameron [74] is a generalization of Amdahl's Law for energy. While this model accurately captures some of the effects of energy management on speedup, it provides little insight to the root cause of poor power-performance scalability. The Energy Resource Efficiency (ERE) metric proposed by Jiang et al [183] defines a link between performance and energy variations in a system to clearly highlight the various performance-energy tradeoffs. As with other models that identify energy efficiency, this model analyzes at a very high-level

and does not identify causal relationships with poor metric results. The energy model proposed by Ding et al [155] uses circuit-level simulation to analyze power-performance tradeoffs. While this model shows promise for circuit-level design, it is too unwieldy for use in analyzing existing large-scale power-scalable clusters. The model also makes a number of simplifying assumptions such as homogenous workloads and no execution time overlap between system components making it less practical for modeling real systems.

Chapter 3

Energy Profiling, Analysis and Evaluation of Parallel Applications on Multi-core Based Systems

In this chapter, we are going to present methodologies and techniques to profile, analyze and evaluate power and performance characteristics of several mainstream parallel applications based on our PowerPack Toolkit [52]. Specifically, we will show how to identify the unique power/performance patterns and related bottlenecks of

individual application [52, 53]. Specifically, we use our power-performance profiling framework PowerPack to study the power and energy profiles of the HPC Challenge benchmarks [55]. We present detailed experimental results along with in-depth analysis of how each benchmark's workload characteristics affect power consumption and energy efficiency. This chapter summarizes various findings using the HPC Challenge benchmarks, including but not limited to: 1) identifying application power profiles by function and component in a high performance cluster; 2) correlating applications' memory access patterns to power consumption for these benchmarks; and 3) exploring how energy consumption scales with system size and workload. This chapter also serves as the foundation for the succeeding chapters because both modeling framework and optimization heavily rely on the profiling, analyzing and evaluating functionalities for better prediction accuracy and results validation.

3.1 Introduction

Today it is not uncommon for large-scale computing systems and data centers to consume massive amounts of energy, typically 5–10 megawatts in powering and cooling [184]. This results in large electricity bills and reduced system reliability due to increased heat emissions. The continuing need to increase the size and scale of these data centers means that energy consumption potentially limits future deployments of high performance computing (HPC) systems.

Energy efficiency is now a critical consideration for evaluating high performance computing systems. For example, the Green500 list [14] was launched in November 2006 to rank the energy efficiency of worldwide supercomputer installations. Meanwhile, the

TOP500 supercomputing list, which ranks the most powerful supercomputing installations, has also started to report total power consumption of its supercomputers [8].

Today, there is no clear consensus as to which benchmark is most appropriate for evaluating the energy efficiency of high performance systems. Both the Green500 and the Top500 use the LINPACK [56] benchmark in their evaluations. In both cases, LINPACK was chosen more for reporting popularity than for its ability to evaluate energy efficiency. As with most benchmark activities, many high performance computing stakeholders do not agree with the choice of LINPACK as an energy efficiency benchmark.

Despite the increasing importance of energy efficiency and the controversy surrounding the need for standardized metrics, there have been few studies detailing the energy efficiency of high performance applications. Understanding where and how power is consumed for benchmarks used in acquisition is the first step towards effectively tracking energy efficiency in high performance systems and ultimately determining the most appropriate benchmark. The only detailed HPC energy efficiency study to date has been for the National Aeronautics and Space (NAS) parallel benchmarks [185].

While both the NAS benchmarks and LINPACK are widely used, the HPC Challenge (HPCC) benchmarks [55] were specifically designed to cover aspects of application and system design ignored by the former benchmarks and to aid in system procurements and evaluations. HPCC is a benchmark suite developed by the Defense Advanced Research Projects Agency (DARPA) High Productivity Computing System (HPCS) program. HPCC consists of seven benchmarks; each focuses on a different part of the extended memory hierarchy. The HPCC benchmark provides a comprehensive

view of a system's performance bounds for real applications.

A detailed study of the energy efficiency of the HPCC benchmarks is crucial to understanding the energy boundaries and trends of HPC systems. In this subsection, we present an in-depth experimental study of the power and energy profiles for the HPC Challenge benchmarks on real systems. We use the PowerPack toolkit [52] for power-performance profiling and data analysis at function and component level in a parallel system. This work results in several new findings:

- The HPCC benchmark not only provides performance bounds for real applications, but also provides boundaries and reveals some general trends for application energy consumption.

- Each HPCC application has its own unique power profiles and such profiles vary with runtime settings and system configurations.

- An application's memory access rate and locality strongly correlate to power c consumption. Higher memory access locality is indicative of higher power consumption.

- System idle power contributes to energy inefficiency. Lower system idling power would significantly increase energy efficiency for some applications.

The rest of this chapter is organized as follows. Section 3.2 reviews PowerPack and the HPCC benchmarks. Sections 3.3 and 3.4 present an in-depth examination and analysis of the power and energy profiles of the HPCC benchmarks running on Multi-

core based clusters. Finally, Section 3.5 summarizes our conclusions.

3.2 Descriptions of the Experimental Environment

To detail the power/energy characteristics of the HPCC benchmarks on high performance systems, we use experimental system investigation. We begin by describing the setup of our experimental environment, which consists of three parts: the PowerPack power-profiling framework, the HPCC benchmark suite, and the HPC system under test.

3.2.1 The PowerPack Power-Profiling Framework

Figure 3.1 shows the software architecture of the PowerPack framework. PowerPack

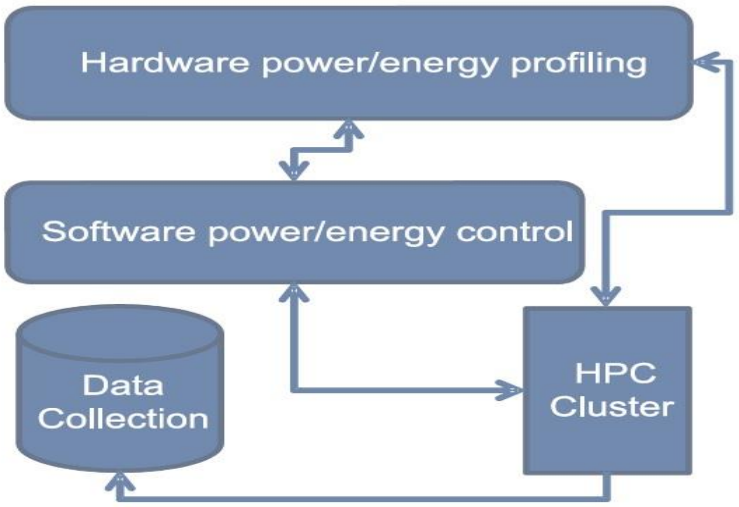


Figure 3.1: PowerPack software architecture.

consists of five components: hardware power/energy profiling, software power/energy profiling control, software system power/energy control, data collection/fusion/analysis,

and the system under test. The hardware power/energy profiling module simultaneously measures system power and multiple components' power. PowerPack measures the system power using a power meter plugged between the computer power supply units and electrical outlets. Component power is measured using precision resistors inserted into the DC lines carrying output from the computer's power supply unit, and voltage meters attached at two ends of the resistors. The software power/energy profiling control module provides the meter drivers and interface (i.e. application programming interfaces (APIs)) for the system or an application to start, stop, or label power profiling. The software system power/energy module is a set of interfaces (i.e. APIs) to turn on or off, or to set the frequencies, of processors. The data fusion module merges multiple data streams and transforms them into a well-structured format. Normally PowerPack directly measures one physical node at a time, but it is scalable to the entire computer cluster by using node remapping and performance-counter-based analytical power estimations.

Table 3.1: Performance characteristics of the HPCC benchmark suite.

Benchmark	Spatial locality	Temporal locality	Mode	Detail
HPL	high	high	Global	Stresses floating-point performance
DGEMM	high	high	Star + Local	Stresses floating-point performance
STREAM	high	low	Star	Measures memory bandwidth
PTRANS	high	low	Global	Measures data transfer rate
FFT	low	high	Global + Star + Local	Measures floating-point and memory-transfer performance
RandomAccess	low	low	Global + Star + Local	Measures random updates of integer memory
Latency/Bandwidth	low	low	Global	Measures latency and bandwidth of communication patterns

3.2.2 The HPC Challenge Benchmark

The HPC Challenge (HPCC) benchmark is a benchmark suite that aims to augment the TOP500 list and to evaluate the performance of HPC architectures from multiple aspects. HPCC organizes the benchmarks into four categories; each category represents a type of memory access pattern characterized by the benchmark's memory access spatial and temporal locality. Currently, HPCC consists of seven benchmarks: HPL, STREAM, RandomAccess, PTRANS, FFT, DGEMM and b_eff Latency/Bandwidth. To better analyze the power/energy profiles in the next section, we summarize the memory access patterns and performance bounds of these benchmarks in Table 3.1.

We use a classification scheme to separate the distinct performance phases that make up the HPCC benchmark suite. In *Local* mode, a single processor runs the benchmark. In *Star* mode, all processors run separate independent copies of the benchmark with no communication. In *Global* mode, all processing elements compute the benchmark in parallel using explicit data communications.

3.2.3 The HPC System under Test

The Dori system is a cluster of nine two-way dual-core AMD Opteron 265-based server nodes. Each node consists of six 1-GB memory modules, one Western Digital WD800 SATA hard drive, one Tyan Thunder S2882 motherboard, two CPU fans, two system fans, three Gigabit Ethernet ports, and one Myrinet interface. The compute nodes run CentOS Linux with kernel version 2.6. During our tests, we used MPICH2 as the communication middleware. The network interconnection is via gigabit Ethernet.

3.3 HPCC Benchmark Power Profiling and Analysis

3.3.1 A Snapshot of the HPCC Power Profiles

The power profiles of each application are unique. Figure 3.2 shows power profiles of the HPCC benchmarks running on two nodes (8 cores) of the Dori cluster. These profiles are obtained using the problem size where HPL achieves its maximum performance on two nodes. Power consumption is tracked for major computing components including CPU, memory, disk and motherboard. As we will see later, these four components capture nearly all the dynamic power usage of the system that is dependent on the application. A single HPCC test run consists of a sequence of eight benchmark tests as follows: 1) Global PTRANS; 2) Global HPL; 3) Star DGEMM +

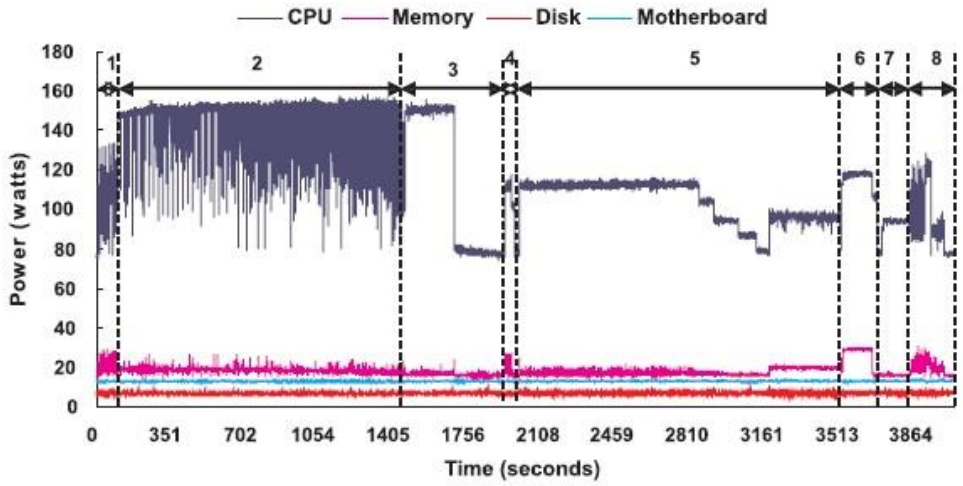


Figure 3.2: A snapshot of the HPCC power profile. The power consumption of major components per compute node when running a full HPCC benchmark suite using eight cores. The entire run of HPCC consists of eight micro benchmark tests in this order: 1) PTRANS; 2) HPL; 3) Star DGEMM + single DGEMM; 4) Star STREAM; 5) MPI RandomAccess; 6) Star RandomAccess; 7) Single RandomAccess; 8) MPI FFT, Star FFT, single FFT and latency/bandwidth. In this test, the problem size fits the peak execution rate for HPL. For LOCAL tests, we run a benchmark on a single core with three idle cores.

Local DGEMM; 4) Star STREAM; 5) Global MPI RandomAccess; 6) Star RandomAccess; 7) Local RandomAccess; and 8) Global MPI FFT, Star FFT, Local FFT and Latency/Bandwidth. Figure 3.2 clearly shows the unique power signature of each mode of the suite. We make the following observations from Figure 3.2:

• Each test in the benchmark suite stresses processor and memory power relative to their use. For example, as Global HPL and Star DGEMM have high temporal and spatial locality, they spend little time waiting on data, stress the processor's floating point execution units intensively, and consume more processor power than other tests. In contrast, Global MPI RandomAccess has low temporal and spatial memory access locality; thus this test consumes less processor power due to more memory access delay, and more memory power due to cache misses.

• Changes in processor and memory power profiles correlate with communication to computation ratios. Power varies for global tests such as PTRAN, HPL, and MPI_FFT because of their computation and communication phases. For example, the HPL computation phases run at 50 watts higher than its communication phases. Processor power does not vary as much during Star (embarrassingly parallel) and Local (sequential) tests due to limited processing variability in the code running on each core. In Global modes, memory power varies, but not widely in absolute wattage since memory power is substantially less than processor power on the system under test.

• Disk power and motherboard power are relatively stable over all tests in the benchmarks. None of the HPCC benchmarks stresses local disks heavily. Thus power variations due to disk use are not substantial. On this system, the network interface card

(NIC), and thus its power consumption, is integrated in the motherboard. Nonetheless, communication using the gigabit Ethernet card does not result in significant power use even under the most intensive communication phases.

• Processors consume more power during Global and Star tests since they use all processor cores in the computation. Local tests use only one core per node and thus consume less energy.

3.3.2 Power Distribution among System Components

Each test within the HPCC benchmarks is unique in its memory access pattern. Together, they collectively examine and identify the performance boundaries of a system

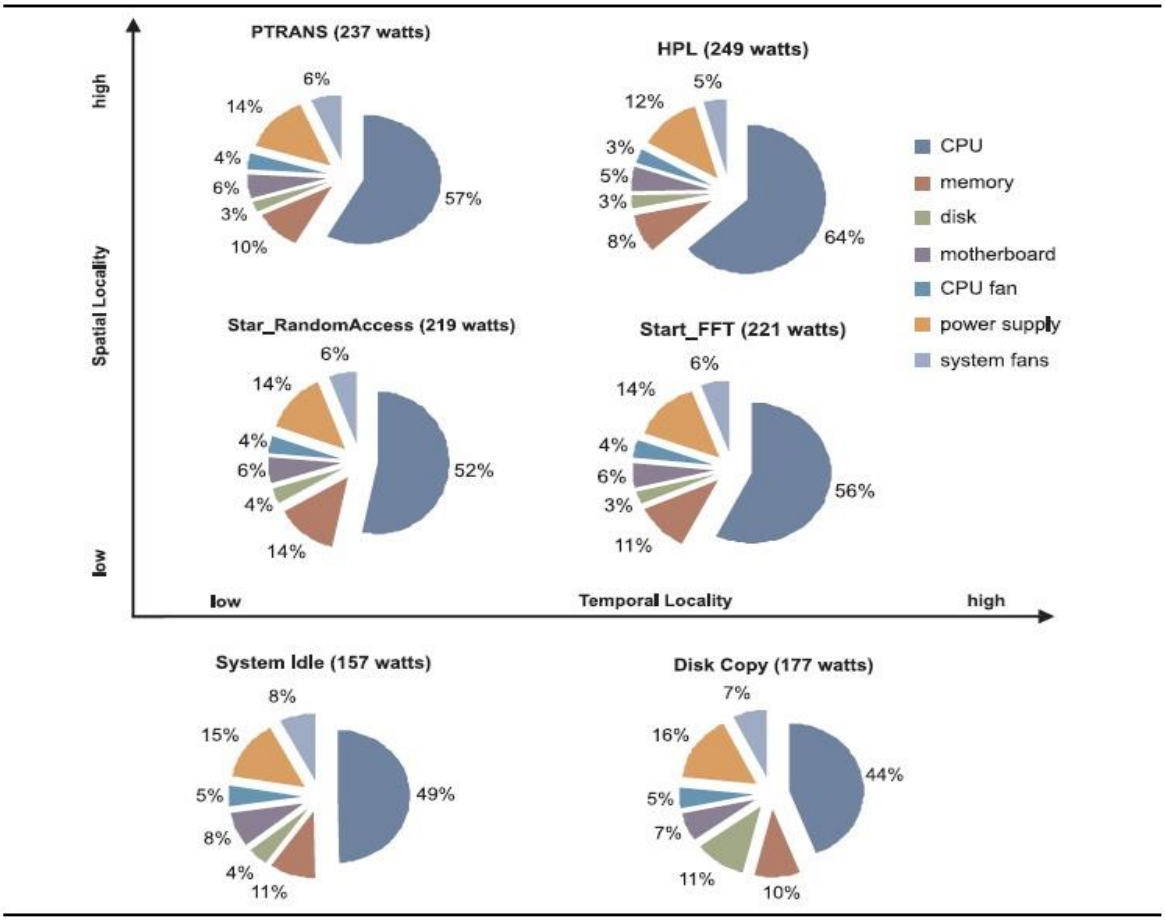


Figure 3.3: System power distribution for different workload categories. This figure shows system power broken down by component for six workload categories. The top four categories represent four different memory access locality patterns. The other two workloads are system idle, i.e., no user application running, and disk copy, i.e., concurrently running the Linux standard copy program on all cores. The number given in the subheading for each category is its total average power consumption.

with various combinations of high and low temporal and spatial locality, as described in Table 3.1. We hypothesize that the HPCC benchmarks will identify analogous boundaries for power and energy in high performance systems and applications. HPL (high, high), PTRAN (low, high), Star_FFT (high, low) and Star_RandomAccess (low, low) represent the four combinations of (temporal, spatial) locality captured by the HPCC tests. Figure 3.3 superimposes the results for these four tests on an X–Y plot of temporal–spatial sensitivity. For comparison, below the graph we also provide the power breakdowns for system idle (i.e. no active user applications), and for disk copy: this stresses disk I/O access, which is not captured by the HPCC tests.

The total power consumption figures for each of the isolated phases of the HPCC benchmark depicted in Figure 3.3, in increasing order, are: system idle (157 watts), disk copy (177 watts), Star_RandomAccess (219 watts), Star_FFT (221 watts), PTRAN (237 watts), and HPL (249 watts). The power numbers for the two Star benchmarks are representative given the stable power profiles, while the power numbers for PTRAN and HPL are of time instances when processor power reaches the respective maximum. We conclude the following from the data:

• Since lower temporal and spatial locality imply higher average memory access delay times, applications with (low, low) temporal–spatial locality use less power on

average. Memory and processor power dominate the power budget for these tests. Idling of the processor during memory access delays results in more time spent idling at lower dynamic processor power use, thus leading to lower average system power use.

• Since higher temporal and spatial locality imply lower average memory access delay times, applications with (high, high) temporal–spatial locality use more power on average. Memory and processor power dominate the power budget for these tests. Intensive floating point execution leads to more activities, and higher active processor power and average system power. Since HPL has both high spatial and high temporal locality, it has the highest power consumption and thus is a candidate for measuring the maximum system power that an application would require.

• Mixed temporal and spatial locality implies mixed results that fall between the average power ranges of (high, high) and (low, low) temporal–spatial locality tests. These tests must be analyzed more carefully by phase, since communication phases can affect the power profiles substantially.

Overall, processor power dominates total system power for each test shown in Figure 7; the processor accounts for as little as 44% of the power budget for disk copy and as much as 61% of the power budget for HPL. Processor power also shows the largest variance: as high as 59 watts for HPL. For workloads with low temporal and spatial locality, such as Star_RandomAccess, processor power consumption varies by as much as 40 watts.

Memory is the second largest power consumer in the HPCC tests and accounts for 8% (HPL) and 13% (RandomAccess) of the consumption of the total system. Low spatial locality causes additional memory accesses and thus higher memory power consumption.

For Star_RandomAccess, the memory power consumption increases by 11 watts over that of the idle system.

Disk normally consumes much less power than processor and memory, but may consume up to 11% of total system power for disk-I/O-intensive applications such as disk copy. HPCC, LINPACK, and the NAS parallel benchmarks, like most HPC benchmarks, do not stress disk accesses. This may be due to the fact that in high performance environments with large storage needs the data storage is remote from the computational nodes, in a disk array or other data appliance. We use the disk copy benchmark primarily to illustrate the way the power budget can change under certain types of loads; this means that local disk power should not be readily dismissed for real applications.

The system under test consumes a considerable amount of power when there is no workload running. We have omitted discussion of power supply inefficiency since it is fundamentally an electrical engineering problem, which accounts for another 20–30% of the total system power consumed under these workloads. Altogether, the processor, the power supply, and the memory are the top three power-consuming components for the HPCC benchmarks and typically account for about 70% of total system power.

3.3.3 Detailed Power Profiling and Analysis of Global HPCC Tests

The previous subsection focused on the effect of memory access localities of applications on power distribution over the system components. In this subsection we study the way in which parallel computation changes the locality of data accesses and impacts the major computing components' power profiles over the execution of the

benchmarks. We still use the same four benchmarks, all of them in Global mode.

PTRANS implements a parallel matrix transposition. HPCC uses PTRANS to test system communication capacity using simultaneous large message exchanges between pairs of processors. Figure 3.4(a) shows the power profiles of PTRANS with problem size N = 15 000. Processor power varies within iterations. The spikes correspond to computation phases and valleys correspond to system-wide large message exchanges, which occur on iteration boundaries as marked in the figure. While PTRANS's power consumption is reasonably high during computation phases, its power consumption during communication phases is close to the lowest. Also, the memory power profile is stable with only slight changes. PTRANS has high spatial locality and consequently a low cache miss rate. Communication phases are sufficiently long for the processor to

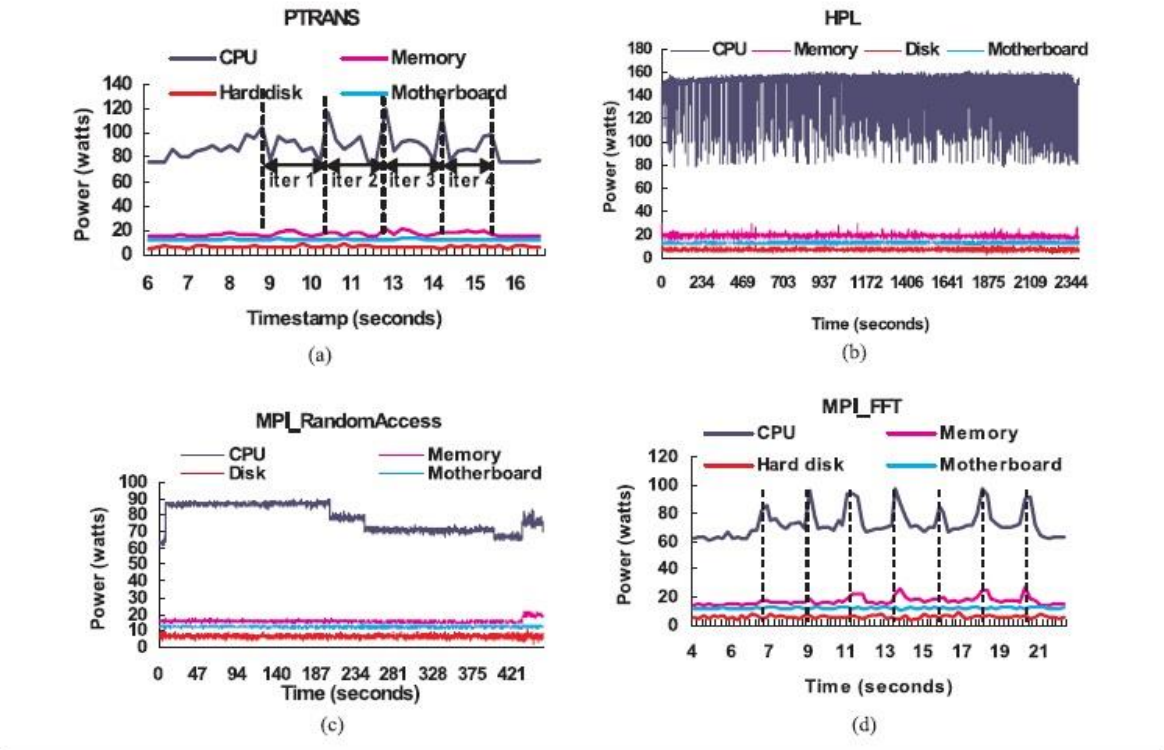


Figure 3.4: Detailed power profiles of four HPCC benchmarks running across eight nodes on 32 cores in Global mode. This figure shows the power consumptions of system major components over time for a) PTRANS, b) HPL, c) MPI_RandomAccess, and d) MPI_FFT during their entire execution.

reduce its power consumption. Additionally, the computation phases of PTRANS focus on data movement to perform the transposition, which is less computationally intensive than the matrix computations in HPL; this results in a lower average power consumption for the PTRANS computation phases.

HPL is a computation-intensive workload interspersed with short communications. HPCC uses HPL to assess peak floating-point performance. HPL's processor power profiles display high variance, as shown in Figure 3.4(b). For this data, we experimentally determined a problem size where HPL achieves peak execution rate on 32 cores across eight nodes. At peak HPL performance, processor power reaches about 150 watts, which is the maximum over all HPCC tests. Nevertheless, processor power fluctuates dramatically, and it may drop to 90 watts during the short communications in HPL. Memory power does not vary much during HPL execution. This is because HPL has high temporal and spatial memory access locality, and thus does not frequently incur cache misses.

MPI_RandomAccess is a Global test that measures the rate of integer updates to random remote memory locations. This code's memory access pattern has low spatial and temporal locality. As shown in Figure 3.4(c), the processor power profile for MPI_RandomAccess follows a steady step function that reflects the relatively low computational intensity of random index generation. MPI_RandomAccess stresses inter-processor communication of small messages. Processor power decreases during the message exchange of indices with other processors. Memory power jumps towards the

end of the run as the local memory updates occur once all indices are received. This memory power consumption is notably almost the highest among all the Global tests. The only higher memory power consumption we observed was for the embarrassingly parallel (EP) version of this code, Star_RandomAccess.

MPI_FFT is a Global test that performs the fast Fourier transform operation. Figure 3.4(d) shows the power profiles of MPI_FFT with input vector size of 33 554 432. MPI_FFT's processor power profile is similar to that of PTRANS, in which communication tends to decrease processor power on function boundaries, as marked in the figure. Unlike that of PTRANS, however, the memory power profile of MPI_FFT exhibits obvious fluctuations. These differences are the results of different memory

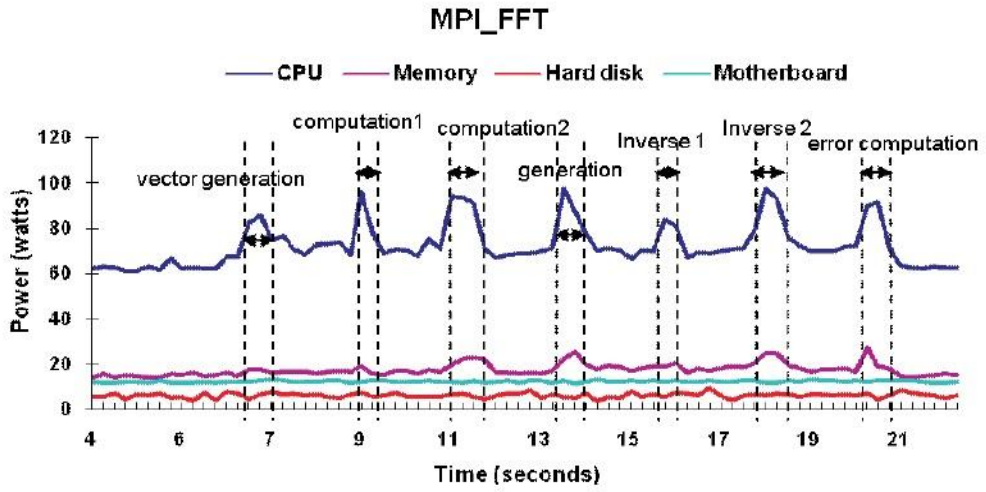


Figure 3.5: Detailed power-function mapping of MPI_FFT in HPCC. PowerPack shows processor power rises during computation phases, and drops during communication phases. The seven spikes in processor power profile correspond to vector generation, computation1, computation2, random vector generation, inverse computation1, inverse computation2, and computation of error; the valleys correspond to transpositions that involve inter-processor communications.

access patterns inherent in these two benchmarks. While PTRANS has low temporal locality and high spatial locality, MPI_FFT has high temporal locality and low spatial locality. Higher miss rates lead to higher power consumption in memory for MPI_FFT compared with PTRANS.

Figure 3.5 shows the amplified version of MPI_FFT at the function level. FFT conducts global transpositions for large complex vectors via inter-processor communication and spends over 70% of its execution time in communication phases. The seven spikes in the CPU power profile correspond to seven computation-intensive phases: vector generation, computation1, computation2, random vector generation, inverse computation1, inverse computation2, and computation of error. The valleys in the processor power profile correspond to data transpositions involving inter-processor communications. The interesting observation is that memory power goes up during computation phases and drops during communication phases, meaning that memory access are more intensive for the former. Such profiling reveals the detailed dynamics of system activity and power consumption.

3.4 HPCC Benchmark Energy Profiling and Analysis

While we are interested in power (P), the rate of energy consumption at time instances when an application executes, we are also interested in the system energy consumption (E) required to finish executing the application. Energy is the area under the curves given by the power profiles previously discussed. It reflects the electrical cost of running an application on a particular system. In this section we analyze the resulting impact of power profiles on energy consumption for parallel systems. In particular, we

study the energy efficiency of these applications and systems as they scale.

3.4.1 Energy Profiling and Efficiency of MPI_FFT under Strong Scaling

Strong scaling describes the process of fixing a workload and scaling up the number of processors. Embarrassingly parallel codes often scale well under strong scaling conditions. For these types of codes, scaling system size reduces execution times linearly with constant nodal power consumption. As a result, the total energy consumption maintains the same. When measuring energy efficiency with the amount of

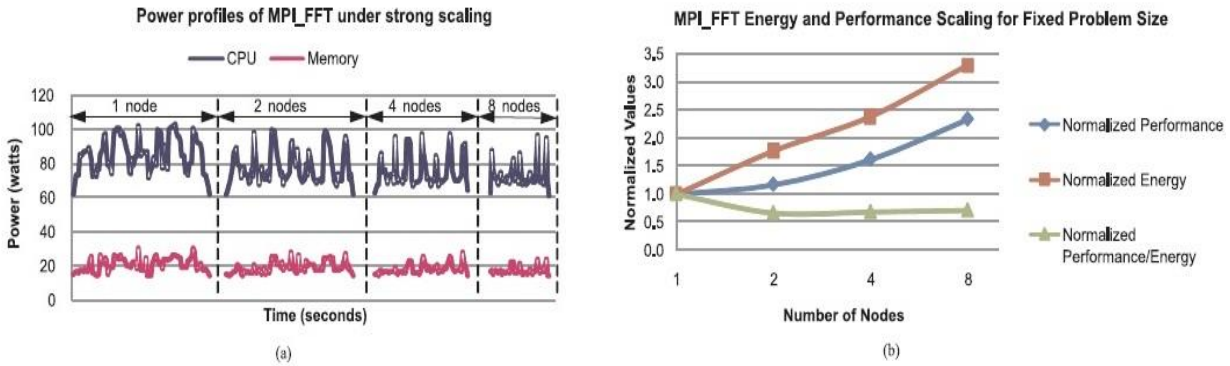


Figure 3.6: The power, energy, performance, and energy efficiency of MPI_FFT. In this figure we run the Global MPI_FFT test for a fixed problem size, using a varying number of nodes with four cores per node. Both performance and energy are normalized against the respective values when using one node. We use normalized performance/energy as a measure of achieved energy efficiency.

workload computed per unit of energy, an embarrassingly parallel code achieves better energy efficiency as system size scales.

However, for codes that are not embarrassingly parallel, such as Global

MPI_FFT, the energy efficiency is less clear. In Figure 3.6, we show the power per node, system energy, performance, and resulting energy efficiency of a set of Global MPI_FFT tests. During the test, we use all the four cores on each node. Thus, running on eight nodes actually creates 32 threads with one thread per core. Figure 3.6(a) shows the components' power per node. As system size scales, the duration of high processor power phases (computation phases) becomes shorter, and the duration of lower processor power phases (communication phases) becomes longer. In addition, the highest processor power within one run slightly decreases when the number of nodes increases. Memory power also shows a similar trend.

Figure 3.6(b) shows that both performance and energy increase sub-linearly as system size scales. However, energy increases much faster than performance for the system sizes we have measured. For example, running on eight nodes gains a 2.3 times speedup, but costs 3.3 times as much energy as that used when running on one node. This observation implies the need for appropriate tradeoffs between running an application faster and maintaining sufficient energy efficiency, or constraining running costs.

3.4.2 Energy Profiling and Efficiency of HPL under Weak Scaling

Weak scaling describes the process of simultaneously increasing both workload and the number of processors being used. The discussions on energy efficiency under strong scaling imply that in order to maintain a desired level of energy efficiency, we also need to maintain the parallel efficiency of the application. In the experiments below, we want to find out how effectively the weak scaling case can maintain its parallel efficiency

and its energy efficiency.

Figure 3.7 shows the directly measured power, energy, performance, and energy efficiency of HPL on the Dori cluster under weak scaling. In our experiments we increase the problem size of HPL in such a way that HPL always achieves its maximum performance on the respective number of nodes. Figure 3.7(a) shows the power profiles of HPL. With weak scaling, neither processor power nor memory power change perceptibly with the number of nodes during the computation phases, though power decreases slightly during the communication phases as the system scales. As a result, average nodal power remains almost the same.

Figure 3.7(b) shows the normalized performance and derived total energy. We observe that both performance and energy increase roughly linearly as system size

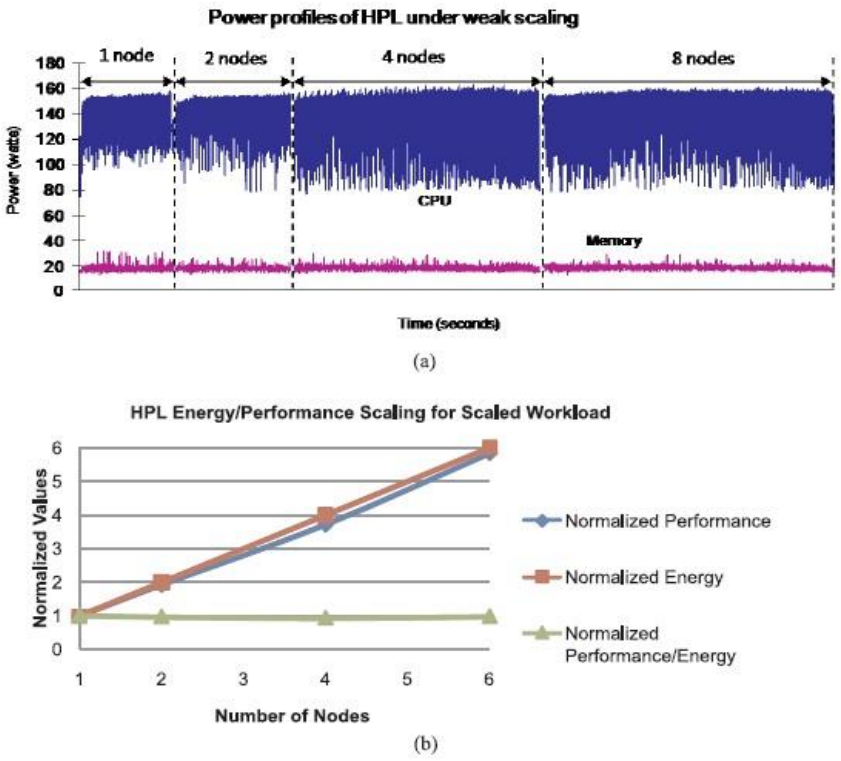


Figure 3.7: The power, energy, performance, and energy efficiency of HPL under weak scaling. In this figure, both performance and energy are normalized against the corresponding values when using one node. We also plot the normalized performance/energy value as a measure of achieved energy efficiency.

increases. Accordingly, the energy efficiency remains constant as system size increases. These observations exactly match our expectations. By increasing the problem size of HPL to match the system's maximum computational power, we maintain the memory spatial and temporal locality and the parallel efficiency, which in turn leads to constant energy efficiency. We would like to note that observations from weak scaling of HPL are very close to those from the embarrassingly parallel case.

3.5 Chapter Summary

In summary, we evaluated the power and performance profiles of the HPC Challenge benchmark suite on a multi-core-based HPC cluster. We used the PowerPack toolkit to collect the power and energy profiles and isolate them by components. We organized and analyzed the power and energy profiles according to each benchmark's memory access locality.

Each application has a unique power profile characterized by power distribution among major system components, especially processor and memory, as well as power variations over time. By correlating the HPCC benchmark's power profiles to performance phases, we directly observed strong correlations between power profiles and memory locality.

The power profiles of the HPCC benchmark suite reveal power boundaries for real applications. Applications with high spatial and temporal memory access locality consume the most power but achieve the highest energy efficiency. Applications with

low spatial or temporal locality usually consume less power but also achieve lower energy efficiency.

Energy efficiency is a critical issue in high performance computing, and it requires further study, since the interactions between hardware and application affect power usage dramatically. For example, for the same problem on the same HPC system, choosing an appropriate system size will significantly affect the achieved energy efficiency.

Chapter 4

ISO-ENERGY-EFFICIENCY

Future large scale high performance supercomputer systems require high energy efficiency to achieve exaflops computational power and beyond. Despite the need to understand energy efficiency in high-performance systems, there are few techniques to evaluate energy efficiency at scale. In this chapter, we propose a system-level iso-energy-efficiency model to analyze, evaluate and predict energy-performance of data intensive

parallel applications with various execution patterns running on large scale power-aware multi-core based clusters [4, 30]. Our analytical model can help users explore the effects of machine and application dependent characteristics on system energy efficiency and isolate efficient ways to scale system parameters (e.g. processor count, CPU power/frequency, workload size and network bandwidth) to balance energy use and performance. First**,** we derive our iso-energy-efficiency model and apply it to the NAS Parallel Benchmarks on two power-aware clusters. Our results indicate that the model accurately predicts total system energy consumption within 5% error on average for parallel applications with various execution and communication patterns. Next, we demonstrate effective use of the model for various application contexts (including EP, CG and FT in NPB suite) and in scalability decision-making**.** Finally**,** we propose the use of "correlation functions" to quantitatively explain the isolated and interacting effects of these two essential system-level parameters, workload and frequency, for three representative applications: LINPACK, row-oriented matrix multiplication, and 3D Fourier transform. We show quantitatively that the iso-energy-efficiency model with correlation functions is effective at maintaining efficiency as system size scales.

4.1 Introduction

As depicted in Figure 1.6 in Chapter 1, the focus in previous works has been developing a controller that uses observational data and (in later techniques) predictive data to schedule power states and balance performance. A key limitation of past approaches is a lack of power-performance policies allowing users to quantitatively

bound the effects of power management on the performance of their applications and systems. Existing controllers and predictors use policies fixed by a knowledgeable user to opportunistically save energy and minimize performance impact. While the qualitative effects are often good and the aggressiveness of a controller can be tuned to try to save more or less energy, the quantitative effects of tuning and setting opportunistic policies on performance and power are unknown. In other words, the controller will save energy and minimize performance loss in many cases but we have little understanding of the quantitative effects of controller tuning. This makes setting power-performance policies a manual trial and error process for domain experts and a black art for practitioners. To improve upon past approaches to high-performance power management, we need to quantitatively understand the effects of power and performance at scale.

We use a modeling based approach that captures power-performance tradeoffs system-wide and at scale. Our basic idea is to apply the concept of iso-efficiency [72] [73] for performance, or the ability to maintain constant per-node performance as a system scales, to power-performance management. We want to create techniques that allow us to quantitatively control and maintain power-performance as systems and applications scale; we thus name our approach *iso-energy-efficiency*. In conducting this work, we found the first essential step toward controlling for *iso-energy-efficiency* was to create a detailed, sophisticated, accurate model of the effects of performance and power on scaled systems and applications.

The contributions of this work include:

- Development of a fine-grained, analytical *iso-energy-efficiency* model that incorporates parallel system components and computational overlap at scale.
- Accuracy analysis and verification of the model on two power-scalable clusters.
- Results from a detailed power-performance scalability analysis of *EP*, *FT* and *CG* from the *NAS* Parallel Benchmarks, LINPACK, and row-oriented matrix multiplication, including use of the *iso-energy-efficiency* model to bound and maintain system energy efficiency at scale.
- Utilization of two correlation functions to identify appropriate value settings for workload and power scaling using semi-runtime monitoring tool PowerScale in order to main constant system-wide energy efficiency.
- Classification of parallel applications by how their system-wide energy efficiency is affected by the problem size scaling approach.

To the best of our knowledge, this is the first system-level, scalable, analytical model of both power and performance on real systems and applications. We begin the succeeding discussions with motivation examples followed by an overview of the model. Then, we show full derivation of the model and its parameters. Next, we discuss about methodology including two correlation functions, followed by model validation and results using the model and correlation functions to perform scalability analysis of the NAS Parallel Benchmarks, LINPACK and row-oriented Matrix Multiplication. We will also provide categorization of parallel applications based on how their energy efficiency is affected by the problem size scaling approach. Lastly, we will summarize the chapter.

4.2 Motivation Examples

a) Can't we just use performance or performance efficiency models for energy/energy efficiency?

Fig 1.4 from Chapter 1 shows the performance and energy efficiency curve for *FT*. This graph clearly shows that performance efficiency model cannot predict system energy efficiency accurately. This is because energy consumption is an interacting effect of both power and performance. Therefore, being able to identify the root cause of energy inefficiency would allow us to improve system and application efficiency more in line with the ideal isoefficient case. However, analyzing and potentially predicting energy efficiency is exceedingly difficult since we must identify and isolate the interacting effects of power and performance. For example, changing the power settings on a

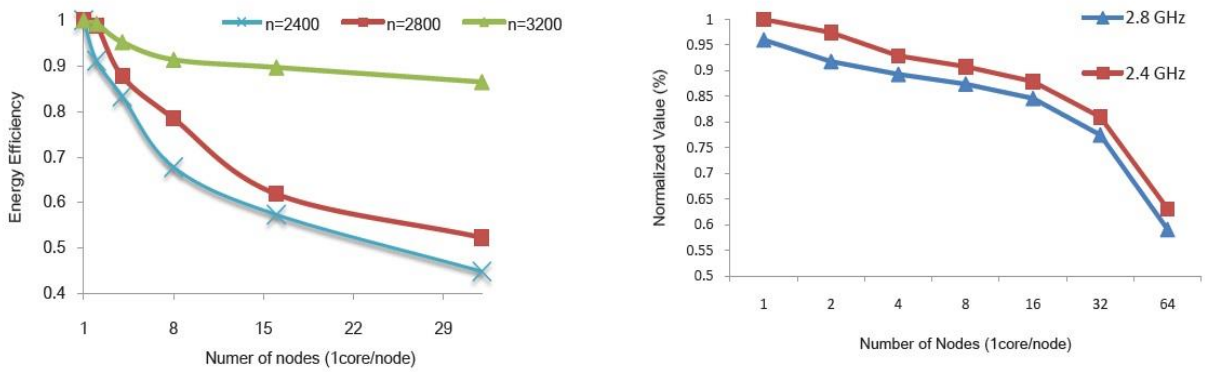


Figure 4.1 (a): Energy efficiency scaling for Cannon's algorithm with various problem size under the fixed frequency. Figure 4.1 (b): Energy efficiency scaling for Fourier Transform under two frequencies.

processor using DVFS affects performance which in turn potentially affects the length of time an application takes to complete which is key to its overall energy usage.

b) Problem Size:

To illustrate the effect of increasing problem size on system energy efficiency, we ran a simple experiment. For three different sizes of matrix (problem sizes) we applied Cannon's algorithm using an increasing number of nodes (system size). We measured the energy efficiency of the system. We observe from Figure 4.1 (a) that overall energy efficiency goes down when increasing system size. However, the energy efficiency improves when increasing the problem size. For example, using 4 nodes with a problem size of 2800 has the same system energy efficiency as 32 nodes with 3200. The graph shows that one potential way to improve or maintain system energy efficiency is to simultaneously increase both system and problem size. Although this is the case for Cannon's algorithm we want to know whether problem size scaling is a generally applicable technique for maintaining system energy efficiency. It may be the case that scaling problem size has a negative effect, or no effect, on some applications.

c) Processor power modes (CPU frequency):

To illustrate the effect of changing frequency on system energy efficiency we performed another experiment. We used Fourier transform with a fixed problem size on an increasing number of nodes (system size) while varying the CPU frequency on all nodes. We measured the energy efficiency of the system. Figure 4.1 (b) shows that using 2.4 GHz as the core frequency improves the energy efficiency by an average of 3:2% compared to 2.8GHz. This shows that one potential way to improve, or maintain system energy efficiency is to scale the CPU frequency. Similarly to the problem size scaling technique, we want to know what the relationship is between CPU frequency scaling and system energy efficiency. It may be the case that CPU frequency scaling is a generally

applicable technique for maintaining system energy efficiency. However, for some applications it may be the case that varying the frequency does not result in energy reduction and will not improve system energy efficiency. Whether we can achieve energy reduction by scaling the CPU frequency heavily depends on the execution patterns of the applications and available frequency scaling range on a system.

4.3 Iso-Energy-Efficiency Model (I-E-E model)

We developed the I-E-E model to address two key points. Firstly predict total energy consumption of large-scale systems and secondly model how energy efficiency is affected by altering system parameters such as CPU frequency, problem size, node count and interconnect. For a complete list of parameters used in the model please refer to Table A in the ***Appendix Section***.

We define E_1 as the total energy consumption of sequential execution on one processor and E_p as the total energy consumption of parallel execution for a given application on p parallel processors. Let E_o represent the additional energy overhead required for parallel execution and running extra system components. So we can define the **general form** of I-E-E model as:

$$\theta = EE = \frac{E_1}{E_p} = \frac{E_1}{E_1 + E_o} = \frac{1}{1 + \frac{E_o}{E_1}} = \Big/\left(1 + \frac{\alpha T_o P_{total-idle} + W_{co} t_c \Delta P_c + W_{mo} t_m \Delta P_m}{\alpha T_1 P_{total-idle} + W_c t_c \Delta P_c + W_m t_m \Delta P_m}\right) \quad (4.1)$$

Where $T_1 = (W_c t_c + W_m t_m)$ and $T_o = \left(W_{co} t_c + W_{mo} t_m + \sum_{i=1}^{p} T_{net_i} \right)$ $(1 \le i \le p)$† *

(: For simplicity,* $\sum_{i=1}^{p} \mathbf{T_{net_i}}$ *can be estimated by* $\mathbf{Mt_{msg} + Bt_{Byte}}$ *. For more complicated communication patterns, communication models such as Pairwise exchange/Hockney[186], LogP [172], LogGP [175], PLogP [179], BSP [174],etc., can be used to replace* $\sum_{i=1}^{p} \mathbf{T_{net_i}}$ *according to specific parallel algorithm.* $\mathbf{T_o}$ *includes: a) the time p-1 processes spend executing inherently sequential code. b) the time all p processes spend performing redundant computations and inter-processor communications.)*

Equation (4.1) forms the basis for computing *iso-energy-efficiency*. The challenge is to capture each of the parameters used in these equations for a given application and system combination. Tables 3 shows the model parameters used to calculate *EE*, which can be classified as either machine-dependent or application-dependent. The machine-dependent variable vector can be described as a function of frequency (i.e. computational speed) and workload bandwidth (i.e. computational throughput) of the hardware:

$$V_{machine}(f, N_{bandwidth}) = (t_c, t_m, t_{msg}, t_{Byte}, P_{total-idle}, \Delta P_c, \Delta P_m) \quad (4.2)$$

The application-dependent variable vector can be described as a function of the amount of parallelism available and the workload for the application:

$$V_{app}(p, n) = (\alpha, W_c, W_m, W_{co}, W_{mo}, M, B) \quad (4.3)$$

In the next Section, we are going to conduct detailed mathematical derivation of the model. Then, we will provide methodology followed by model validation and accuracy analysis of equation (4.1).

4.4 I-E-E Model Detailed Derivation

In this section we describe the details for deriving the *iso-energy-efficiency* model first presented in the previous subsection.

A. Performance Model:

At the system level, the theoretical sequential execution time for an on-chip/off-chip workload comprises three components [21, 187]: computation time $W_c t_c$ (with on-chip instruction execution frequency), main memory access latency $W_m t_m$, and I/O access time T_{IO} (with off-chip instruction execution frequency). Thus the theoretical execution time can be expressed as:

$$T = W_c t_c + W_m t_m + T_{IO} \tag{4.4}$$

Since optimization techniques could raise various levels of overlap between components [188], we multiply *T* by a corrector factor α ($\mathbf{0 \leq \alpha \leq 1}$):

$$T' = \alpha T = \alpha(W_c t_c + W_m t_m + T_{IO}) \tag{4.5}$$

where T' *is the actual execution time.*

B. Energy Model for one and p parallel processor(s):

When executing a parallel application, total energy consumption can be divided into four parts: computation energy, E_c main memory access energy, E_{mem}, I/O access energy, E_{IO}, and other system components energy, E_{other}, such as motherboard, system and CPU fans, power supply, etc. Thus, we have total energy *E* [6]:

$$E = E_c + E_{mem} + E_{IO} + E_{other} \quad (4.6)$$

The first three parts of this equation can be further separated into two energy states: running state and idle state. For example, E_c can be divided into E_{c-on} and E_{c-idle}. Thus, we can deduce total energy *E* as [52, 53]:

$$E = E_{c-on} + E_{c-idle} + E_{mem-on} + E_{mem-idle} + E_{IO-on} + E_{IO-idle} + E_{other} \quad (4.7)$$

From (4.5) and (4.7),

$$E = \alpha TP_{total-idle} + W_c t_c \Delta P_c + W_m t_m \Delta P_m + T_{IO} \Delta P_{IO}$$

(4.8)

Where $W_c t_c$ is the total computation time; $W_m t_m$ is the total memory access time and T_{IO} is the total I/O access time.

$$\Delta P_c = P_{c-on} - P_{c-idle}; \quad \Delta P_m = P_{m-on} - P_{m-idle}; \quad \Delta P_{IO} = P_{IO-on} - P_{IO-idle}.$$

Equation (4.8) seems quite cumbersome; however, it is intuitive: $\alpha TP_{total-idle}$ is the total energy consumption of an idle-state system during an application's execution time. $W_c t_c \Delta P_c$ is the additional energy used while an application is performing computation. Similarly, $W_m t_m \Delta P_m$ and $T_{IO} \Delta P_{IO}$ are the additional energy consumption for conducting main memory and I/O accesses.

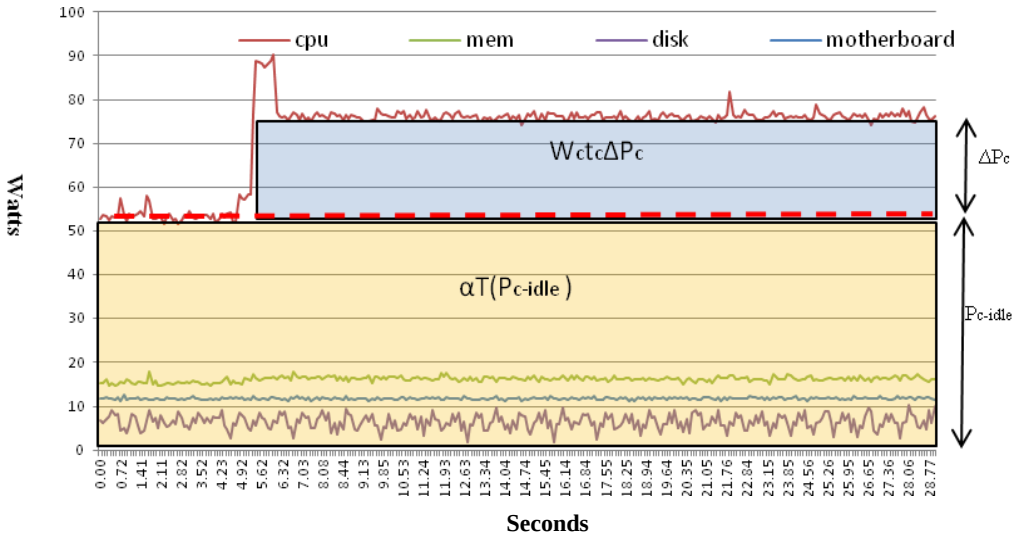


Figure 4.2: Power Profiling of MPI_FFT program in HPCC Benchmark.

Figure 4.2 provides additional insight to Equation (4.8). It shows the power profiling of the *MPI_FFT* program in the HPC Challenge Benchmark [55] measured by the *PowerPack* framework. The power fluctuates for each component over the idle-state power line (dashed line) during the execution time. For the CPU, the red shaded (lower) portion in Figure 4.2 represents total CPU energy consumption in idle-state, and the blue (upper) portion represents the additional energy while doing computation.

In reality, I/O access time includes the network and all kinds of local storage devices accesses. If an application is disk I/O-intensive, it should introduce T_{disk} to the performance and the energy model. For simplicity, we assume a simple, flat model for I/O accesses though the benchmarks we measured did not exercise I/O making this component effectively zero. Users can always replace $T_{IO}\Delta P_{IO}$ with any combinations of specific I/O components according to their parallel applications. Demonstrating the

accuracy of the model for all types of I/O is beyond the scope of this work and the subject of future work.

The equations follow similar to Equation (4.5):

$$T' = \alpha T = \alpha(W_c t_c + W_m t_m + T_{net})$$

With energy model:

$$E = \alpha T P_{total-idle} + W_c t_c \Delta P_c + W_m t_m \Delta P_m + T_{net}(P_{net-on} - P_{net-idle}) \quad (4.10)$$

In our experiments (on both the *Dor*i system with Ethernet and *SystemG* with InfiniBand), the difference between P_{net-on} and $P_{net-idle}$ is not significant so we simply ignore the effect $T_{net}(P_{net-on} - P_{net-idle})$ in equation (4.10):

$$E = \alpha T(P_{total-idle}) + W_c t_c \Delta P_c + W_m t_m \Delta P_m \quad (4.11)$$

C. Energy Model for A Single Processor:

Equations (4.9) and (4.11) are the kernel components of the *performance model* and *iso-energy-efficiency model* in this work. Let us apply these to E_1 which we discussed in Section 4.3. When an application executes on a single processor, there are no messages exchanged. This means no T_{net} in (4.9). Thus, E_1 **becomes:**

$$E_1 = \alpha T_1(P_{total-idle}) + W_c t_c \Delta P_c + W_m t_m \Delta P_m \quad (4.12)$$

where $T_1 = (W_c t_c + W_m t_m)$

D. Energy Model for p Parallel Processors:

Similarly, to get E_p, we define the energy model in *i*th ($1 \le i \le p$) processor among *p* parallel processors:

$$E_{p_i} = \alpha T_{p_i}(P_{total-idle}) + \left(w_{c_i} + w_{co_i}\right) t_c \Delta P_c + \left(w_{m_i} + w_{mo_i}\right) t_m \Delta P_m \qquad (4.13)$$

where $T_{p_i} = \left[\left(w_{c_i} + w_{co_i}\right) t_c + \left(w_{m_i} + w_{mo_i}\right) t_m + T_{net_i}\right]$ _()

In (4.13), w_{co_i} and w_{mo_i} are computation and memory access overheads for the *i*th of *p* processors in terms of parallelism. Thus, we have E_p representing total energy consumption for all processors:

$$E_p = \sum_{i=1}^{p} E_{p_i} = \alpha T_p(P_{total-idle}) + (W_c + W_{co}) t_c \Delta P_c + (W_m + W_{mo}) t_m \Delta P_m \qquad (4.14)$$

where $T_p = \sum_{i=1}^{p} T_{p_i} = \left[(W_c + W_{co}) t_c + (W_m + W_{mo}) t_m + \sum_{i=1}^{p} T_{net_i}\right]$

__D_Dd_____ŷá22_____ _ 〖_〗 __ (_())_ _ __ _ _____

From equation (4.1) we can calculate the energy overhead E_o:

$$E_o = E_p - E_1 = \alpha T_o(P_{total-idle}) + W_{co} t_c \Delta P_c + W_{mo} t_m \Delta P_m \qquad (4.15)$$

$$where\ T_o = \left(W_{co} t_c + W_{mo} t_m + \sum_{i=1}^{p} T_{net_i}\right)$$

In equations (4.14) and (4.15), W_{co} is the total parallel computation overhead ($W_{co} = \sum_{i=1}^{p} w_{co_i}$) and W_{mo} represents the total parallel memory access overhead

($W_{mo} = \sum_{i=1}^{p} w_{mo_i}$). $\sum_{i=1}^{p} T_{net_i}$ stands for accumulated networking time. $\sum_{i=1}^{p} T_{net_i}$ can be further divided into two parts: message start up time and data transmitting time [189]. Communication overhead modeling varies depending on application and network infrastructure. Equation (4.16) is a general approach and specific parameterization for network modeling is applied for each application (see *FT* example in the later section).

$$\sum_{i=1}^{p} T_{net_i} = Mt_{msg} + Bt_{Byte} \tag{4.16}$$

So that E_o can be expressed as:

$$E_o = \alpha T_o P_{total-idle} + W_{co} t_c \Delta P_c + W_{mo} t_m \Delta P_m \tag{4.17}$$

$$where\ T_o = (W_{co} t_c + W_{mo} t_m + Mt_{msg} + Bt_{Byte})$$

E. Energy Efficiency Factor (EEF):

Using the Equations (4.12) and (4.17), we can formulate the Energy Efficiency Factor *(EEF)* more accurately,

$$EEF = \frac{E_o}{E_1} = \frac{\alpha T_o (P_{total-idle}) + W_{co} t_c \Delta P_c + W_{mo} t_m \Delta P_m}{\alpha T_1 (P_{total-idle}) + W_c t_c \Delta P_c + W_m t_m \Delta P_m} \tag{4.18}$$

$$where\ T_o = (W_{co} t_c + W_{mo} t_m + Mt_{msg} + Bt_{Byte})$$
$$T_1 = (W_1 c\ t_1 c + W_1 m\ t_1 m\ )$$

Equation (4.18) contains two categories of parameters which directly impact performance and energy consumption: 1) machine dependent variables $t_c, t_m, t_{msg}, t_{Byte}, P_{total-idle}, \Delta P_c, \Delta P_m$. and 2) application dependent variables:

$\alpha, W_c, W_m, W_{co}, W_{mo}, M,$ and *B*. For the application dependent vector, V_{app}, the processor number *p* and problem size *n* are two main factors affecting these parameters.

The values of W_c, W_m, W_{co}, W_{mo} can be obtained by the combination of analyzing an application's algorithm and directly measuring the specific performance counters to estimate the on-off chip workload. Also, *M* and *B* can be acquired by using PMPI in MPICH2 [190] or TAU[191]. The corrector factor α can be estimated using:

$$\alpha = \frac{actual\ execution\ time}{W_c t_c + W_m t_m + T_{net}} \tag{4.19}$$

The machine dependent vector can be represented as:

$$V_{machine}(f, N_{bandwidth}) = (t_c, t_m, t_{msg}, t_{Byte}, P_{total-idle}, \Delta P_c, \Delta P_m)$$

For machine dependent variables, machine frequency f and the network bandwidth, $N_{bandwidth}$, are the main factors affecting these parameters. For the time parameters, t_c is $\frac{CPI_{on}}{f}$. t_m and t_{msg} can be also described as functions of f. Only t_{Byte} is related with the network bandwidth. From Kim et al [67, 68]:

$$f_{max} \propto \frac{(V_{dd} - V_{th})^{\beta}}{V_{dd}}$$
$$P = P_{dyn} + P_{leak} = ACV_{dd}^2 f + I_{leak} V_{dd} \tag{4.20}$$

We can assume $P_{total-idle}, \Delta P_c, \Delta P_m$, are also functions of f. Here we assume power is proportional to f^{γ} ($\gamma \geq 1$). We use the correlation between power and frequency in our energy model to predict total energy consumption and energy efficiency of large scale parallel system.

From Equations (4.1) and (4.18), the *iso-energy-efficiency* model for parallel applications can be defined as:

$$EE = \frac{1}{1 + EEF} = \frac{1}{1 + \frac{\alpha T_o(P_{total-idle}) + W_{co} t_c \Delta P_c + W_{mo} t_m \Delta P_m}{\alpha T_1(P_{total-idle}) + W_c t_c \Delta P_c + W_m t_m \Delta P_m}} \quad (4.21)$$

$$\text{where } T_o = (W_{co} t_c + W_{mo} t_m + M t_{msg} + B t_{Byte})$$
$$[T]_1 = (W_1 c\, t_1 c + W_1 m\, t_1 m)$$

In equation (4.21), *EEF* is a combination of machine and application dependent parameters. To maximize the system energy efficiency, we need to keep *EEF* as small as possible by scaling characteristics such as degree of parallelism, workload, processor frequencies and network bandwidth.

F. Corrector factor:

Accurately capturing performance characteristics is critical to a model of *iso-energy-efficiency*. Early on in our attempt to create an *iso-energy-efficiency* model we realized computational overlap, or the ability to conduct computations while waiting on memory or communication delays, could not be ignored since they can reduce execution time dramatically [188]. The amount of overlap varies with an application, the underlying machine architecture, and compiler settings.

In addition to the overlaps described above, extra costs such as parallel overhead data transmission between nodes, synchronization overhead due to load imbalance and serialization, and the unexpected extra computation overhead caused by parallel scheme and task assignment also contribute as parts of inaccuracy of modeling performance. Thus we propose a comprehensive corrector factor, α, to capture the effects above and help

adjust the time obtained by model to actual execution time. For simplicity, we estimate α as:

$$Theoretical\ Execution\ Time = Computation\ Time + Memory\ Access\ Time + Network\ Transmission\ Time$$
$$\alpha = Actual\ Execution\ Time / Theoretical\ Execution\ Time$$

For parallel applications, we found empirically for the applications studied that an application using the same compiler settings has the same α value under different levels of parallelism. However, different applications could have different α due to different execution patterns. In addition, same applications running on different machines also have different α values because of diverse underlying architectures.

4.5 Methodology

In this work we focus on using the I-E-E model to show the affects of problem size or frequency scaling on maintaining or improving system-level energy efficiency (other system level parameters can also use similar methodology). To do this we need to solve two correlation functions when system scales:

$$N = F(p, \theta)\ ,\ under\ fixed\ processor\ frequency\ f. \quad (4.22)$$

$$f = F(p, \theta)\ ,\ under\ fixed\ problem\ size\ N \quad (4.23)$$

$F(p, \theta)$ in (4.22) and (4.23) is a function of p with user desired energy efficiency θ as a constant. Using the model (4.1), we find what the effect (improve, degrade or no effect) on system energy efficiency is when scaling problem size or

processor frequency while *p* scales up for a specific application.

When you know effects of problem size or frequency scaling on energy efficiency for a specific application, we then find valid *N* or *f* values to maintain or improve energy efficiency θ under specific p . In the following case studies we show how to use equations (4.1) (4.22) (4.23) to evaluate the energy efficiency of several commonly used parallel applications.

A. Software:

Precisely measuring the machine and application parameters such as total on-chip computation workload is the key to building highly accurate model. For a complete list of dependent parameters see Table A in Appendix. Measuring large numbers of parameters by hand becomes labor intensive and error prone. To solve this problem we implemented a program called PowerScale to automate the process.

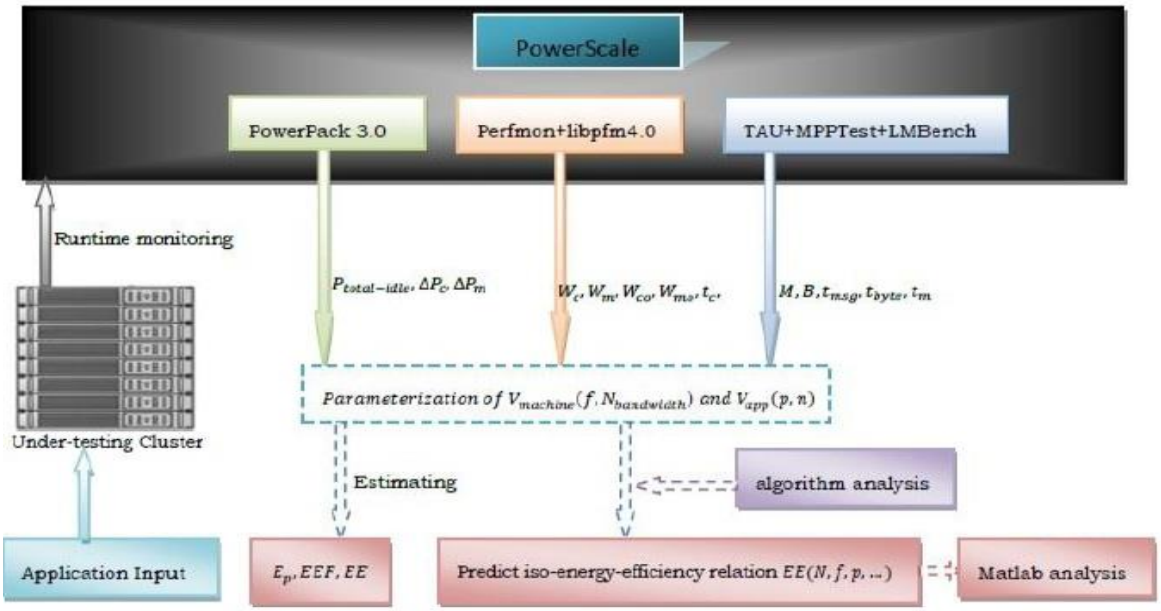


Figure 4.3: Runtime monitoring tool PowerScale software components and data flow diagram.

Figure 4.3 shows our semi-automatic runtime monitoring tool PowerScale and its major software components. Table A in Appendix is annotated to show parameters automatically measured by PowerScale. PowerScale is architecture independent because it will map the architecture specific hardware counters to the correct model parameters. Hardware counters are able to capture effects such as increased memory controller contention caused by a different number CPU cores. Details will be shown in Test Environment and Model Validation Section.

B. The estimating procedure for N and f:

```
01: Collect and process the runtime monitoring data and
    machine/application dependent parameters used for
    model.
02: Plug the parameters into the I-E-E model.
03: Pre-define the value of θ ( it can be a loop with a
    group of interested  θ values)
04: for p=1 to (max range of p user wants to evaluate) do
05:   Try to solve N=F(p,θ) under fixed frequency
06:   if !N or N or out of range then
07:       Give value -1 to data structure *A [p]
          and return to the for loop to continue
08:   else *A [p]=1 and *(A [p]-> next)=N
09:   end if
10: end for
11: Find the problem sizes with *A [p]=1 and output their
    *(A [p]->next) with corresponding p
```

Figure 4.4: Pseudo code for estimating N in order to maintain energy.

As explained in earlier we define *N* as problem size and *f* as CPU frequency. In order to maintain system energy efficiency at the user desired level we can estimate *N*

and f for a system size (p).

Figure 4.4 shows the pseudo code for estimating problem size N in order to maintain energy efficiency while p scales up, described in functions (4.22) (4.23) can use a similar method to estimate f. However, finding a frequency level to maintain energy efficiency while system scales is sometimes impractical due to the limitations of DVFS. More discussion about the frequency scaling approach can be found in the Case Study Section.

4.6 Test Environment and Model Validation

A. Experimental setup:

Table 4.1 shows the configurations of two power-aware clusters: *SystemG* and *Dori*. Most of the experiments and modeling were conducted on SystemG. Some validation comparison tests are done on *Dori*.

Table 4.1
System Configurations for SystemG and Dori Clusters

Cluster	SystemG	Dori
System size (nodes)	325	8
Processor	2 x Quad-core Xeon	2 x Dual-core Opteron
Memory	8GB	6GB
L1 cache	32KB	64KB
L2 cache	6MB	1MB
Interconnection	40Gbytes/s InfiniBand	1Gbytes/s Ethernet
CPU frequencies	2.8, 2.4 GHz	1.8, 1.6, 1.4, 1.2, 1.0 GHz

B. Model Validation:

To validate the *iso-energy-efficiency* model, we need to verify the correctness of the model on single and parallel processor configurations. We vigorously measure and derive the parameters from Table 3 in Appendix; namely the machine and application dependent parameters.

For the machine-dependent parameters, we built a tool using the *Perfmon* API from UT-Knoxville to automatically measure the average t_c (time per on-chip computation instruction) derived as $\frac{CPI_{on}}{f}$. We use the *lat_mem_rd* function from the *LMbench* microbenchmark [192] to estimate memory costs t_m. t_{msg} and t_{Byte} are obtained by using the *MPPTest* tool [190] for both the InfiniBand [193] and Ethernet interconnects in the two clusters. In addition, $P_{total-idle}$, ΔP_c and ΔP_m can be obtained by using *PowerPack* [52]. We did not include disk I/O in our estimations for our energy efficiency model because the applications we tested are not disk intensive. We leave this to future work. For completeness, though it is not used in the current study, we were able to estimate T_{IO} by using the Linux pseudo file /proc/stat.

For the application-dependent parameters, we build a workload and overhead model for each parameter by analyzing the algorithm and measuring the actual workload for each application. We use *Perfmon* to measure each workload parameter, W_c, W_m, W_{co}, W_{mo} and we use the *TAU* [194] performance tool from the University of Oregon to measure *M* and *B.* Fig. 16 illustrates the accuracy of the energy model for *P* processors. (Note: Specifically, these results are for Equation (4.14)).

Figure 4.5 compares the energy consumption predicted by the *iso-energy-efficiency* model with the actual energy consumption obtained using the *PowerPack* framework on *Dori* cluster for *P*=4. We repeated all the experiments to reduce measuring

errors. The results indicate that the proposed energy model can accurately predict the energy consumption within 5% prediction error. We conducted similar experiments on *SystemG* for *P*=1, 2, 8, 16, 32, 64, 128. Figure 4.6 shows the average error rate of *EP*, *FT*, and *CG* applications on *SystemG* under different levels of parallelism using the InfiniBand interconnect. The results show good accuracy. Upon detailed analysis, the relatively higher errors (8.31%) found with *CG* were due to: 1) The current *CG* memory model is based on analysis of the *CG* program and its memory behavior across different parallel platforms; 2) Our measurements show that the *CG* memory workload decreases substantially while scaling up on the *SystemG* system. This appears due to the fact that

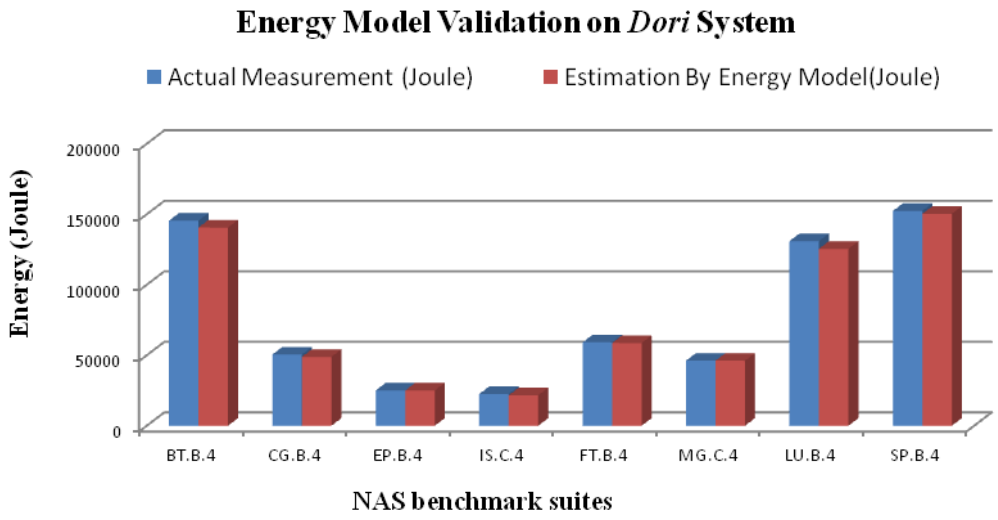


Figure 4.5: Model validation on Dori system. All the applications run on 4 nodes under same CPU clock frequency. Model accuracy for all the benchmarks are over 95 %.

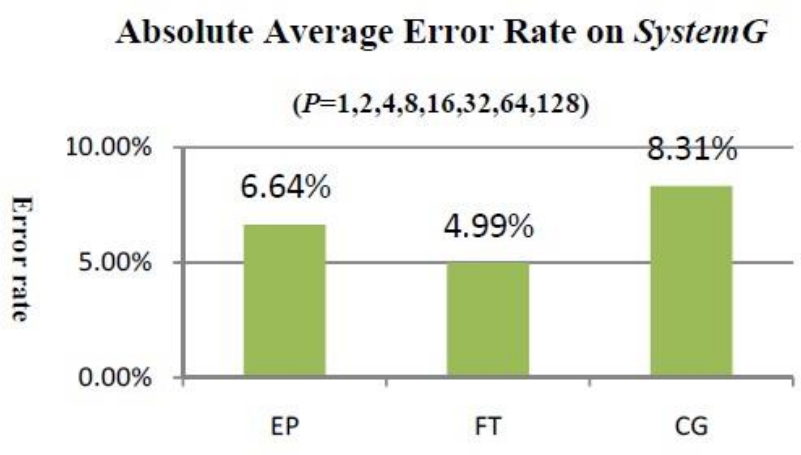


Figure 4.6: The average error rate of EP, FT and CG program. Class=B in node number p= 1,2,4,8,16, 32,64,128.

CG favors both temporal and spatial locality as a system scales. Improving the accuracy for *CG* is the subject of future work.

Based on the accuracy results for both *SystemG* and *Dori* clusters, we conclude that our *iso-energy-efficiency* model performs well on different network interconnection infrastructures and can predict total system energy consumption with an average of 5% prediction error rate for parallel applications with various execution and communication patterns.

4.7 Case Studies and Discussion

First, before start our case studies, we want to talk about energy consumption and efficiency prediction for large scale systems. Given the accuracy of our modeling techniques as described in the previous section, we use measurements from smaller configurations to predict and analyze power-performance tradeoffs on larger systems.

Initially, we obtain machine-dependent variables from the smaller system and use these values and our models to predict values for increasing number of nodes:

$$V_{machine}(f, N_{bandwidth}) = \quad (t_c, t_m, t_{msg}, t_{Byte}, P_{total-idle}, \Delta P_c, \Delta P_m)$$

All variables can be measured as described in the previous section. Frequency-dependent variables can be combined by normalizing measurements obtained through the use of hardware counters, *LMbench*, *MPPTest* and *Powerpack*. For example, t_c can be described as $\frac{11.9}{f} \times 10^{-10}$ sec on *SystemG*. We assume power is proportional to f^{γ} ($\gamma \geq 1$).

Next, we model application-dependent variables from the smaller system:

$$V_{app}(p, n) = (\alpha, W_c, W_m, W_{co}, W_{mo}, M, B)$$

Except for α, all of these variables in $V_{app}(p, n)$ depend on a performance model and can be described as a function of problem size, *n*, and the level of parallelism, *p*. For example, W_{co} could be described as $n \log_2 p$ in one-dimensional, unordered and radix-2 binary exchange Fast Fourier Transform. With all parameters accounted for, we can solve for equation (4.23). We can then project these parameters for larger value of *p* to predict the power-performance behavior and tradeoffs of large scale systems.

In the rest of this section, we will analyze the power-performance characteristics of FT, EP, CG, high performance LINPACK (HPL), and row-oriented Matrix Multiplication. We plant to isolate power-performance efficiency problems and use the

model findings to tune system-level parameters to maintain or improve efficiency. Therefore, the following studies are focused on two aspects: problem size and frequency scaling while system size scales up.

In each case, we use the methods described in the previous sections to obtain model parameters and build our model from measurements on a smaller system. Once we've identified estimates for $V_{machine}(f, N_{bandwidth})$ and $V_{app}(p, n)$ vectors, we build I-E-E model as described in equation (4.22). In the rest of this section, all the parameterizations are obtained for the *SystemG* cluster though the same methodology can be applied to other platforms (Our model uses multiple hardware counters to estimate on-off chip workload pattern and execution time. Underlying architecture level changes are currently hidden in the hardware counter driven model building. So the changes of CPU and Memory level activities will be captured).

A. Problem size scaling:

For scaling problem size *N* to achieve constant energy efficiency, we derive from the general form of the I-E-Emodel (4.1):

$$EE(N,p) = \theta = \frac{E_{1,N}}{E_{p,N}} \tag{4.24}$$

(*where* $0 < \theta \leq 1$, $0 < N \leq total\ allowable\ memory$ *for* p *processors,* $E_{1,N}$ *and* $E_{p,N}$ *are the system energy consumption for running on one and p procssors with problem size N. For simpilicity, CPU has fixed frequency.*)

In this subsection, we will apply equation (4.24) to five parallel applications with different runtime execution patterns and discuss how to solve (4.22) in order to estimate problem size N for specific parallelism p to maintain system-wide energy efficiency θ under fixed frequency. Frequency f is fixed at 2.8 GHz. For the simplicity of modeling network related parameters M and B, we assume a one port communication model and bi-directional communication links. The applications used are:

- EP from NPB (Embarrassingly Parallel)
- CG from NPB (non-ignorable communication)
- High Performance LINPACK (CPU and Memory intensive with non-ignorable communication);
- Simple row-oriented matrix multiplication (high computation to communication ratio with a large memory footprint);
- 3D Fourier transform from NPB (communication intensive).

We will also use these applications to categorize common parallel applications based on their individual execution pattern and overall increasing rate of N for maintaining energy efficiency while system scales up.

I) Case A – Embarrassingly Parallel (EP) :

In parallel computing, an embarrassingly parallel (*EP*) workload has little inter-processor communication between parallel processes. *EP* in the NPB benchmarks generates pairs of Gaussian random deviates using *Marsaglia polar method*. It separates tasks with little or no overhead. Results of *EP* can also be considered as a reference of

peak performance of a given machine. We use our measuring tools, *MPPTest* and the *PowerPack* framework to obtain the machine dependent parameters:

$$V_{machine-EP}(f, N_{bandwidth}) = (t_c, t_m, t_{msg}, t_{Byte}, P_{total-idle}, \Delta P_c, \Delta P_m) = \left(\frac{11.9}{f} * 10^{-10},\quad 1.12 * 10^{-7},\quad 2.53 * 10^{-5},\quad 1.82 * 10^{-8},\quad 18.9f^2,\quad 2.67f^2,\quad 1.52f^2\right)$$

After analyzing the parallel *EP* codes, we have:

$$V_{app-EP}(p,n) = (\alpha, W_c, W_m, W_{co}, W_{mo}, M, B)$$

$$= (0.93,\ 109.4*n,\ 1.03* 10^{-6} *n,\ 0,\ 6.7* 10^{-7} *n *(p-1),\ 0,\ 0)$$

Since communication in embarrassingly parallel is trivial, we simply set *M* and *B* to zero in $\mathbf{V_{app-EP}(p,n)}$. Thus, from Equation (5.2.16), we have EEF_{EP} :

$$EEF_{EP} = \frac{E_o}{E_1}$$

$$= \frac{\alpha T_o(P_{total-idle}) + W_{co} t_c \Delta P_c + W_{mo} t_m \Delta P_m}{\alpha T_1(P_{total-idle}) + W_c t_c \Delta P_c + W_m t_m \Delta P_m} = \frac{1.43(p-1)f^2}{2.63 * 10^6 f + 2.2f^2}$$

So EE_{EP} becomes:

$$EE_{EP} = \frac{1}{1 + \frac{1.43(p-1)f^2}{2.63 * 10^6 f + 2.2f^2}}$$

Since this is nearly *ideal iso-energy-efficiency*, we cannot improve the energy efficiency by scaling problem size *n* at all because E_o increases as fast as E_1 .

2) Case B --- CG:

The NAS *CG* benchmark evaluates a parallel system's computation and communication performance. It uses the conjugate gradient method to find out the smallest eigenvalue of a large, sparse matrix. It solves a sparse linear algebra problem which is common to scientific applications on large-scale systems. We first obtain the machine-depended parameters using the previous methods:

$$V_{machine-CG}(f, N_{bandwidth}) = (t_c, t_m, t_{msg}, t_{Byte}, P_{total-idle}, \Delta P_c, \Delta P_m) = \left(\frac{1.14}{f} * 10^{-9}, 1.12 * 10^{-7}, 2.53 * 10^{-5}, 1.82 * 10^{-8}, 24f^2, 4.57f^2, 1.25f^2\right)$$

For the application-dependent parameters we obtain:

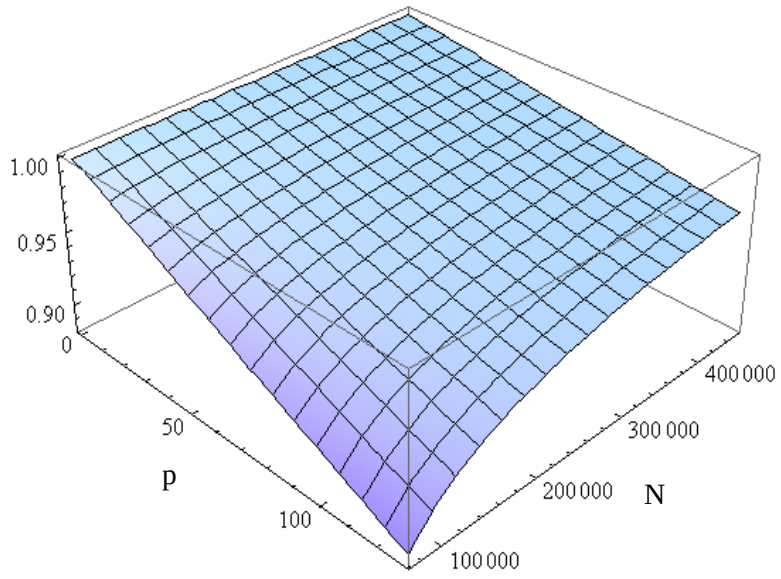


Figure 4.7: 3D plot of $\mathbf{EE_{CG}}$, Assume frequency f=2.8GHz, with p and n as variables.

$V_{app-CG}(p,n) = (\alpha, W_c, W_m, W_{co}, W_{mo}, M, B) = (0.85,\ 2.13 * 10^5 n^{1.25},\ 0.96\ n^{1.75},\ 1.86 * 10^6 n^{0.7} * 2^{1.1(log_2 p-1)}, -4.75 n^{0.5} * 2^{0.14*[(log]_2 P-1)^2}, 0, 0)$

Thus, we solve for EEF_{CG}:

$$EEF_{CG} = \frac{E_o}{E_1} = \frac{\alpha T_o(P_{total-idle}) + W_{co} t_c \Delta P_c + W_{mo} t_m \Delta P_m}{\alpha T_1(P_{total-idle}) + W_c t_c \Delta P_c + W_m t_m \Delta P_m} =$$

$$\frac{5.3 * 10^{-2} f n^{0.7} * 2^{1.1(log_2 p-1)} - 1.15 * 10^{-3} f^2 n^{0.5} * 2^{0.14*[(log]_2 P-1)^2}}{6.07 * 10^{-3} f n^{1.25} + 2.33 * 10^{-6} f^2 n^{1.75}},$$

and then for EE_{CG}:

$$EE_{CG} = \frac{1}{1 + \frac{5.3 * 10^{-2} f n^{0.7} * 2^{1.1(log_2 p-1)} - 1.15 * 10^{-3} f^2 n^{0.5} * 2^{0.14*[(log]_2 P-1)^2}}{6.07 * 10^{-3} f n^{1.25} + 2.33 * 10^{-6} f^2 n^{1.75}}}.$$

From EEF_{CG} and EE_{CG}, we plot the relationships between level of parallelism, *p*, and problem size, *n*. In Fig. 18, we first fix the frequency *f* at 2.8 GHz to examine the relation between *p* and *n*. We notice that the energy efficiency decreases as *p* increases. However, increasing the workload size, *n*, will improve the energy efficiency.

3) Case C --- High Performance LINPACK (HPL):

High Performance LINPACK is widely used in the HPC community to measure peak system performance. It is also the sole performance benchmark used by the TOP

500 list [56] to rank the world's fastest supercomputers. HPL is a portable benchmark that solves a (random) dense linear system ($Ax = b$) in double precision (64 bits) arithmetic on distributed-memory computers. The problem size (N) specifies the order of matrix A (therefore A has $N * N$ elements). In order to solve x, we first need to apply LU factorization algorithm to the coefficient matrix [A, b] (shown in Figure 4.8) and then use backward substitution method to achieve x. Figure 4.8 shows that the coefficient matrix has been logically divided into block size $NB*NB$ (computational granularity) and then distributed on a P*Q (here 2*4) grid of processes for LU factorization. HPL system configuration has several parameters affecting overall maximum performance in terms of *Gflops*, including problem size N, block size NB, rows of process grid P and columns of process grid Q. Compared to these four parameters, other HPL parameters are considered to have a trivial impact on maximum performance . In our modeling process shown in

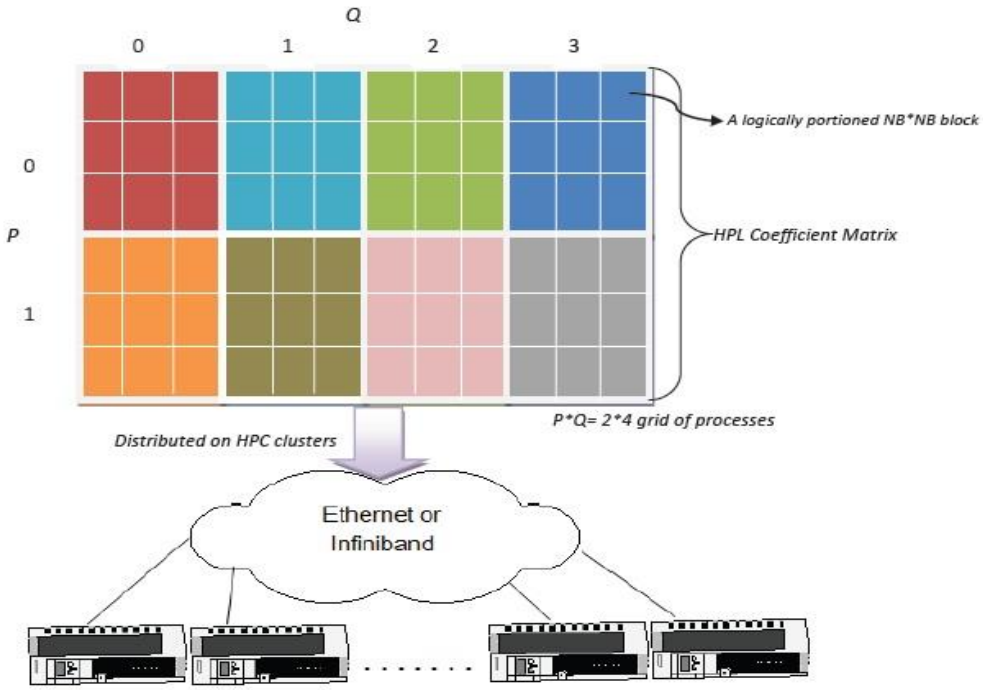


Figure 4.8: Data distribution for HPL on the logical Process Grid.

Table 4.2, NB, P and Q are involved in the network performance related parameters M and B. Since the focus of this case is to show how scaling problem size N to maintain or improve system-wide energy efficiency instead of performance maximization, we fix the value of NB to 32 and set P and Q approximately equal, with Q slightly larger than P according to *p*: P*Q = *p* and P$\leq$ Q. Work such as [195] discuss how to finely tune NB, P and Q to maximize the performance benefit for both data distribution and parallel computation. Detailed discussion of these three parameters is beyond the scope of this prelim work.

Correlation (4.22) can help us estimate the value of N according to the number of

Table 4.2: Parameters' estimation for HPL

Machine	Estimation
t_c	$\frac{4.31}{f} * 10^{-10}$
t_m	$1.12 * 10^{-7}$
t_{msg}	$2.53 * 10^{-5}$
t_{Byte}	$1.82 * 10^{-8}$
$P_{(total-idle)}$	$27.68 * f^{\gamma}$ †
ΔP_c	$2.19 * f^{\gamma}$
ΔP_m	$1.56 * f^{\gamma}$

Application	Estimation
α	0.989
W_c	$107927 * N^2$
W_m	$14.6 * N^2$
W_{co}	$3600 * log_2^p N^2$
W_{mo}	$(25.4 * N^2 * log_2^p)/p$
M	$N * ((NB+1) * log_2^p + p)/NB$
B	$16 * N^2/p^{1/2}$

† For *SystemG* we fix γ to 2.

processes p and desired energy efficiency under fixed frequency. Users can preset θ based on (4.24) function under an energy efficiency upper bound for a specific number of *p*. In (4.24), the memory upper bound we set for N in HPL is around 80% of total parallel system memory to avoid memory swapping and TLB misses. Table 4.2 shows the machine and application dependent parameters estimated by PowerScale and manual analysis of HPL. Based on the parameter estimation in Table 4.2 and equations (4.1) (4.24), we are able to build the I-E-E model for HPL using a fixed frequency of 2.8GHz.

Based on Table 4.2, we can plot energy efficiency scaling for HPL under fixed CPU frequency shown in Figure 4.9. Figure 4.9 shows that when we scale up the system size *p*, the overall system energy efficiency degrades under fixed frequency. After following the procedure in Figure 4.4, we discover that there is no possible rate to scale N in order to keep up with increasing *p* to maintain energy efficiency θ . In other words,

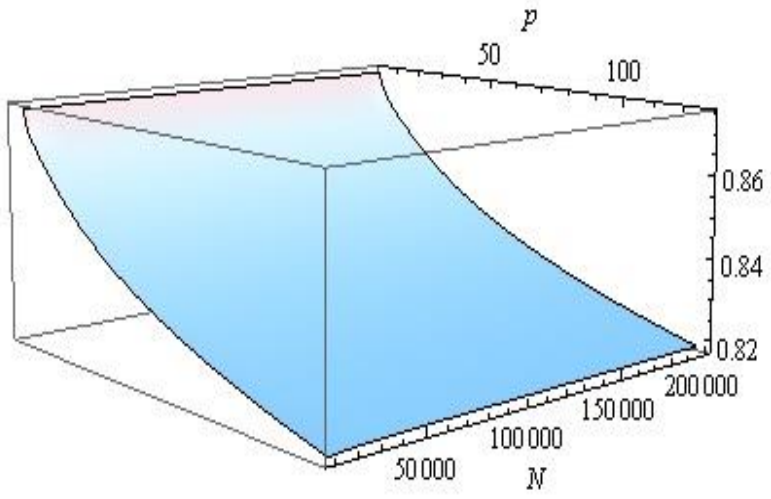


Figure 4.9: 3-D illustration of energy efficiency scaling for HPL while scaling p (from 4 to 128) and problem size N.

overall energy efficiency does not react to more workload while system scales up, so the

problem size scaling approach for improving energy efficiency is not feasible for HPL. This observation also shows that, for HPL, energy is more efficiently used for parallel execution if we increase problem size to the upper bound with the fixed p since energy efficiency stays approximately constant while N scales. HPL is a typical weak scaling case stressing both CPU and memory, and its energy efficiency pattern using scaling N will be categorized at the end of this section.

4) Case D --- Row-Oriented Matrix Multiplication (ROMM):

In this case study, we demonstrate how to use our I-E-E model to estimate N for ROMM in order to maintain constant system-wide energy efficiency. ROMM has a high computation to communication ratio with a large memory footprint. To simplify our analysis, we assume that A, B and C are all *n*n* matrices and the total number of computations is $N = 2n^3$ (including both additions and multiplications).

Table 4.3: Parameters' estimation for ROMM

Machine	Estimation
t_c	$\frac{3.64}{f} * 10^{-10}$
t_m	$1.12 * 10^{-7}$
t_{msg}	$2.53 * 10^{-5}$
t_{Byte}	$1.82 * 10^{-8}$
$P_{(total-idle)}$	$27.68 * f^2$
ΔP_c	$1.43 * f^2$
ΔP_m	$0.99 * f^2$

Application	Estimation
α	0.854
W_c	$26.1 * N$
W_m	$4.35 * 10^{-2} N$
W_{co}	$2.5 * 10^{-3} Np$
W_{mo}	$6.17 * 10^{-3} * \frac{N}{(p-1)}$
M	$2.53 * 10^{-5} * (p-1) N^{\frac{2}{3}}$
B	$(p-1)(1 + N^{\frac{2}{3}} + \frac{2}{p} * N^{\frac{2}{3}})$

Table 4.3 shows the machine and application dependent parameters for SystemG under a fixed frequency of 2.8GHz. Based on this, we can derive ROMM's I-E-E model. Figure 4.10 is the rendered image of the I-E-E model of ROMM. We now use the pseudo code from Figure 4.4 to solve I-E-E correlation $N_{ROMM} = F(p, \vartheta)$. For example, we set $\vartheta = 0.85$ and $\vartheta = 0.98$ with solutions:

$$\vartheta = 0.85 = EE_{ROMM}(N, p) = EE_{ROMM}(1.3824 * 10^{10}, 64) = EE_{ROMM}(1.1128 * 10^{11}, 128)$$

$$\vartheta = 0.98 = EE_{ROMM}(1.3824 * 10^{10}, 8) = EE_{ROMM}(1.8432 * 10^{10}, 16) = EE_{ROMM}(3.093 * 10^{10,32})$$

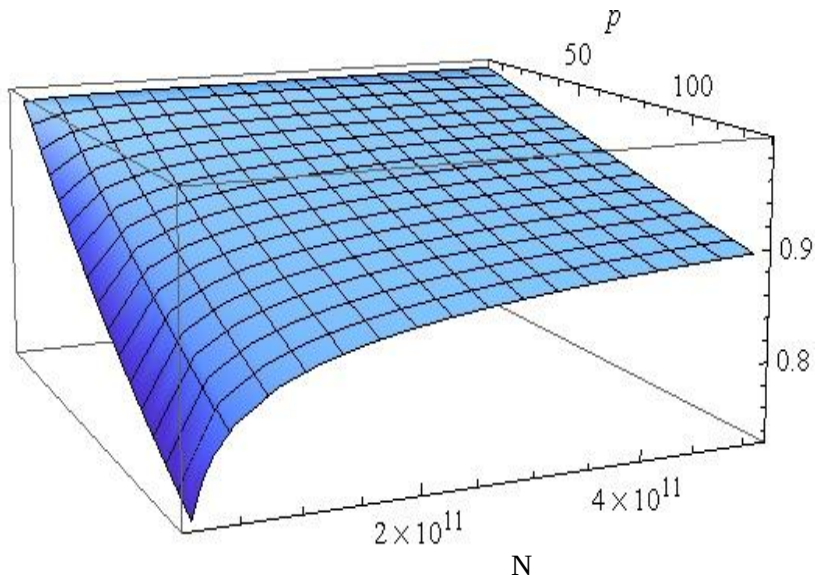


Figure 4.10: 3-D demonstration of ROMM energy efficiency trend while scaling number of *p* and problem size *N on SystemG.*

Using Table 4.3, we can approximately estimate the Iso-energy-efficiency

relationship between N and p : $N_{ROMM} = f'(\theta)\varphi(\log p)$, where $f'(\theta)$ is a function of θ and $\varphi(x)$ is the set of all functions that have the same growth rate as x . From the equation, we can infer that the total problem size N needs to grow with the number of processors at an overall rate of $\varphi(\log p)$ in order to maintain system-wide desired energy efficiency θ ($N > 0$). So ROMM's energy efficiency scalability is less than Embarrassingly Parallel (EP) which is an approximately ideal case for Iso-energy-efficiency but better than linear growth applications. This shows that the problem size scaling approach can be used to help maintain energy efficiency for ROMM. For performance efficiency analysis, we use Michael J. Quinn's [73] performance isoefficiency relation to evaluate ROMM. We find that in order to maintain a constant level of performance efficiency, memory utilization per processor needs to increase as $\varphi(p)$. We conclude that in general, system energy efficiency is a combination of the effects of power and performance and so may not be evaluated by only using basic performance efficiency analysis ($\varphi(\log p) \neq \varphi(p)$) .

5) Case E --- 3D Fourier Transform (3D-FT):

3D-FT is one of the eight benchmark suites in the NAS Parallel Benchmark which uses a divide-and-conquer strategy to evaluate a 3D partial differential equation. As a communication intensive application, 3D-FT is composed of several computation and communication phases, and stresses CPU, memory and network during execution. Among all the communication calls, all-to-all communication dominates the overhead. We use PowerScale to measure both machine and application dependent parameters, shown in Table 4.4.

Table 4.4: Parameters' estimation for 3D FFT

Machine	Estimation
t_c	$\frac{6.41}{f} * 10^{-10}$
t_m	$1.12 * 10^{-7}$
t_{msg}	$2.53 * 10^{-5}$
t_{Byte}	$1.82 * 10^{-8}$
$P_{(total-idle)}$	$27.68 * f^2$
ΔP_c	$3.4 * f^2$
ΔP_m	$0.76 * f^2$

Application	Estimation
α	0.86
W_c	$1.06 * 10^4 N$
W_m	$9.49N$
W_{co}	$4.46 * 10^3 * Nlog_2^p$
W_{mo}	$-0.73 * Nlog_2^p$
M	22
B	$\frac{4N}{4^{(log_2^p - 1)}}$

According to Table 4.4, we can solve the correlation $N_{3D-FFT} = F(p, \vartheta)$ while p and N scale simultaneously. For example, we find that $\vartheta = 0.9$ when $p = 8$ and $N = 0.8 * 10^6$. Looking at the graph, 90% energy efficiency is a reasonable target θ for this case. We need to find N when p scales to 16 and above. We can estimate the workload as:

$$\theta = 0.9 = EE_{3D-FT}(0.42 * 10^{6,8}) = EE_{3D-FT}(6.185 * 10^{6,16})$$

In the example above, users can easily find that there is no valid *N* value that will keep 90% system-wide energy efficiency when *p* scales to 32 and above. However, users can still choose the *N* that is close to 90% as an approximate to run the application. By

using equations (4.22) and (4.24), we can estimate correlation between N and p under fixed f for 3-D FT on *SystemG*: $N_{3D-FT} = f'(\vartheta)\varphi\left(\frac{p^2}{\log_2 p}\right)$. We can infer that N needs to grow with p at an overall rate of $\varphi\left(\frac{p^2}{\log_2 p}\right)$ in order to maintain system-wide desired energy efficiency ($N > 0$). 3D-FT has a larger communication to computation ratio compared to ROMM, thus energy efficiency decreases faster while p scales up due to faster growth of E_p caused by communication overhead. Overall energy efficiency improvement reacts to increasing problem size faster for 3D-FT than ROMM, especially for larger system size (for instance, when p = 128). The problem size scaling approach can be effectively used for 3-D FT to reduce the negative impact on energy efficiency caused by increasing p. The salability rendering of 3D-FT is shown in Figure 4.11.

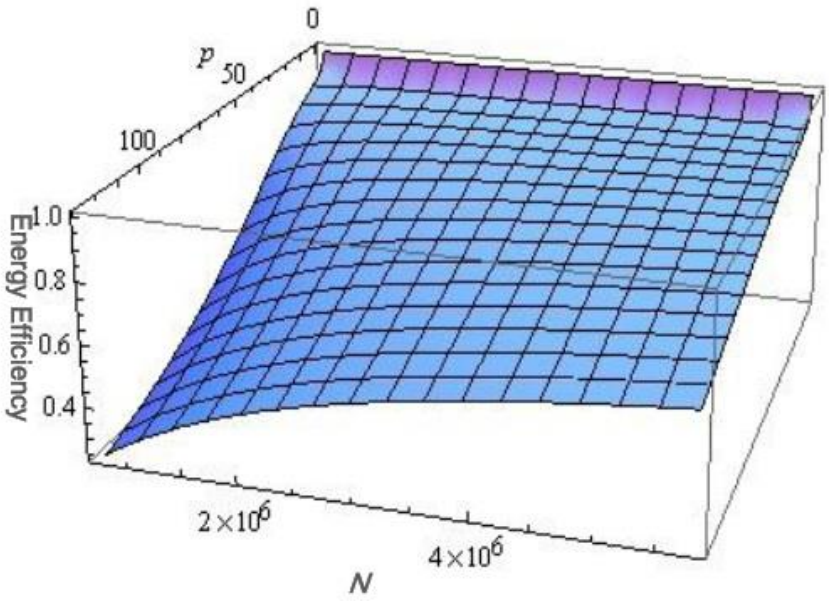


Figure 4.11: Illustration of 3D-FT's energy efficiency scaling with frequency fixed to 2.8 GHz.

6) Discussion and Categorization:

Figure 4.12 shows classification of several applications with individual *N*'s growth rate. Based on the case studies, we can categorize parallel applications by how they are affected by the problem size scaling approach:

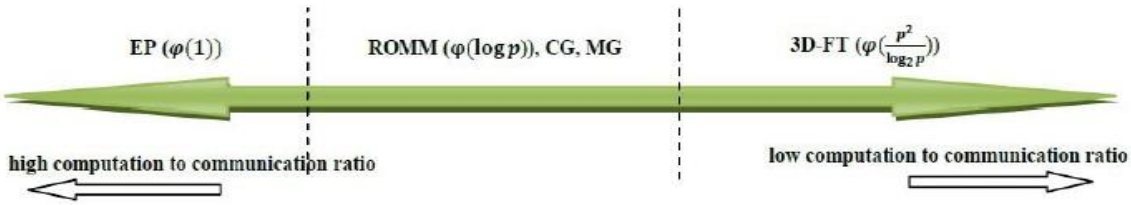


Figure 4.12: Categories of applications based on computation to communication ratio.

1) Purely computation intensive with minimal communication (ideal case). These applications do not react to increasing problem size to maintain energy efficiency while scaling up *p*. Neither *p* nor N scaling will significantly improve or degrade the overall energy efficiency. For example Embarrassingly Parallel.
2) Both computation and non-negligible communication. Applications such as ROMM, Conjugate Gradient (CG) and Multigrid (MG) from NPB belong in this category. This type of application cannot maintain system energy efficiency as well as the ideal case. The problem size scaling approach is able to help these applications improve their overall energy efficiency.
3) Communication intensive. Both performance and energy efficiency do not scale well. 3D-FT belongs in this category. The problem size scaling approach is able to help these applications improve their overall energy efficiency.
4) Both computation and memory intensive with non-negligible communication. Their system-wide energy efficiency is not sensitive to changing workload. Unlike the ideal

case, increasing system size will cause energy efficiency degradation. However, scaling workload will not help improve energy efficiency while *p* scales. Examples such as HPL and LU from NPB.

B. Problem Size Scaling:

Frequency scaling can be another effective approach to maintain or improve overall system energy efficiency while system size scales. For a DVFS-based power aware cluster, we assume each of its compute nodes has *s* power/performance modes or processor frequencies available $\{f_1, f_2, f_3, \cdots, f_s\}$ satisfying $f_{min} = f_1 < f_2 < \cdots < f_s = f_{max}$. For the simplicity of discussion, problem size *N* is fixed in this section. We extend the general model described in equation (4.1):

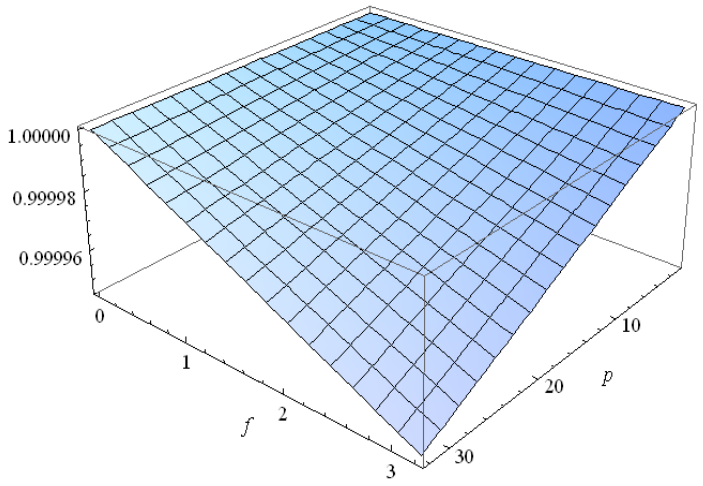


Figure 4.13: 3D plot of EP with p and f as variables under fixed workload.

$$EE_{(f,p)} = \theta = \frac{E_{1,f_{min}}}{E_{p,f_\delta}} \tag{4.25}$$

where $1 \leq \delta \leq s$. We denote total energy consumption for application with fixed workload running on one processor with the minium frequency as $\mathbf{E_{1,f_{min}}}$ and on p processors with frequency $\mathbf{f_\delta}$ as $\mathbf{E_{p,f_\delta}}$.

While problem size *N* can scale continuously, the number of CPU frequencies available are typically very limited and non-contiguous, so it is not practical to use the method in Figure 4.4 to find f in order to reach specific desired θ . We focus on studying how to use frequency scaling to improve energy efficiency instead. In this subsection, we use EP, CG, 3D-FT and HPL as case study candidates.

1) Case A--- EP:

Based on the EP I-E-E model introduced in previous section (problem size scaling: Case A), we can render EP's image with p and f as variables, shown in Figure 4.13:

Figure 4.13 illustrates the variation of EE_{EP} . This figure indicates that energy efficiency hardly changes with *p* and *f*. Energy efficiency is close to 1 for different combinations of *p* and *f* because only minimum communication overhead is imposed.

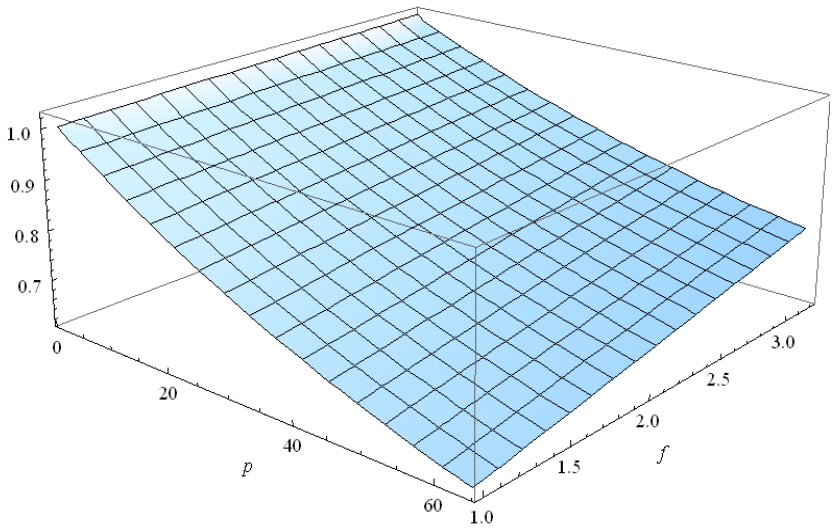


Figure 4.14: 3D plot of CG, assuming problem size n=75000, with p and f as variables.

2) Case B---CG:

Figure 4.14, a 3D energy efficiency plot of CG with *p* and *f* as variables, shows energy efficiency declines with increase in the level of parallelism. In contract to *EP*, the energy efficiency increases with CPU frequency. Digging further to examine the energy overhead E_o and energy consumption of E_1, we observe both increase when frequency increases. However, the EEF_{CG} decreases while frequency increases because E_1 increases faster than E_0. In this strong scaling case, users can scale the frequency up using DVFS to achieve better energy efficiency. Also, compared to *FT* (see Figure 4.15), the effects of frequency have more impact on the on-chip workload of *CG* than *FT* as *p* scales due to a lower communication to computation ratio.

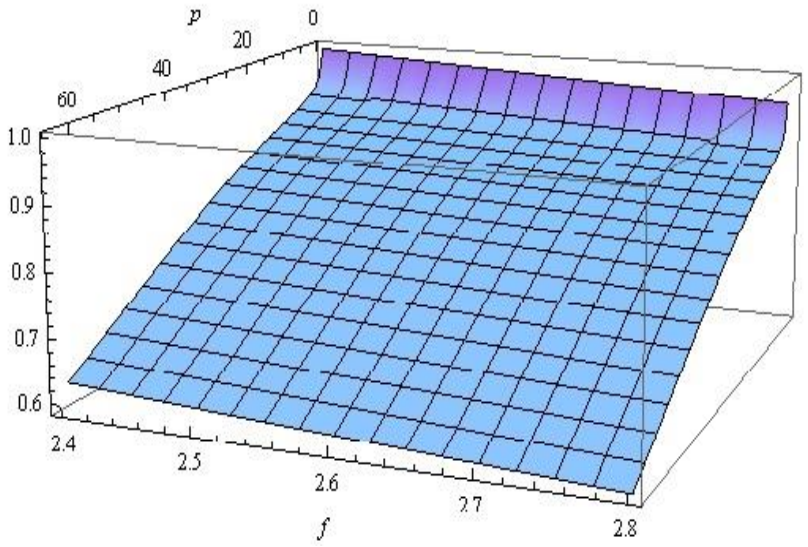


Figure 4.15: Illustration of 3-D Fourier Transfer under fixed problem size $N = 1.15 * 10^6$ on *SystemG* while *p* scales up.

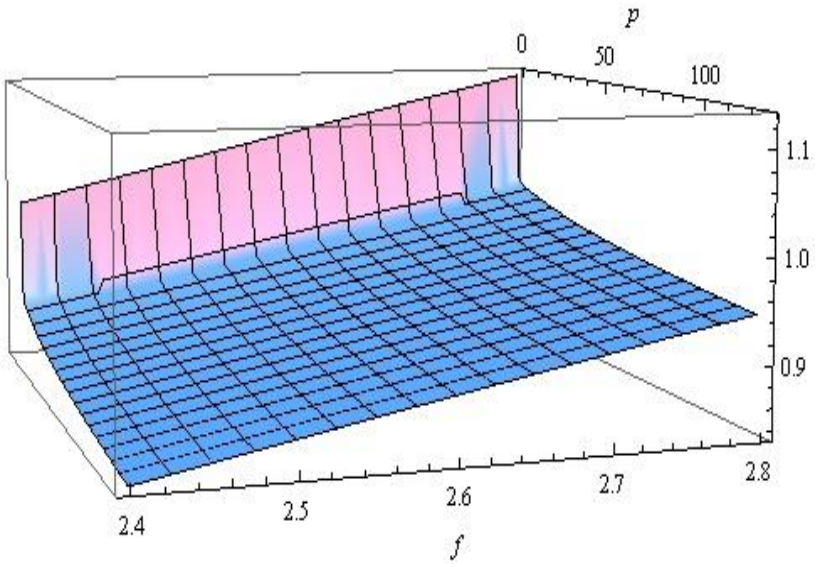


Figure 4.16: Illustration of HPL under fixed problem size $N = 10000$ on *SystemG* while *p* scales up.

3) Case C---3D FT and HPL:

Reusing the parameters from Table 4.4, we model 3D-FT under fixed problem

size $N = 1.15 * 10^6$, shown in Figure 4.15. Figure 4.15 shows that, for the fixed problem size, scaling down the frequency (from 2.8GHz to 2.4 GHz) will save total energy consumption (normalized to $E_{1,f_{min}}$) and improve overall system energy efficiency. However, scaling frequency does not improve energy efficiency significantly. This is because 3D-FT is a communication intensive application and the effect of frequency change on on-chip workload is diminishing while p scales up. Also, there are only two available frequency levels supported on our test cluster SystemG. This limits the possibility of higher energy efficiency improvement by further scaling down frequency. In order to investigate whether continuing to lower the CPU frequency can further reduce the total energy consumption we also ran 3D-FT on another power-aware cluster Dori that has more frequency levels (see Table 4.1).

Figure 4.17 shows that the pattern of energy reduction continued on our second cluster as frequency was lowered until 1.2 Ghz. Between 1.2 and 1Ghz, energy consumption increases because the performance penalty of lowering frequency offsets the benefit of lower CPU power. For the HPC community, the balancing point in this context is the system configuration that maximizes energy efficiency. Using coefficient function (4.23), users can learn more about how to scale frequency to achieve better energy efficiency.

For the applications that are not dominated by communication such as HPL, scaling to higher frequency may improve the overall energy efficiency. Figure 4.16 shows that HPL exhibits different behavior than the 3D-FT case. Using a higher frequency for processors improves the total energy efficiency. The graph area that is above 1.0 energy efficiency is because we use $E_{1,f_{min}}$ to normalize all the energy

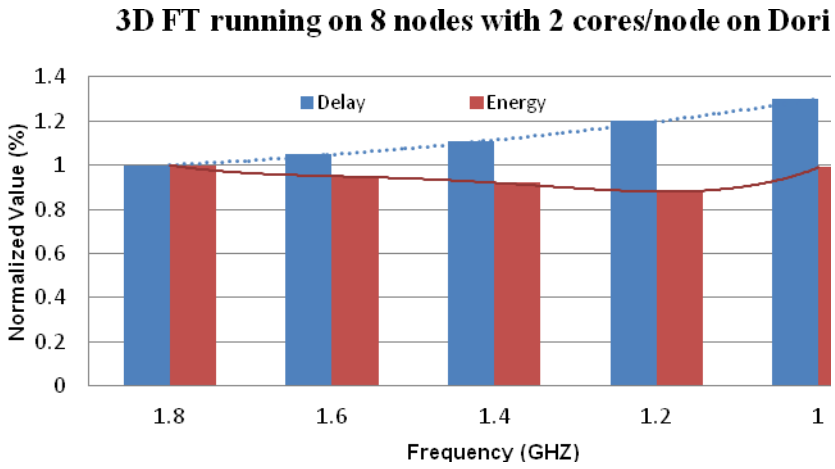


Figure 4.17: Energy and Delay of FT running on *Dori* cluster with five available frequencies. Results are normalized with 1.8 GHz situation.

efficiency results. For a computation and memory intensive benchmark such as HPL, higher frequency will help achieve better performance and less latency. Also, unlike 3D-FT, the frequency impact on on-chip workload will not dramatically diminish when system scales up due to its higher computation to communication ratio. Since total system energy consumption E is a sum of products of average power and delay over all the computing nodes $E = \sum_{1}^{\#nodes} \overline{Power} * Delay$, performance benefits overcome the increased average power consumption caused by higher frequency and result in a reduced total energy consumption and improved system-wide energy efficiency.

In addition to applying an appropriate frequency to all the operating processors to save energy for a communication intensive case shown in Figure 4.15, further energy reduction is possible at the application level by applying DVFS to "communication slack". Application level optimizations are beyond the scope of this work.

C. Discussion:

Discussion of $P_{total-idle}$, ΔP_c, ΔP_m

We classify $P_{total-idle}$, ΔP_c and ΔP_m into machine-dependent variables because their behaviors are highly related to Chip's V_{add} [67] and frequency, *f*. However, they are not only affected by machine architecture but also affected by traits of application. The execution pattern of an application could also affect the power consumption during execution. For simplicity, we assume they are only affected by hardware. From Kim, et al [68], we assume power is proportional to f^{γ} ($\gamma \geq 1$). Different hardware architecture could result in different γ value.

Discussion of the effect of the level of parallelism, *p*

We can rewrite Equation (4.15) as follows to see the relation between E_0 and *p* when the workload is evenly divided among processers (homogeneous workload):

$$E_O = E_p - E_1$$

$$= \alpha T_o(P_{total-idle}) + pw_{co}t_c\Delta P_c + pw_{mo}t_m\Delta P_m$$

$$= p[\alpha(w_1co\ t_1c + w_1mo\ t_1m + T_1net\ ) + w_1co\ t_1c\ \Delta P_1c + w_1mo\ t_1m\ \Delta P_1m,$$

where $T_o = (pw_{co}t_c + pw_{mo}t_m + pT_{net})$

Thus, E_0 is $O(p^k)$ ($k \geq 1$). Generally speaking, more parallelization will incur lower energy efficiency. In this case, the application's tasks among all nodes require extra computation, memory accesses and communication efforts to coordinate with each other to complete the job. We observe this phenomenon in *FT* and *CG*. In contrast, *EP* incurs

almost no overhead and energy efficiency doesn't decrease significantly with the increase of the levels of parallelization.

Discussion of problem size *n*.

Problem size is a dominant factor affecting energy efficiency. The *EE* for applications *FT* and *CG* improve if the problem size scales. However, increasing problem size does not necessarily improve energy efficiency as in the case of energy efficiency for *EP*.

Discussion of frequency, *f*

Decreasing frequency can either increase or decrease energy efficiency. For *EP* and *FT*, we observed no energy efficiency improvements for parallel execution when we adjust to low frequency. However, in the case of *CG*, we found that higher frequencies can improve energy efficiency because the memory overhead W_{mo} value decreases while system scales up.

4.8 Chapter Summary

Table 4.5 presents the Pros and Cons for the problem size and frequency scaling approaches to maintain or improve energy efficiency:

Table 4.5: Pros and Cons for workload and frequency scaling approaches for improving overall system-wide energy efficiency.

Scaling	Pros	Cons
f	1) From the view point of saving total energy in order to improve energy efficiency. 2) Easy to apply.	1) Limited levels of frequencies available to scale. This also limits the rate of system energy efficiency improvement. 2) Encounter a certain degree of performance penalty.
N	1) From the view point of more efficiently utilize energy to complete more workload. 2) Has a bigger range to scale and very flexible.	1) Does not fit for the applications that have very limited input data range or underlying systems that have very small memory.

Chapter 5

A Simplified and Accurate Model of Power-Performance Efficiency on Emergent GPU Architectures

Emergent heterogeneous systems must be optimized for both power and performance at exascale. Massive parallelism combined with complex memory hierarchies form a barrier to efficient application and architecture design. These challenges are exacerbated with GPUs as parallelism increases orders of magnitude and power consumption can easily double. Models have been proposed to isolate power and performance bottlenecks and identify their root causes. However, no current models combine simplicity, accuracy, and support for emergent GPU architectures (e.g. NVIDIA Fermi).

In this chapter, we combine hardware performance counter data with machine

learning and advanced analytics to model power-performance efficiency for modern GPU-based systems [57, 58]. Our performance counter based approach is simpler than previous approaches and does not require detailed understanding of the underlying architecture. The resulting model is accurate for predicting power (within 2.1%) and performance (within 6.7%) for application kernels on modern GPUs. Our model can identify power-performance bottlenecks and their root causes for various complex computation and memory access patterns (e.g. global, shared, texture).

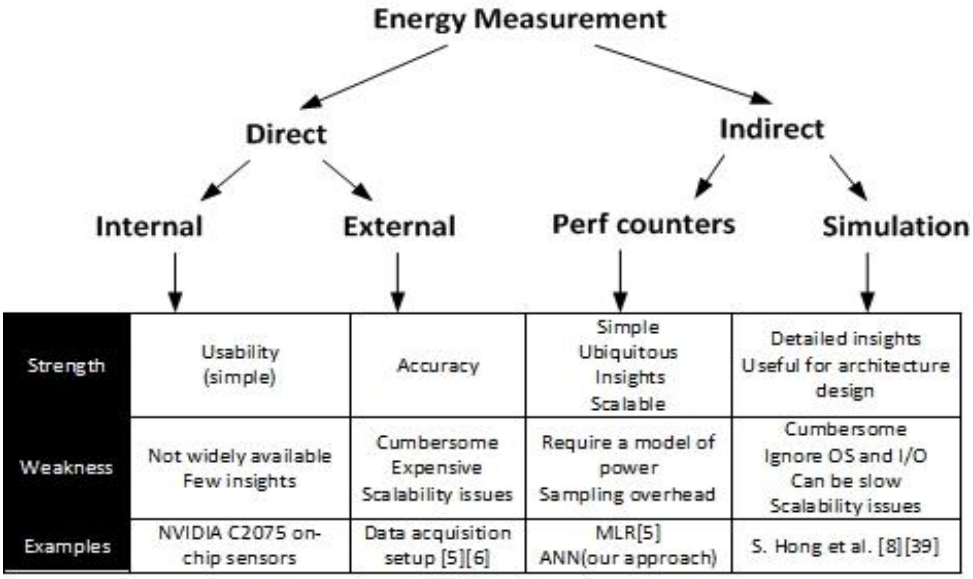
We measure the accuracy of our power and performance models on a NVIDIA Fermi C2075 GPU for more than a dozen CUDA applications. We show our power model is more accurate and robust than the best available GPU power models — multiple linear regression models MLR and MLR+. We demonstrate how to use our models to identify power-performance bottlenecks and suggest optimization strategies for high-performance codes such as GEM, a biomolecular electrostatic analysis application. We verify our power-performance model is accurate on clusters of NVIDIA Fermi M2090s and useful for suggesting optimal runtime configurations on the Keeneland supercomputer at Georgia Tech.

We will also show how we can integrate the GPU models described in this chapter into our iso-energy-efficiency model illustrated in Chapter 4 to cover a more broad range of HPC systems.

5.1 Introduction

According to the US DOE [3], we will need a 1000-fold performance increase with only a 10-fold power increase to achieve the goal of a sustained petaflop within 20

Megawatts by 2019. Thanks in part to efficiency gains from GPU-based supercomputers [8, 9], the most efficient systems now surpass 2,000 MFLOPS/watt. However, meeting the DOE's goals will require 50,000 MFLOPS/watt and significant innovation in power and performance efficiency.



	Internal	External	Perf counters	Simulation
Strength	Usability (simple)	Accuracy	Simple Ubiquitous Insights Scalable	Detailed insights Useful for architecture design
Weakness	Not widely available Few insights	Cumbersome Expensive Scalability issues	Require a model of power Sampling overhead	Cumbersome Ignore OS and I/O Can be slow Scalability issues
Examples	NVIDIA C2075 on-chip sensors	Data acquisition setup [5][6]	MLR[5] ANN(our approach)	S. Hong et al. [8][39]

Figure 5.1: Past and current approaches to energy measurement in GPU based HPC systems.

Despite their growing importance to supercomputing, it is exceedingly difficult to quantify and optimize GPUs for both power and performance on real applications and systems. Figure 5.1 shows there have been some work to understand GPU energy use through measurement. While internal on-board power sensors [196] are emerging and showing promise, they lack ubiquity or standard interfaces making them less practical for wide-spread use today. External meters [49, 91, 144] can provide accurate measurements at the expense of usability since instrumentation may be difficult and require physical system access and invasive probing.

Simulations can provide excellent detail provided the user is an expert with in-depth understanding of the GPU architecture. The best available GPU power model to date [69] uses emulated performance data combined with detailed architectural information to estimate integrated GPU power-performance efficiency. However, such

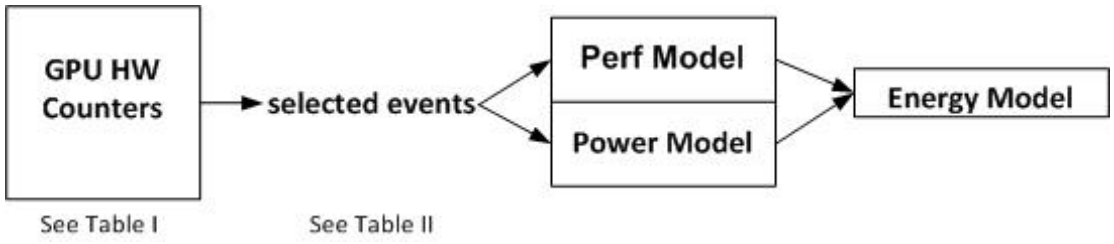


Figure 5.2: A GPU performance counter based approach to estimate energy use on a real system.

simulators tend to ignore the effects of the operating system and the I/O subsystem. Additionally, the more than 70 parameters used by the model while of great use to an architect make the model unwieldy for use in application and system optimization at runtime. Lastly, long simulation times are not conducive to trial and error power-performance optimization of full applications and systems.

Without the aid of internal or external sensors and due to the time consuming nature of simulations, there are few alternatives to obtaining energy usage information for running applications and systems. For CPU-based systems, power models based upon hardware performance counters have shown promise for live runtime analysis [162] as a simpler alternative to complicated simulation. Such models are useful for real system analysis since (thanks to the growing use of hardware event counters) they are portable to a large number of systems and have been shown to be reasonably accurate.

The best available GPU power models using hardware performance counters rely on statistics to correlate power to performance. Such statistical models [49] use multiple linear regression (MLR) and hardware counter data to estimate GPU power consumption. These models provide a good balance of abstraction and application/system detail but potentially ignore the nonlinear relationships between power and performance. This could lead to inaccuracy for individual kernel power estimation. Furthermore, existing models require significant adaptation to support modern complex GPU architectures with true cache hierarchies [197].

We believe GPU power models must be simpler, more accurate, and applicable to emergent systems (e.g. NVidia Fermi [197]). Furthermore, such models should lend themselves to use at runtime and provide enough insight to isolate both power and performance bottlenecks despite their simplicity. We propose an approach (see Figure 5.2) that relies on GPU performance counter data to estimate energy use on a real system without the need of external power metering hardware or simulation.

This chapter contributes three novel, accurate, GPU models in succession: 1) a power model that estimates average power use and produces a GPU power profile for an application; 2) a performance model that reveals the correlation between execution time and key GPU performance parameters; and 3) an energy model that estimates energy consumption using performance counter events and the aforementioned power and performance models.

We validate our models on two NVIDIA Fermi-based systems including the Keeneland supercomputer [198] at Georgia Tech and compare against the best available performance counter based power models: MLR and MLR+ [49, 199]. We demonstrate

how to use the GPU models to: 1) identify power bottlenecks; 2) identify potential performance bottlenecks; 3) predict optimal power-performance efficiency; and 4) study energy-performance tradeoffs for GPU based clusters.

Our experiments show that MLR and MLR+ mispredict power by as much as 23% for some CUDA kernels while our power model is always within 5% of the actual power.

Our performance model predicts execution time within 6.7% on average for the representative CUDA kernels (CUDA SDK [199, 200] and GEM [13]). Our energy model predicts within 8.9% average error rate for the SHOC parallel benchmarks [201] on Keeneland supercomputer.

5.2 Related Work

5.2.1 GPU Power Modeling and Analysis

Huang et al.[202] analyzed the performance and energy-delay product for the GEM package running on GTX 280. However, they only analyzed one application without providing any general modeling approach. Collange et al. [203] conducted experiments to empirically study how computation and memory access impact GPU power consumption without actually offering a general model for addressing various execution patterns. PowerRed [204] and Qsilver[205] have been proposed to estimate power consumption using architectural simulation. However, they are designed for applications with graphics APIs which are not suitable for modeling general purpose computing kernels running on modern GPU architectures such as Fermi. Ma et al. [144] proposed a statistical power modeling approach using conventional regression methods to

primarily model graphics applications. However, their approach fails to address different memory access patterns on modern architectures such as global and shared memory access. Nagasaka et al. [49] proposed a statistical power modeling approaching using conventional MLR. Unlike our BP-ANN model, they predefined a linear relationship between power and performance counters, which results in higher prediction errors for individual kernels and lower average prediction accuracy. Also, their model cannot predict kernels with texture access and intensive shared bank conflicts accurately. Based on another model [162], Hong et al. [69] proposed an empirical method to model GPU power consumption based on architectural details statically. However, this approach relies on the GPU PTX emulator called Ocelot [206] and other low level architectural information. Since CUPTI has been supported across generations of NVIDIA GPUs, our performance counter directed power models are more portable and simpler to use for real application and system optimization.

B. GPU Performance Modeling

Search-based performance auto tuning approaches have been proposed [50, 207-209] to locate the optimal program configuration in order to achieve best GPU performance. However, these approaches give users and designers little information on performance bottlenecks and their root causes. Zhang et al.[50] proposed an analysis model based on the Barra [210] simulator to identify performance bottlenecks for the GeForce 200-series GPU. This approach relies heavily on the simulation results and models a flat GPU memory system from previous generations. Also, the Barra simulator is designed specifically for G80-based GPUs which makes the model less portable. Based

on detailed architecture level information, Hong et al. [92, 211] proposed an analytical model to estimate execution time by considering global memory parallelism for previous generations of GPU devices. However, their approach does not consider shared memory bank conflicts, texture cache access, and costs for different computation instructions. Also, it relies on PTX code and detailed architecture information and can be difficult to adapt. A compiler-based performance model [169] using an abstract interpretation of GPU kernels (Workflow Graph) has been proposed to predict execution time for CUDA kernels for previous generations of GPUs. Compared to this high level approach developed primarily for compilers, our approach is an easy to use, system level estimation method considering major performance related aspects of Fermi and identifying the performance bottlenecks for optimizations. Their work also models a flat GPU memory system.

5.3 Background

A. GPU Architectures and the CUDA Programming Model

In this thesis, we focus on the NVIDIA Fermi architecture [197], which our power-performance models are based on. Fermi is now a main stream HPC accelerator; 51 out of 53 Top 500 supercomputers [8] equipped with GPUs are Fermi based. We believe our general methodology can be applied to model any GPU architecture with minimal adaptations. The NVIDIA Fermi architecture contains up to 16 streaming multiprocessors (SM). Each SM is integrated with 32 streaming processors (SP). Each SP has a fully pipelined integer arithmetic logic unit (ALU) and floating point unit (FPU) and can execute a floating point or integer instruction per clock per thread [197]. Each

SM has 16 load/store units and 4 special function units (SFUs). A warp, which contains 32 parallel threads, executes in a single instruction multiple data (SIMD) fashion. Each SM has a dual warp scheduler that concurrently schedules and dispatches instructions from two independent warps [197]. Fermi has a 64k configurable L1 cache on-chip (contains shared memory) and an unified L2 cache off-chip. Shared memory needs to be

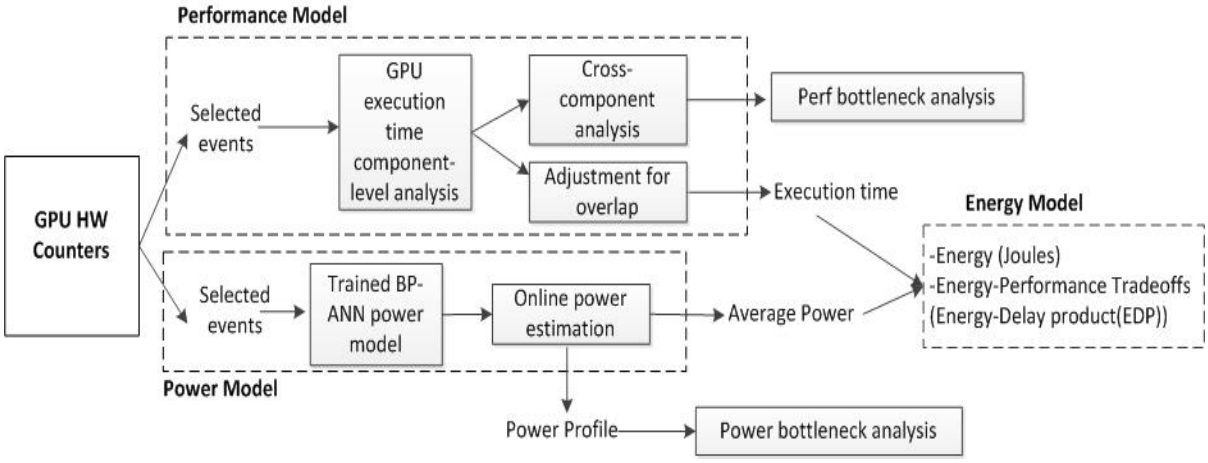


Figure 5.3: Detailed view of our performance counter based approach to estimate energy use on a real system.

explicitly managed but L1 and L2 caches are automatically managed. Global memory, an off-chip GDDR5 DRAM, is divided into 6 partitions in Fermi. A PCI-Express bus is used to connect between host system and GPU(s) for data transferring.

The CUDA parallel programming model [212] uses the single-program multiple-data (SPMD) paradigm. In CUDA, a grid is an array of thread blocks that execute the same kernel and the thread blocks in it are mapped across SMs. A thread block contains several warps and coordinates threads through inter-block synchronization and shared memory. To some extent, the size of the thread block may impact the kernel performance due to occupancy [213] or hardware resource limitations.

B. Bottom-up Modeling Process

Figure 5.3 shows a detailed overview of our hardware counter driven power-performance modeling process. For the power model, we first train an artificial neural network (ANN) [214] based model offline using the selected events from the GPU kernels representing various execution patterns. Our trained power model can use the selected performance events from any application at runtime to profile power consumption. By analyzing the power profile and the corresponding values of the selected events, users can identify power bottlenecks for an application. Users can retrain the ANN model when porting to a different architecture since performance counters are widely available across generations of GPUs.

For the performance model, we use selected performance events to simply isolate the time spent on each major GPU architecture component. By comparing the time spent on each component, we can identify potential performance bottlenecks. Finally, we combine our GPU power and performance models, along with our previous multi-core system energy model [4] , to estimate total system energy consumption and the efficiency for GPU-based clusters .

5.4 GPU Power Modeling

After attempting to use conventional regression models such as MLR to predict GPU power, we found several limitations: a) These models predefine a relationship (e.g. linear relation) between training inputs and output. However, if the actual relationship is

more complicated than that (i.e. non-linear), large prediction errors may appear. b) These models lack flexibility and adaptivity. c) These models provide added simplicity at the expense of suitability for modeling complex architectures such as GPUs.

To address the limits of MLR (Multiple Linear Regression) based approaches, we use an ANN (Artificial Neural Network) to model GPU power consumption. ANNs can capture nonlinear dependencies even when there is significant variance in the training set; ANNs are highly flexible and adaptive (i.e. we can retrain the model with a new pattern without redeveloping the correlation); and ANNs are context-dependent and suitable for working with complex underlying architectures like GPUs since they are capable of learning various GPU computation and memory access patterns. Section 5.6-B(2) shows an accuracy comparison of ANN and MLR based models.

A. Preparation for Training Inputs

1) **Power Data Collection:** This phase is required by model training and validation only. Once the model is trained using internal or external power sensors (see Figure 5.1) for one specific architecture, it can be used to predict power at runtime and deployed in a cluster environment.

In this work, for the purpose of model training and validation, we use an internal approach where the power data is sampled internally over the PCI bus through the NVIDIA Management Library (NVML) [215] interface supported by our underlying test beds equipped with on-chip sensors. To profile power data at runtime, we use one thread to record the current power by calling nvmlDeviceGetPowerUsage () function at every predefined time interval while running a GPU kernel simultaneously. Users who do not

own GPUs equipped with on-chip sensors can use the external approach [49, 91, 144] to train the model and then apply it to every node in the cluster for profiling runtime power.

2) **Performance Counter Profiling:** We need to identify a select group of performance events that are strongly correlated to the power consumption. We use the correlation analysis method introduced by Curtis-Maury et al [38] to automate the event

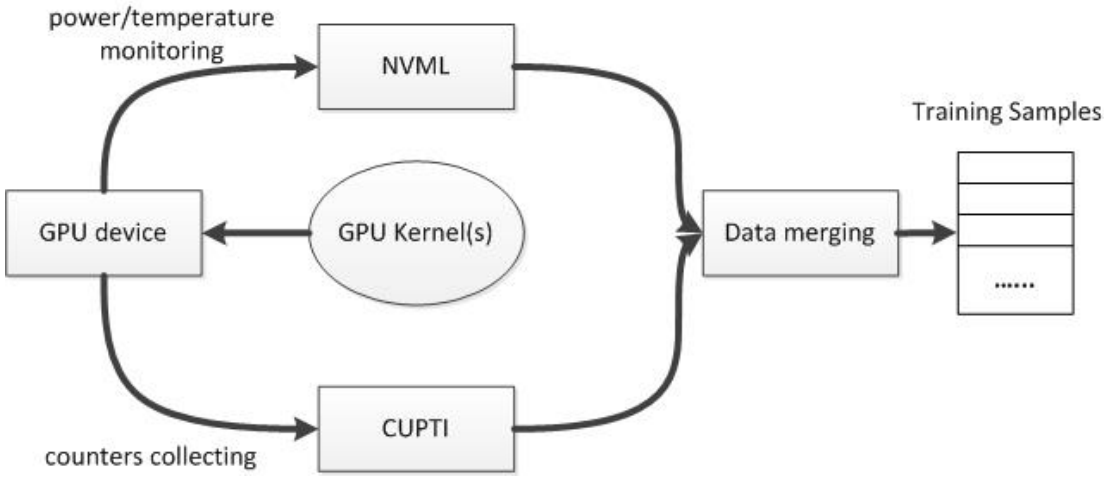


Figure 5.4: The runtime data monitoring and consolidation system for generating training inputs.

selection process. We choose the top 10 performance events (shown in Table 5.2) as the training variables for our ANN.

At runtime, we sample the selected performance events through CUPTI [216]. We use one thread to monitor these events at a predefined time interval (in milliseconds). All event data will be normalized to event rate — the number of occurrences of event i divided by a predefined sampling time interval (cycles). To simplify the data collection and merging process between power and performance counters, we wrote a set of runtime profiling and data consolidation APIs to provide functions such as: performance event

sampling, power sampling, data merging, and event callback. The component diagram of the runtime profiling system is shown in Figure 5.4. Sampling occurs roughly 60 times per sec in the format of a tuple (time, power, $E_{1,s}$, $E_{2,s}$, ..., $E_{i,s}$), where s is the sample id, i is the event id, and $E_{i,s}$ is counter i's event rate at sample s. These sets of tuples will be used as inputs to train ANN power model. For shorter kernels, we run multiple repetitions

Table 5.1: CUPTI Counters Used in This Chapter

counter name	description
gld_request	number of executed global load instructions per warp on a SM
l1_global_load_hit(miss)	cache lines that hit in L1 cache for global load(miss) per SM
global_store_transaction	number of global store transactions per SM
local_load(store)	number of executed local load (store) instructions per warp
inst_executed	number of instructions executed, no replays
branch	number of branches taken by threads executing a kernel
divergent_branch	number of divergent branches within a warp
Shared_load	number of executed shared load instructions per warp
l1_shared_bank_conflict	number of shared bank conflicts per SM
tex0_cache_sector_queries	number of texture cache requests per SM
tex0_cache_sector_misses	number of texture cache misses per SM
uncached_global_load_trans	number of uncached global load transactions per SM

Table 5.2: Performance Event for Training

events	composition	category
G_{load}	gld_request+l1_global_load_hit +l1_global_load_miss	global memory
G_{store}	global_store_transaction	global memory
L_{local}	local_load	local memory
L_{store}	local_store	local memory
exec_inst	inst_executed	instruction pipeline
branch	branch	instruction pipeline
divergent_branch	divergent_branch	instruction pipeline
S_{Access}	Shared_load+l1_shared_bank_conflict	shared memory
T_{hits}	tex0_cache_sector_queries	texture memory
T_{miss}	tex0_cache_sector_misses	texture memory

of the same kernel to gain more training samples for our ANN. Power overhead caused by profiling threads at runtime has been taken into consideration in the final inputs for ANN training (e.g. for both C2075 and M2090 GPUs, the average power overhead is less than 5 watts).

B. Major GPU Power-Related Factors

A complete description of the hardware performance counters used in this thesis is shown in Table 5.1. These are based on NVIDIA devices with Computer Capability 2.x. All the GPU performance counters discussed below are accessed through the CUDA Profiling Tools Interface (CUPTI) [216] which is a runtime performance profiling and tracing interface to provide low level architecture information for CUDA applications. For a detailed description of the selected performance events (each event may contain one or more counters) used in model training, please refer to Table 5.2.

1) ***Global Memory and L1 Access:*** GPU power use is sensitive to global memory accesses (see Figure 5.5). We use three counters to model global memory load access: g gld_request, l1_global_load_hit, and l1_global_load_miss. gld_request represent global memory access intensity per warp. However, the number of load requests may not be equal to the number of actual memory transactions due to the coalescence of global memory accesses. Achieving the number of actual memory transactions is important since a higher number of global memory accesses can cause longer warp waiting time and result in lower arithmetic intensity [217] which ultimately affects instantaneous power. We use the sum of l1_global_load_hit, and l1_global_load_miss to estimate the total number of global memory load transactions per SM. Combined with gld_request,

we then formulate these three factors into a unified training event called G_{load} (shown in Table 5.2). This unified event can effectively describe the global memory access intensity per warp and the actual number of global load transactions. It also reduces the total number of training events for our ANN. For global memory store, we use a single event G_{store} to address its contributing effect to the power consumption.

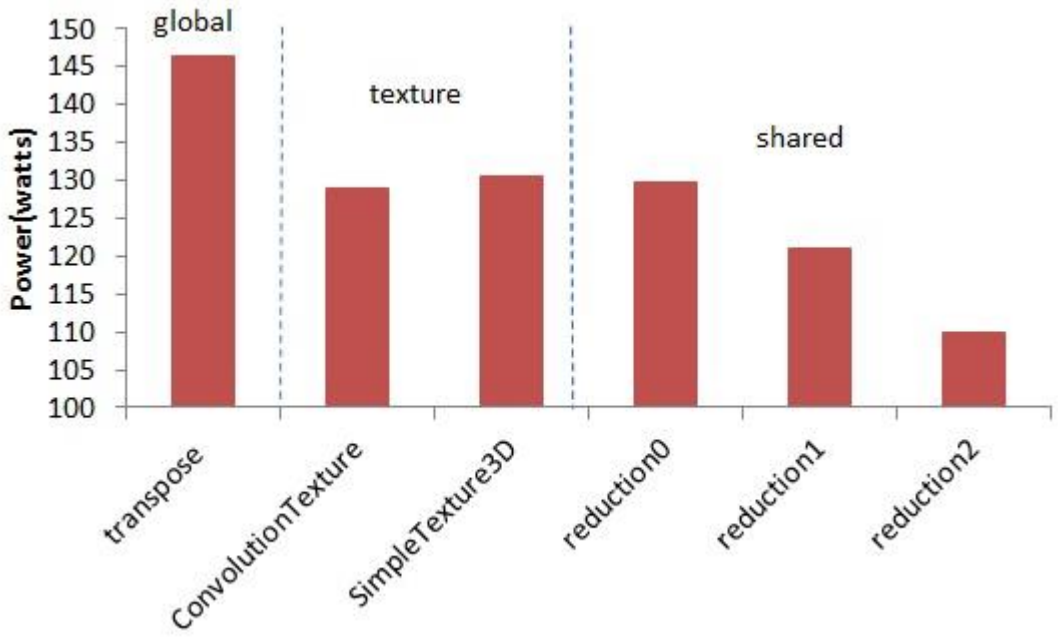


Figure 5.5: Average power consumption of several CUDA kernels that have different memory access patterns on Tesla C2075.

2) ***Texture Cache Access:*** Figure 5.5 shows that two texture memory intensive applications ConvolutionTexture and SimpleTexture3D from CUDA SDK [199] consume less average power (~16 watts) than the transpose kernel, which is global memory intensive. On-chip SRAM access consumes less power than directly loading data from off-chip DRAM. However, texture memory access has not been addressed in previous power models making the scenarios hard to predict accurately. We use T_{hits} and

T_{miss} as training events to model this pattern. A higher number of texture cache misses can cause more global memory loads which may increase the overall power consumption.

3) ***Shared Memory Access:*** An increasing number of shared memory bank conflicts can also affect overall GPU power consumption. More shared memory bank conflicts result in more shared memory transactions being serialized — this results in higher average SM power consumption.

For example, Figure 5.5 shows the average power of three different versions of the reduction kernel from SDK. Reduction2, optimized with no shared bank conflicts, consumes 11 watts less than reduction1 and 14 watts less than reduction0. We define a unified performance event S_{Access} ($Shared\_load + l1\_shared\_bank\_conflict$) to describe shared memory accesses (shown in Table 5.2).

4) ***L2 Cache Access:*** CUPTI provides several hardware counters to address the performance of the L2 cache. After using correlation analysis [38], we found that these counters are not strongly correlated with GPU power like others listed in Table 5.2 and including them as training events will not improve overall model accuracy. Unlike texture cache, global and shared memory access, threads do not interact with the L2 cache directly — the hardware services all the operations (load, store and texture) to L2 and regardless of the L2 access pattern, its power generally remains stable. Since L2 does not contribute to the power changes significantly, it has little effect on our power model's accuracy and therefore we do not include L2 related events in our power model.

C. ANN Design and Training

To ease portability to different architectures, we use a configurable, back-

propagation, artificial neural network (BPANN) [214] to estimate GPU power consumption. Since GPUs are highly parallelized architectures, performance events are not necessarily mapped to power consumption in an affine manner. Therefore, in our BP-ANN model, we use a nonlinear sigmoid function as our activation function in the neuron [214]. A sigmoid function is a special case of logistic functions and can describe the nonlinear mapping from inputs to our training target. The sigmoid function can be described as:

$$Z_j = \frac{\mathbf{1}}{1 + e^{-net_j}} \tag{5.1}$$

Z_j is the output of the current neuron; net_j is the weighted sum of all the inputs from the previous neural layers; and j is the node index. When we introduce more neurons in our ANN, we can capture more non-linear mappings from inputs to our target in order to describe a complicated relationship between input and output. Increasing the number of neurons may improve the accuracy of the model prediction while potentially increasing the computational complexity. To balance accuracy with complexity, we carefully adjust the number of neurons and internal layers in order to achieve high prediction accuracy within user-tolerable complexity. The design of our BP-ANN is illustrated in Figure 5.6. Our BP-ANN model takes the 10 carefully selected performance training events as inputs to the input layer. Based on our experiments, we use two internal hidden layers, each of which uses 4 neurons to train the inputs. The output layer has only one neuron which is the output of the power prediction. Each layer is strongly connected by specific weight values with its previous layer and the next layer. At the training stage, the error between the power prediction and the real power value is used in each iteration

to update the weights on the connections. For a more detailed description of BP-ANN models, please refer to [214] and [218].

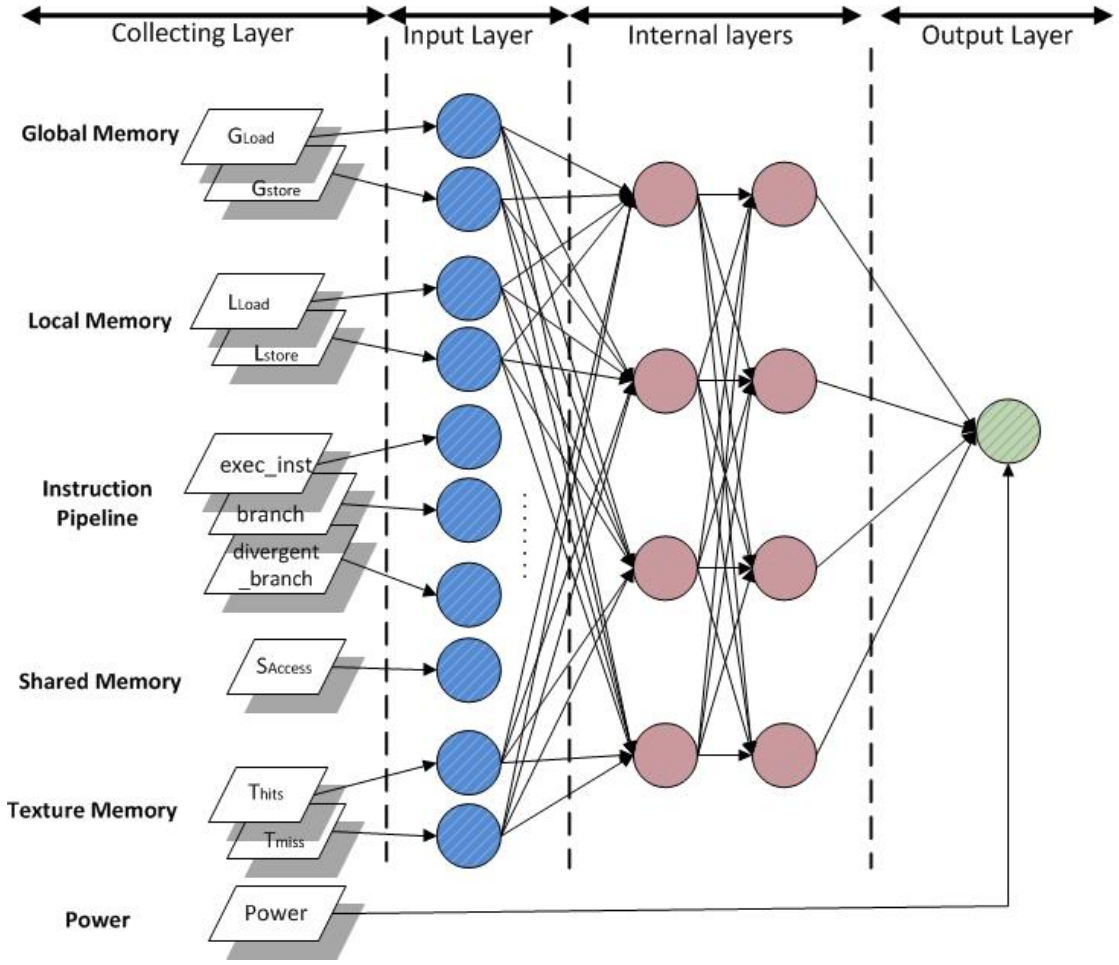


Figure 5.6: Design diagram for the BP-ANN based GPU power model. Circles represent neurons.

D. Limitations of Our Power Model

The performance counters we select for building our power model are measured

in the context of a warp or an SM (except for the number of branches). This makes it difficult for our model to predict two uncommon cases accurately: asymmetric applications and applications not running on all the SMs. This can be improved in future if CUPTI is extended to provide more global counters. For this thesis, we use applications that do not have asymmetric workload and run on all the SMs.

5.5 Modeling Performance for GPU Applications

Our analytical performance model does not rely on simulators or PTX code and it addresses various GPU memory access patterns. It also provides quantitative analysis of performance bottlenecks. The most significant contribution of this high level model is that programmers can estimate the time duration for individual architecture components (e.g. shared, texture and global memory) and then identify which component is the major performance bottleneck using cross-component analysis (see Figure 5.3). Our counter-based performance model is not only faster than the cycle-based simulations but also presents a very decent accuracy. Descriptions for all the selected counters and related parameters are shown in Table 5.1 and 5.3.

A. Global Memory Operations and L1 cache

We use the default caching load (loads one L1 cache line on each transaction) option for CUDA kernels. To estimate the total number of global load transactions per SM (see equation (5.3)); we add: 1) the number of cache lines that hit and miss in L1 for global loads; to 2) the number of uncached global load transactions. This result is then averaged as the number of global load transactions per warp (see equation (5.4)).

Similarly, global store transactions per warp are estimated in equation (3.5). One important reason for estimating global store transactions is that there may exist uncoalesced global stores in some kernels (e.g. a naive transpose kernel in the CUDA SDK).

Once we have the average number of global load and store transactions per warp,

Table 5.3: Parameters Used in Performance Modeling

Parameters	**Definition**	**How to achieve**
$\sharp blocks$	Total number of blocks	program setup
$\sharp warp\_per\_block$	Number of warps per block	program setup
$\sharp active\_sm$	Active number of SMs	program setup
$\sharp warps\_per\_sm$	Number of warps per SM	program setup
$\sharp active\_warps(orAW)$	Number of warps run concurrently on one SM	hardware ceilings
$\sharp active\_blocks(orN)$	Number of blocks run concurrently on one SM	hardware ceilings
$num\_g\_ld\_sm$	Number of global load transactions per SM	hardware counters
$num\_g\_ld\_warp$	Number of global load transactions per warp	hardware counters
$num\_g\_st\_warp$	Number of global store transactions per warp	hardware counters
$\sharp iter$	Number of global load (or store) instructions per thread	kernel code
T_c	A single computation time period per warp	equation(8)
$num\_comp\_inst_i$	Number of type i computation instructions per computation period per thread	kernel code
$type\_cost_i$	latency cost for executing type i instruction	Whitepaper[10]
$num\_register\_access$	Number of register access per computation period per thread	kernel code
T_{tex}	Texture cache access time per warp	equation(9)
T^g_{load}	Cycles of one global memory load period for a warp	equation(6)
in_warp_delay	Delay between two uncoalesced global memory transactions	microbenchmarks
out_warp_delay	Delay between two consecutive warps' memory operations	microbenchmarks
T^g_{store}	Cycles of one global memory store period for a warp	equation(7)
T_{shared}	Cycles for the accumulated shared memory access in one group of active warps	equation(10)
$num\_s\_base\_active$	Base number of shared memory access per active group of warps	kernel code
$num\_s\_conf\_active$	Number of shared bank conflicts per active group of warps	hardware counters
T_{active}	Cycles for executing one active group of warps on a SM	kernel code
$\sharp round$	Number of active group of warps executed on a SM	equation(11)
T_{exec}	Execution time	program
$Sync$	Synchronization overhead	program
Parameters	**Definition**	**Est. on C2075**
CYC^g_{load}	Estimated cycles spent on a round trip global memory load transaction	392 cycles
CYC^g_{store}	Estimated cycles spent on a global store transaction	210 cycles
CYC^s	Estimated cycles per shared memory access	4 cycles
CYC^r	Estimated cycles per register access	1 cycle
CYC^t	Estimated cycles per texture cache access	18 cycles
mem_band	Estimated global memory bandwidth	116.7 GB/s
Comp Instruction type	**Examples**	**Estimated cycles**
$Regular$	FMA	1 cycle[10]
$Transcendental$	sin,cosine,reciprocal,square root	8 cycles[10]
$Double\ Precision$	double FMA	2 cycles[10]

we can use equations (5.6) and (5.7) to estimate the execution time for a single global load and store per warp. Refer to Table 5.3 for equation parameters.

$$\#warps\_per\_sm = (\#blocks)/(\#active\_sm) \cdot \#warp\_per\_block \quad (5.2)$$

$$num\_g\_ld\_sm = l_1\_global\_load\_hit + l_1\_global\_load\_miss + uncached\_global\_load\_transaction \quad (5.3)$$

$$num\_g\_ld\_warp = \frac{num\_g\_ld\_sm}{\#warps\_per\_sm}, num\_g\_ld\_warp \geq 1 \quad (5.4)$$

$$\mathbf{num\_g\_st\_warp = \frac{global\_store\_transaction}{\#warps\_per\_sm}, num\_g\_st\_warp \geq 1} \quad (5.5)$$

$$\mathbf{T^g_{load} = CYC^g_{load} + (num\_g\_ld\_warp - 1) \cdot in\_warp\_delay} \quad (5.6)$$

$$\mathbf{T^g_{store} = CYC^g_{store} + (num\_g\_st\_warp - 1) \cdot in\_warp\_delay} \quad (5.7)$$

B. Shared Memory and Bank Conflicts

The effects of on-chip shared memory accesses and bank conflicts on execution time have been either ignored or not fully addressed in previous performance models. Even though the access latency to shared memory is typically short, intensive shared

memory accesses with large numbers of bank conflicts can dominate execution time. There are two features of shared memory access that can help us model the execution pipeline: 1) shared memory is divided into num_{sbanks} of adjacent banks (num_{sbanks} =32 for Fermi). num_{sbanks} represents the number of threads that can concurrently access the shared memory; 2) if n threads in a warp access different 4-byte words in the same bank, these n transactions will be serialized. We will demonstrate how these two features impact the performance in the case study section of this chapter.

Shared memory access latency is comprised of two components: base number of loads and stores and additional serialized accesses caused by L1 shared bank conflicts. The former can be estimated through program analysis. The latter can be estimated by the l1_shared_ bank _conflict counter.

C. Computation

The time for executing a type of computation instruction (non-memory instruction) by a warp depends on the number of functional units available for this type of computation. For instance, in Fermi, there is a fully pipelined ALU and an FPU on each SP [197] and 32 SPs on each SM. One clock cycle is needed to dispatch the same fused multiple-add (FMA) instruction for all threads in a warp. Times to execute other types of instructions are shown in Table 5.3. We treat branch instructions as computation instructions by counting their number on each thread's critical path and adding them into the computation period T_c. For instance, in a for loop, we add the comparative and incremental instructions of one iteration to Tc. In our model, estimation for a single computation period per warp (Tc, computation period, denoted as "C" in Figure 5.8) is

calculated through equation (5.8) below. V is the number of computation instruction types.

$$T_c = \sum_{i=1}^{V} (num\_comp\_inst_i \cdot type\_cost_i) + num\_register\_access \cdot CYC^r \quad (5.8)$$

D. Texture Cache

In the Fermi architecture, if all the threads within a warp access different addresses in the texture cache, these accesses will be serialized since the texture cache only provides one value at a time. However, when all the threads in a warp access the same address, there will only be one transaction due to a multicast. If a miss happens, the penalty of a global load will be applied. Based on these rules, we can estimate the texture cache access time per warp using texture counters:

$$T_{tex} = \frac{tex0\_cache\_sector\_queries - tex0\_cache\_sector\_misses}{\#warps\_per\_sm} \cdot CYC^t + \frac{tex0\_cache\_sector\_misses}{\#warps\_per\_sm} \cdot CYC^g_{load} \quad (5.9)$$

E. Active Warps

GPUs hide memory and instruction pipeline latency by launching enough threads to maximize the overlap of parallel tasks (i.e. warps). However, there are several architectural upper bounds that limit the total number of active warps running on an SM concurrently. They are documented in the CUDA Occupancy Calculator [213]. For instance, the maximum concurrently running blocks per SM is 8 and the maximum active warps per SM is 48. In the Fermi architecture, the partition camping problem [219] has

been significantly alleviated thanks to hashing and new memory controller design. For simplicity, we assume each SM uses $\frac{1}{\#active\_sm}$ of the total global memory bandwidth. So in order to fully utilize the global memory bandwidth we need to maximize the number of warps loading data from global memory concurrently — which can be estimated as $\frac{CYC^{g}_{load} \cdot \frac{mem\_band}{\#acive\_sm}}{128}$ (we assume 128 bytes per warp load here). Combining all of the above factors, we can estimate the maximum number of active warps that can run concurrently in an SM and denote this as $\#active\_warps$.

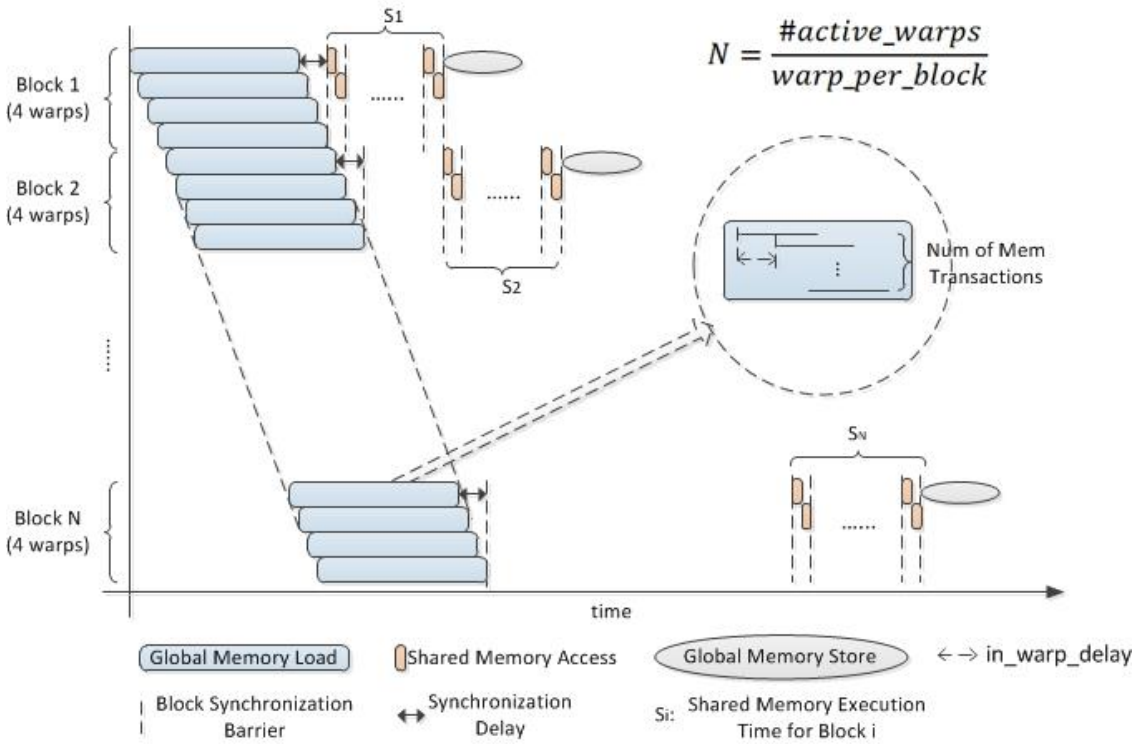


Figure 5.7: Execution time pipeline of reduction 1 kernel (from the CUDA SDK) with one group of active warps. This analysis is performed similarly to each kernel to associate performance events to code characteristics.

F. Case Study: Parallel Reduction

In this section, we will use the parallel reduction kernel (reduction1 kernel in SDK) with interleaved addressing and bank conflicts to show how to estimate the execution time for shared memory intensive applications. This case study can also serve as a general approach to model kernel execution time.

Figure 5.7 shows the general execution pipeline for the active warps that can be executed concurrently in one SM in reduction1 kernel. In this study, we use 128 threads (4 warps) per block as an example. Also, the shared memory access section Si contains several iterations and the iteration number depends on the number of threads per block. The following are the steps for estimating the execution time for the reduction1 kernel shown in Figure 5.7:

a) Decide the maximum number of warps or blocks that can be launched concurrently based on the previous subsection (E). In this case, since reduction1 is not global memory intensive and the program configurations are under the hardware ceiling, there are 8 blocks (32 warps) available to run (N=8) concurrently on an SM.

b) Estimate the actual number of memory transactions for global loads and stores in order to address uncoalesced memory accesses. We can use equations (5.3)(5.4) and (5.5) to estimate the memory transactions for global loads/stores per warp.

c) Estimate shared memory access latency based on the previous subsection (B). The S_i (i is an integer and $i \in [1, N]$) in Figure 5.7 represents shared memory access period per block. We cannot illustrate the memory latency caused by shared memory bank conflicts in the figure because the length of the orange rectangles may vary depending upon the amount of memory conflicts. For instance, the number of shared bank conflicts in this example is much higher than the number of base shared memory

and global memory accesses.

d) Combining all the steps above with equations (5.6) and (5.7), we can estimate the total cycles for executing one active group of warps on an SM as:

$$T_{active} = T^{g}_{load} + T_{shared} + T^{g}_{store}$$

where,

$$T_{shared} = \sum_{i=1}^{AW} S_i = (num\_s\_base\_active + num\_s\_conf\_active) \cdot CYC^{s} \quad (5.10)$$

When we calculate the final kernel execution time T_{exec} (see equation (5.13)), the number of $\mathbf{T^{g}_{store}}$ and $\mathbf{T^{g}_{store}}$ varies depending on the application. For this case, only one of each is counted in T_{exec} because of the execution pipeline depicted in Figure 5.7.

Since we only model the cases of symmetric kernels in this thesis, the execution time for the reduction1 kernel is:

$$\#round = (\#blocks)/(\#sm \cdot \#active\_blocks) \quad (5.11)$$

$$\Sigma Sync = (\#active\_warps - 1) \cdot out\_warp\_delay \quad (5.12)$$

$$T_{exec} = \#round \cdot T_{active} = \#round \cdot (T_{load}{}^{g} + \Sigma Sync + num\_s\_base\_active \cdot CYC^{s} + T_{store}{}^{g}) + l_1\_shared\_bank\_conflict \cdot CYC^{s} \quad (5.13)$$

Similar methods can be used to model other CUDA kernels. For instance, we can estimate the execution time for the coalesced transpose kernel by summing up the different components shown in Figure 5.8 to estimate T_{exec}. The execution time of the coalesced transpose kernel from Figure 5.8 (assuming 4 warps per block) is:

$$T_{active} = T_c + T^g_{load} + Sync \cdot N + T_c(N-1) + \frac{T_{shared}}{N} + T^g_{store}$$
$$Sync = (\#warp\_per\_block - 1) \cdot T_c$$

The benefit of using our event counter based performance model is that users can isolate the time spent on individual architecture components and then compare them to identify potential bottlenecks without relying on simulation or PTX code. For example, in Figure 5.7, we can isolate the time spent on the shared memory bank conflicts and estimate that it is about 28% of the total execution time.

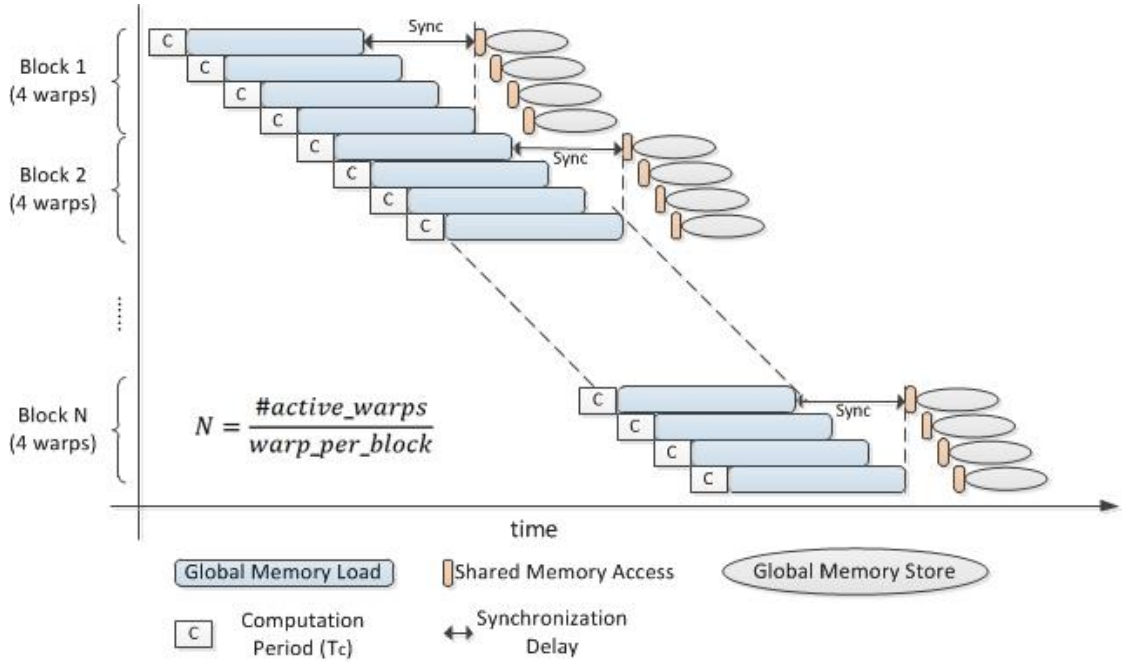


Figure 5.8: Execution time pipeline of coalesced transpose kernel with one group of active warps. This kernel is global memory access intensive.

G. Limitations and Discussion

There are several limitations for our performance model. First, our performance model cannot predict asymmetric kernels very accurately due to limitations of current

GPU hardware counters. This can be improved in the future if CUPTI is extended to provide additional global counters. Second, we use a simple global memory model to facilitate estimation but in reality the situation may be more complicated. Third, some amount of inaccuracy is attributable to our kernel code analysis. Currently, our model can guide users to identify potential bottlenecks but does not point out which lines of codes are problematic. We plan to develop a tracing tool for this purpose in future work.

5.6 Combined Energy Model for GPU Based Clusters

Equation (5.14) provides a general equation for calculating a parallel system's total energy consumption. It employs average power and delay (i.e. execution time). While power P_i ($i \epsilon$ *set of all nodes* Ω) describes the rate of energy consumption at a discrete point in time on node i , energy E specifies the total number of joules in the time interval ($t_1 1$, $t_1 2$) , as a sum product of the average power $\overline{P_i}$($i \epsilon$ *set of all nodes* Ω) and delay ($Delay = t_1 2 - t_1 1$) :

$$E = \sum_{i(i \epsilon \Omega)}^{\Omega} \int_{t_1}^{t_2} P_i(t) dt = \sum_{i(i \epsilon \Omega)}^{\Omega} \overline{P_i} \times Delay \qquad (5.14)$$

Based on (5.14), we derive a simple energy model specifically for a GPU-based cluster. Assuming we have a set of hybrid nodes Ω (nodes are exactly the same) and each node has multi-core processors and multi-GPUs on it, we denote the set of GPUs as

λ. We also have $i \in \Omega$ and $k \in \lambda$. So we can estimate the energy consumption of this hybrid system as:

$$\forall i,k, i \in \Omega, k \in \lambda, E = \sum_{i=1}^{\Omega}[(\sum_{k=1}^{\lambda} \overline{P_g^{ik}} \cdot t_{gpu}) + \overline{P}_{idle} \cdot t_{gpu} + E_{para-overhead}] \quad (5.15)$$

$$t_{gpu} = t_{pci} + t_{kernel} \quad (5.16)$$

In this thesis, we have not focused on modeling the host multi-core system because our goal is to provide an energy estimation method for GPU-based clusters. Please refer to our previous work [18][26] for detailed component-level energy [4, 30] modeling for multi-core systems.

5.7 Experimental Results

A. Experimental Setup

For experimental results of our power and performance models in this section, we use an NVIDIA Tesla C2075 as our test bed. For the energy model validation and analysis for scalable GPU-based systems, we use the newly updated Keeneland cluster [198] which has three NVIDIA Tesla M2090 on each node. Hardware specifications from the C2075 and M2090 GPUs are shown in Table III.

Table 5.4: Hardware Specifications for Fermi GPUs Used in This Study

Device	C2075	M2090
Capability	2.0	2.0
$\sharp SM$	14	16
$\sharp SP\ (CUDA\ cores)$	32	32
GPU clock rate	1.15 GHz	1.3GHz
Memory clock rate	1.53 GHz	1.8GHz
Memory bus width	384-bit	384-bit
Memory size	6GB	5.24GB
Memory bandwidth	116.7GB/s	117.8GB/s
L2 size	0.75MB	0.75MB

In order to validate the accuracy of our models and conduct detailed analyses, we use the CUDA SDK 4.1 kernels [213], GEM package [200], and SHOC benchmark suit [201] for our study. These applications cover a wide range of execution and memory access patterns. All the power measurements, performance counters, and execution time data are the average of at least 10 runs.

B. Model Accuracy Analysis

1) ***Evaluation of BP-ANN Based GPU Power Model:*** We first conduct a model accuracy test by comparing average power prediction for 20 CUDA kernels using our proposed BP-ANN model, and the best available multiple linear regression model from Nagasaka et al. [49] (denoted ”MLR” in our graphs). We also compare our model to a revised version of MLR (denoted ”MLR+” in our graphs). In this work, MLR excludes texture events from the training set in order to be consistent with the model from Nagasaka et al. [49] while MLR+ is an improved version we created to include texture events — we believe this to be a natural extension of their work.

To guarantee fairness, we use the same training samples for all three models. To

evaluate model accuracy, we collect samples using the format shown in Section 5.4-B-(2) from runs of CUDA kernels covering various execution patterns. Then we randomly partition the entire training data set into 10 subsets and conduct 10-fold cross validation [220].

Figure 5.9 shows these cross-validation results. According to Figure 5.9, for each individual kernel, we observe that BP-ANN consistently outperforms MLR and MLR+ in power prediction accuracy. For several kernels, MLR and MLR+ demonstrate much higher prediction errors than BP-ANN: for instance, 15 watts less than actual power for SobelQRNG, 17 watts more than actual power for Kernel Idle, 26 watts more than actual power for HSOpticalFlow, and 15 watts more than actual power for dwtHarr1D. One major reason causing this is the pre-defined linear relationship between training variables and the output by MLR and MLR+. MLR+ (trained with texture cache access counters)

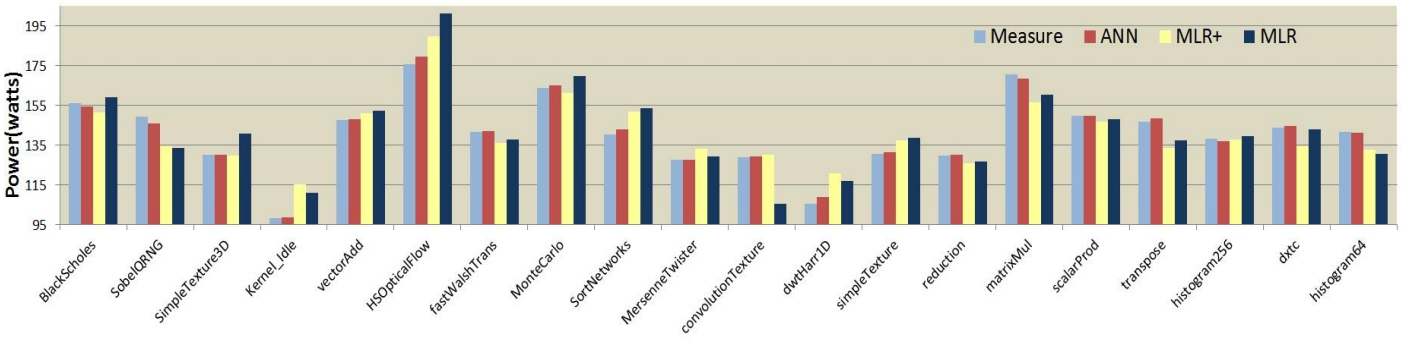


Figure 5.9: Prediction accuracy comparisons of BP-ANN, MLR and MLR+ for 20 CUDA kernels

does not show consistently better performance than MLR, which indirectly indicates that conventional linear regression models have less flexibility and adaptivity than a BP-ANN based model for predicting GPU power consumption.

Across 20 evaluated kernels, our BP-ANN based power model has a 2.1%

average prediction error rate, while MLR has 6.9% and MLR+ has 6.3%. However, average prediction error rate across all the evaluated kernels is too general for evaluating model accuracy since prediction accuracy for individual kernels is vital to end users. For instance, even with 6.9% average error rate across all kernels, MLR still performs poorly with a 15 to 30 watts misprediction for several kernels. To clarify, we additionally evaluate using Root Mean Square Error (RMSE) for model accuracy and robustness. RMSE is defined as $RMSE = \sqrt{\frac{1}{n}\sum_{k=1}^{n}(S_m^k - S_p^k)^2}$, where k is sample id, n is number of samples, $\mathbf{S_m^k}$ is measured power, and $\mathbf{S_p^k}$ is predicted power. The higher the RMSE value, the farther the prediction is from the actual measurement. Figure 5.11 illustrates the RMSE values for the individual CUDA kernels for all three models. It shows that BP-ANN's predictions have smaller deviation than those of MLR and MLR+, which makes

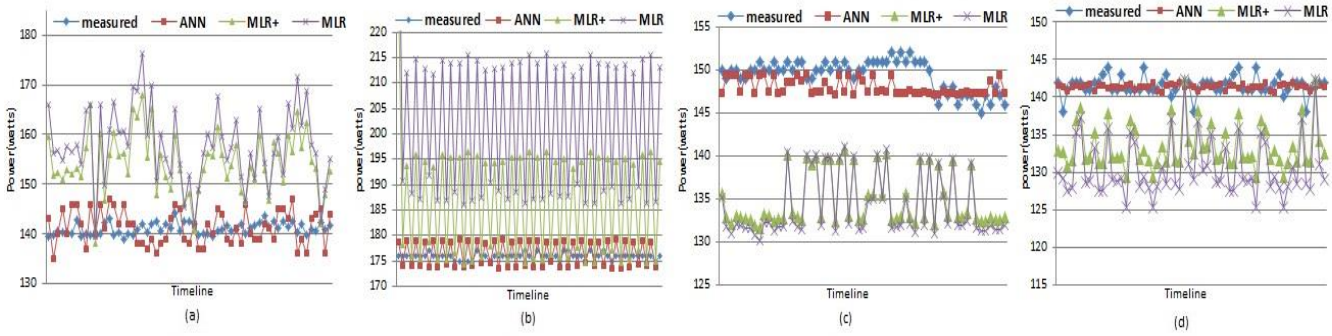


Figure 5.10: Power prediction comparisons of four CUDA SDK kernels that have been observed with high RMSE values. (a)SortNetworks (b)HSOpticalFlow (c)SobelQRNG (d)Histogram64. The timeline is plotting the multiple repetitions of the same kernel.

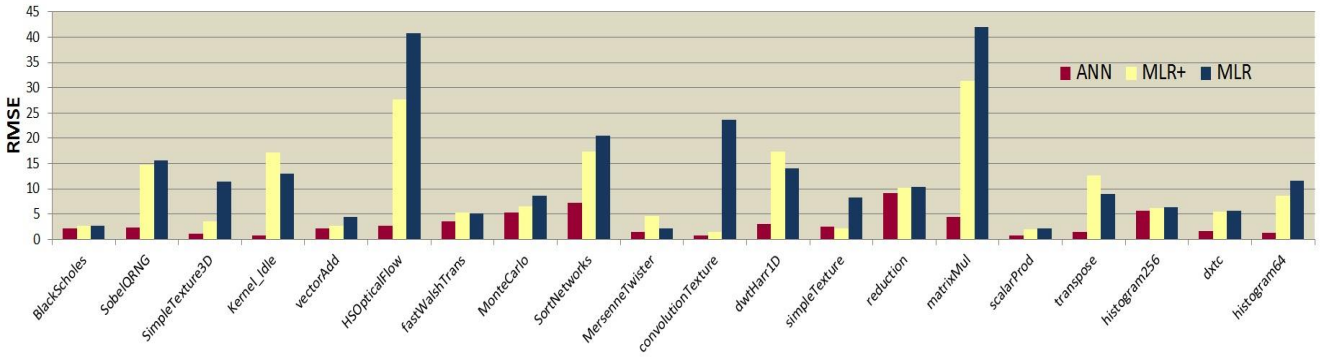


Figure 5.11: RMSE value for each CUDA kernel by using three different models.

BP-ANN more accurate and robust for individual kernel prediction. Figure 5.10(a)(b)(c) and (d) show the model power prediction comparisons of four sample kernels which have been observed with higher RMSE values in Figure 5.11. Each graph shows that BP-ANN tracks measured power more closely than MLR and MLR+ for the duration of each kernel. This strongly indicates that our BP-ANN based power model has less deviation and is very effective to predict both instantaneous and average power consumption.

2) ***Evaluation of The Performance Model:*** Figure 5.12 shows the measured and predicted (using our performance model illustrated in Section 5.5) execution time for

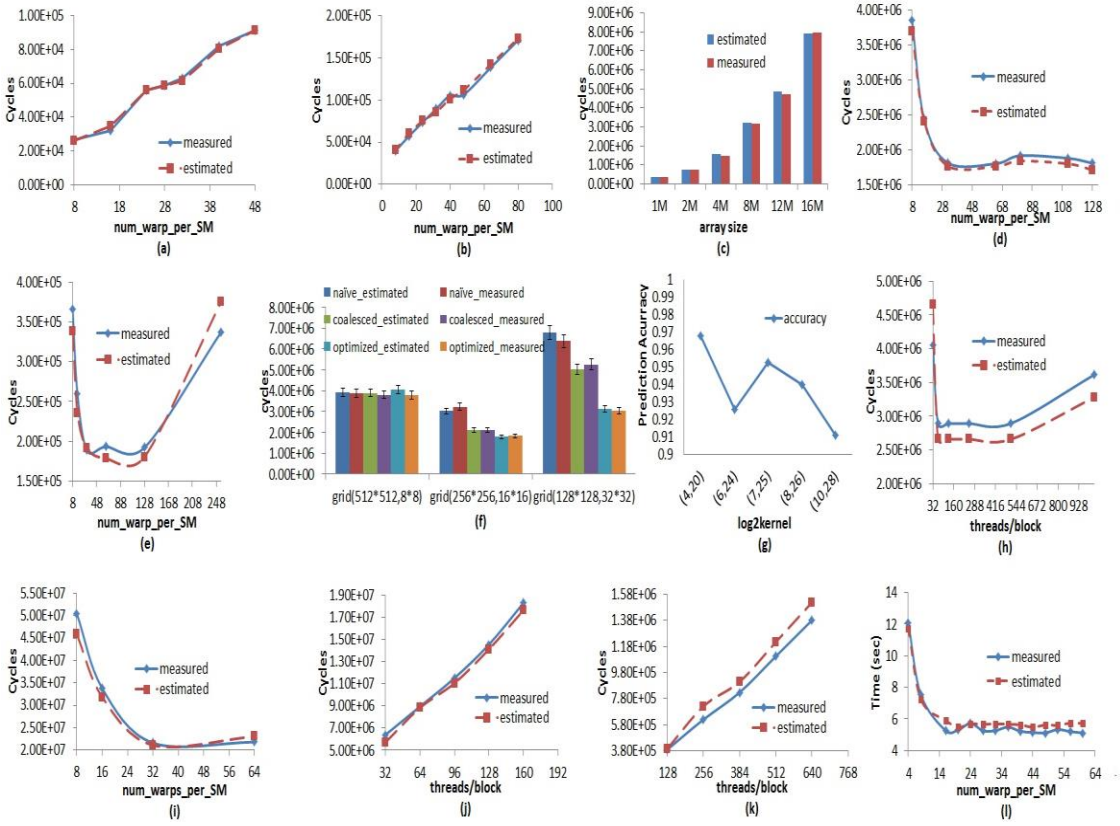


Figure 5.12: Predicted execution time using our performance model for a dozen CUDA kernels. (a)Matrix multiply naive (b)Matrix multiply tiling (c) Parallel reduction (d)BlackScholes (e)ScalaProd (f)Transpose (g)FastWalshTransform (h)ConvolutionTexture (i)Histogram256 (j)MersenneTwister (k)QuasirandomGenerator (l) GEM(calc potential single step dev kernel).

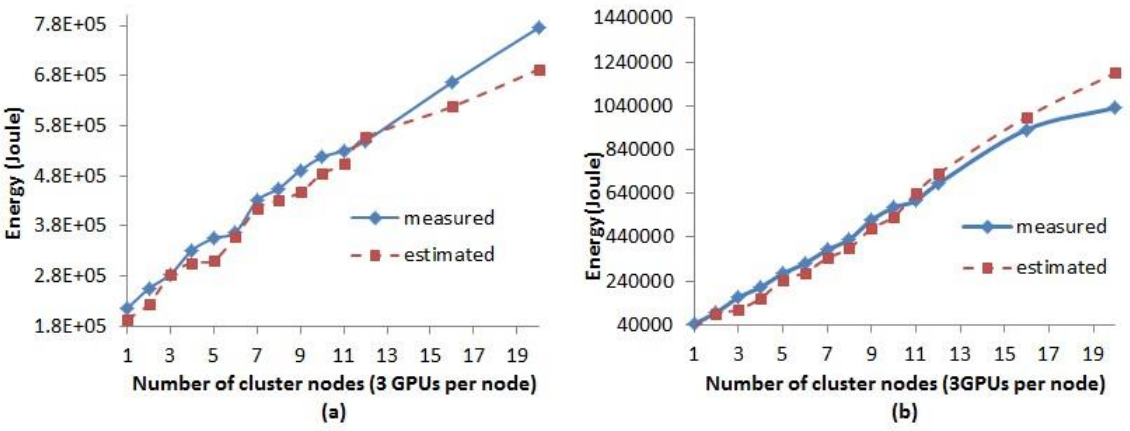


Figure 5.13: Comparison of the measured and estimated total energy consumption of 20 nodes Keeneland (3 GPUs per node) running with (a) TP-QT clustering and (b) TP-Reduction.

CUDA kernels with various execution patterns (including SDK kernels and GEM). To obtain the inputs for the performance model, we calculated the accumulated values of the selected counters for the critical region by recording them before and after each GPU kernel. We predicted the individual execution time using kernel specific input parameters from the original CUDA SDK codes. Across all the cases shown in Figure 5.12, the average absolute prediction error rate is within 6.7% using our proposed performance model. The figure also shows that our performance model can generally predict the performance while varying the number of warps or different configurations (e.g. (c) with different problem size). For instance, case (d)(e)(h)(i)(l) all have fixed workload when scaling the number of warps per SM and our estimations generally predict the performance accurately. For (e) and (h), the performance degradation when using larger number of warps per SM or blocks can be caused by long waiting queues for groups of warps to be executed and register spills to the local memory. Figure 5.12 (i) uses seconds to represent execution time instead of cycles because GEM code is composed of

iterations of calc_potential_single_step_dev kernel and our prediction is against the overall execution time.

3) ***Evaluation of the Combined Energy Model:*** We apply our energy model to the Keeneland cluster to conduct model accuracy analysis. Since the SHOC benchmarks we use are not network intensive workloads, we simply assume Pnic = 0. Across all the "True Parallel" (TP, with both communication and synchronization) CUDA applications in SHOC, our results indicate that the model accurately predicts total system energy consumption within 8.9% error on average. For instance, Figure 5.13 shows the accuracy test for QT Clustering (QTC) and Parallel Reduction. QTC is a strong scaling case while Reduction exhibits weak scaling. Figure 5.13 indicates that our energy model can predict the general trend of total system energy consumption while system size scales (the average prediction error rate for QTC is 6.6% and for Reduction is 11.02%).

C. Usage of The Models

1) ***Identifying Power Bottlenecks:*** Figure 5.14 shows the access ratios of major power-related GPU components for a dozen CUDA kernels (from Figure 5.9) covering various memory access patterns. For different memory accesses, the component access ratio is calculated as the number of the actual transactions per SM divided by the total number of executed instructions per SM. For arithmetic computation, it is the number of computation instructions per SM divided by total executed instructions per SM. The component access ratio gives a general picture of what factors contribute to overall GPU power consumption. By cross-comparing Figure 5.9 and Figure 5.14, we can determine that kernels with a higher global memory access ratio (i.e. component access ratio for global memory accesses) generally consume more power than the others. Computation

intensity is another major contributor due to on-chip SM power. Shared, texture and local memory accesses can also impact power but not as significantly. For instance, scalarProd consumes more power on average than fastWalshTransform and dwtHarr1D due to its higher global memory access ratio even though it has a lower shared memory access ratio. Using the correlation between training events and power, programmers and architects can identify power bottlenecks and their causes. Then, they can implement software and architecture level optimizations accordingly.

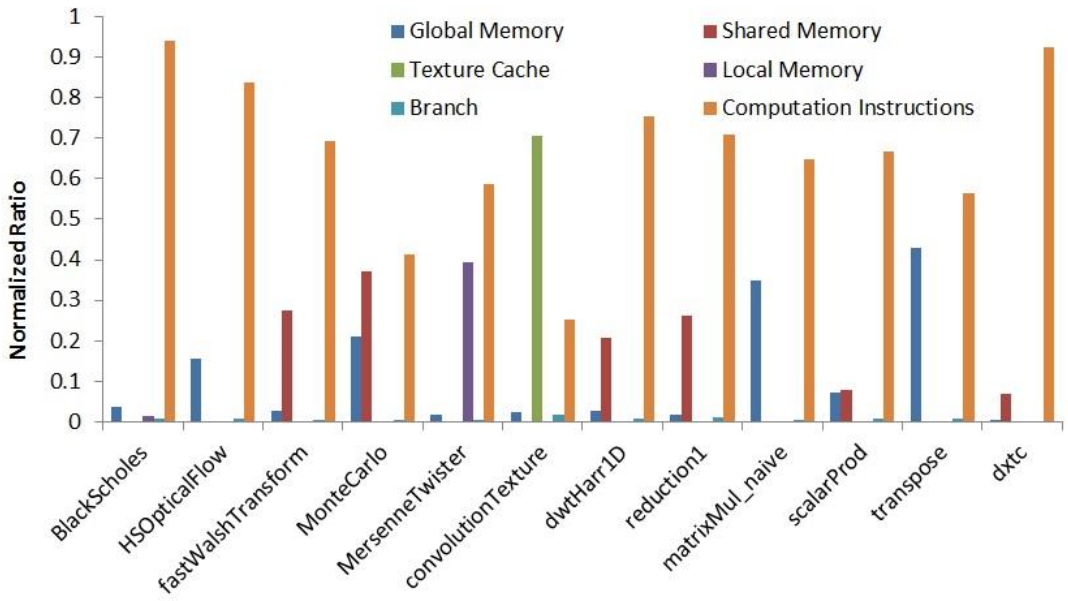


Figure 5.14: The normalized access ratios of the major power-related GPU components for several representative CUDA kernels. Each application on the x-axis includes six component bars, starting from blue and ending with orange. Zero access ratio for a component results in "no bar" for that application.

2) ***Identifying Potential Performance Bottlenecks:*** In addition to the approach

introduced in Section 5.5-F (estimating the time spent on the individual components and then comparing them to identify bottlenecks), we can also use estimated execution time and performance events to guide the optimization process. Figure 5.15 demonstrates this iterative process. After eliminating all the bottlenecks step by step using this method, programmers may reach a state where there no more further improvements can be gained through code optimizations. Further optimizations may be possible through compiler and micro-architecture level changes but are outside the scope of this work.

3) ***Predicting The Optimal Power Efficiency:*** We can combine the BP-ANN power model with the performance model to predict the power efficiency (normalized performance/watt) for CUDA kernels running on a single GPU. For instance, Fig. 5.16(a)

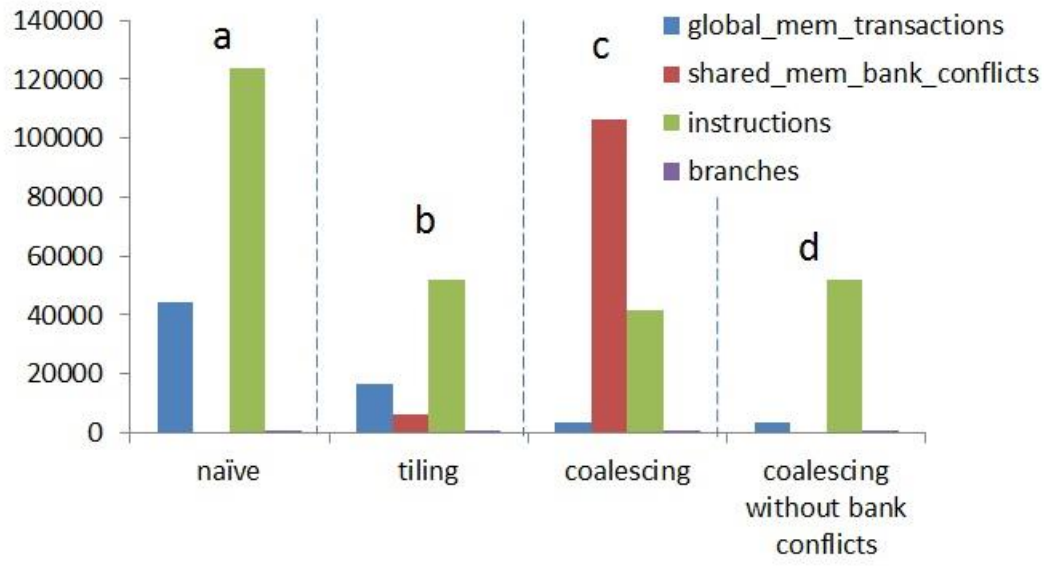


Figure 5.15: Performance bottleneck identification and optimization process using MatrixMul as an example. For execution time, a > b > c > d. Optimization process: a-> b ->c ->d. a:identify global memory access as the primary bottleneck. b: reduce the usage of global memory by tiling. c: further reduce global memory usage by coalescing memory access and using shared memory. Shared memory access becomes the new bottleneck due to bank conflicts. d: eliminating shared bank conflicts.

shows the power measurement and prediction (from BP-ANN model) for GEM codes while varying the number of warps per SM under the fixed workload (nucleosome input). We can observe that the power prediction generally matches the actual power. Figure 5.16(b) demonstrates power efficiency prediction using a combination of our power and performance models (see Figure 5.12-(l) for the performance prediction results on GEM). The prediction shows power efficiency is highest when number of warps per SM is around 20. This power efficiency value is the second best result from the actual measurement where the best number of warps per SM is 44.

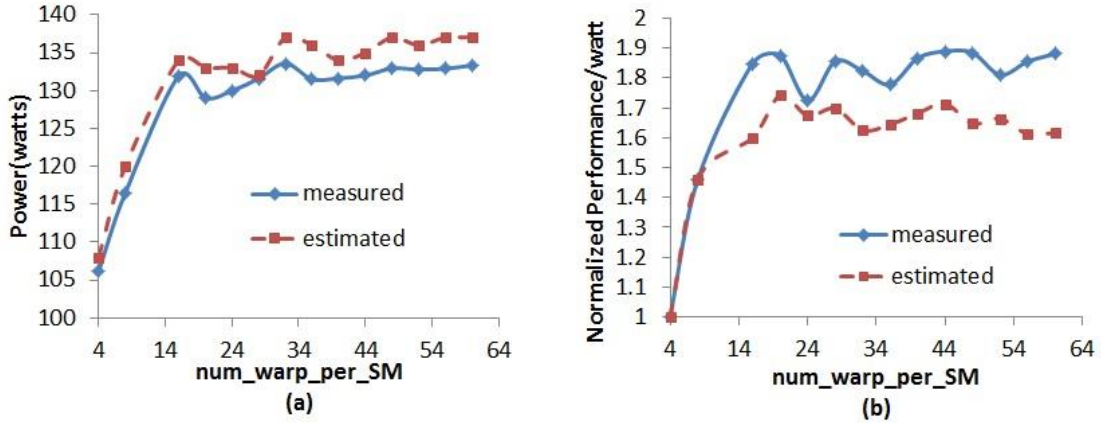


Figure 5.16: (a) Power measurement and prediction (BP-ANN) for GEM code while varying the number of warps per SM under the same workload; (b) power efficiency prediction using our models

4) ***Studying Energy-Performance Trade-offs In an HPC Environment***: In order to study the system energy-performance trade-offs, we use our energy model to predict the Energy-Delay product [85]. The Energy-Delay product $E^{\alpha}D^{\beta}$ ($and\ \beta$ are integers

and ≥ 0) aligns energy-performance trade-offs to users' priorities by allowing them to weight energy and delay using α and β respectively. The smaller the value of $E^{\alpha}D^{\beta}$, the better the system energy-performance efficiency. Figure 5.17 demonstrates that our combined energy model can predict the optimal or near optimal number of cluster nodes to run QTC in order to gain the minimum Energy-Delay products. For ED (emphasize both energy and performance equally), our model predicts 5 as the optimal number of nodes which is close to the optimal 6 nodes indicated as optimal by direct measurement. For E2D (emphasize saving energy), measurement indicates 3 nodes while our model predicts 2 nodes. For ED2 (emphasize performance), our model predicts 20 nodes as the optimal configuration which matches the measurement. The same methodology can be

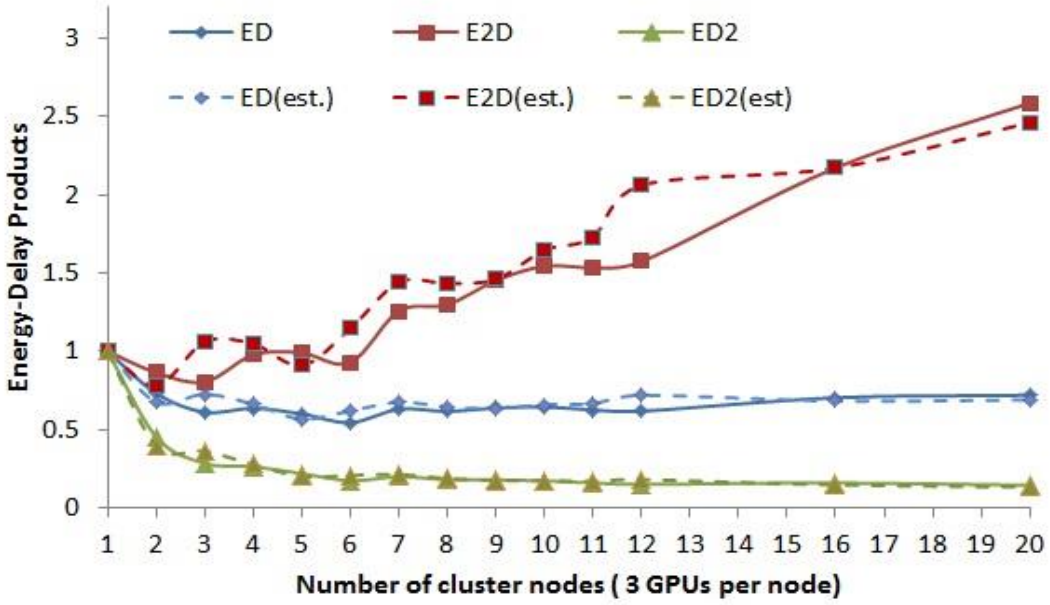


Figure 5.17: Predicting the optimal number of cluster nodes to achieve the best Energy-Delay products for QT Clustering Benchmark using our energy model. — represents measured values. - - - represents predicted values for GEM code. Both use 128 threads/block.

applied to predict energy-performance trade-offs for other parallel applications running on GPU based clusters.

5.8 Chapter Summary

In order to design highly efficient heterogeneous systems for future exascale systems, we need to quantitatively understand power, performance, and their interactive effects—energy consumption at scale. In this chapter, we combine hardware performance counters with machine learning and advanced analytics to model power-performance efficiency for emergent GPU architectures and GPU based HPC systems. Experimental results show that our models are simple, accurate, and cover various computation and memory access patterns presented by emergent GPU architectures such as Fermi. We also show how to use our models to identify power-performance bottlenecks, predict power efficiency for a single GPU-based system, and to study the energy-performance efficiency for GPU based clusters. In future, we want to combine our multi-core iso-energy- efficiency model with the current GPU energy model to further investigate the relationship between energy efficiency and system-level parameters.

Chapter 6

Designing Energy Efficient Communication Runtime Systems: A View from PGAS Models

In this chapter, we present a design for Power Aware One-Sided Communication Library –PASCoL. The proposed design detects communication slack, leverages Dynamic Voltage and Frequency Scaling (DVFS), and Interrupt-driven execution to exploit the detected slack for energy efficiency [59, 60]. We implement our design and evaluate it using synthetic benchmarks for one-sided communication primitives, *Put*, *Get*, and *Accumulate* and uniformly noncontiguous data transfers. Our performance evaluation indicates that we can achieve significant reduction in energy consumption without performance loss on multiple one-sided communication primitives. The achieved results are close to the theoretical peak available with the experimental test bed.

6.1 Introduction

Parallel applications in a wide range of scientific domains are being designed to use the proposed exascale systems. Message Passing Interface has become the *de facto* standard for writing these applications. However, several of these scientific domains are a natural fit for Partitioned Global Address Space (PGAS) models. These models provide abstractions for distributed data structures (Arrays, Trees, etc.) and primitives for one-sided data transfer to provide load balancing with different execution paradigms. PGAS models use one-sided communication runtime systems to achieve scalability and high performance, while providing abstractions from variability of networks.

As the runtime systems continue to evolve, many architectural innovations for energy efficient computing are being proposed in the literature and becoming available with commodity architectures. User-space abstractions such as Dynamic Voltage and Frequency Scaling (DVFS) have become available, which allow a user process to dynamically change the frequency and voltage of processing elements. High-speed networks such as InfiniBand, BlueGene [221], and Quadrics [222] provide methods for interrupt based notification of data transfer—a powerful mechanism which may be used to exploit communication slack.

In this chapter, we design a Power Aware One-Sided Communication Library PASCoL) using Aggregate Remote Memory Copy Interface (ARMCI) [223], which leverages the architectural and network abstractions to exploit the communication slack and achieve energy efficiency. We lay down the design issues of various one-sided communication primitives and associated communication protocols for different datatypes, specifically focusing on contiguous and uniformly noncontiguous datatypes as

a use case from many scientific applications. We implement our design and evaluate it using synthetic benchmarks on an InfiniBand Cluster. Our performance evaluation with benchmarks using various one-sided communication primitives shows that we can achieve significant energy efficiency with negligible performance degradation. The observed energy efficiency is close to the theoretical peak provided by the experimental test bed.

The rest of the chapter is organized as follows. In Section 6.2, we present the background of our work. In Section 6.3, we present the design of energy efficient communication runtime system—PASCoL using ARMCI. In Section 6.4, we present the performance evaluation of PASCoL using synthetic benchmarks on an InfiniBand cluster. At last, we conclude and present our future works in Section 6.5.

6.2 Background of One-Sided Communication Runtime Systems

Many one-sided communication runtime systems have emerged to serve the requirements of programming models. MPI-Remote Memory Access, Global Address Space Network (GASNet) [224], Aggregate Remote Memory Copy Interface (ARMCI) [223], Low Level API (LAPI) [225], and Deep Computing Messaging Framework (DCMF) [221], are examples of one-sided communication runtime systems, which provide put, get, and accumulate communication primitives. We specifically focus on ARMCI [223] in this work.

The ARMCI [223] communication runtime system provides a general-purpose,

efficient, and widely portable one-sided communication operations optimized for contiguous and noncontiguous (strided, scatter/gather, I/O vector) data transfers. In addition, ARMCI includes a set of atomic and mutual exclusion operations. ARMCI exploits native network communication interfaces and system resources (such as shared

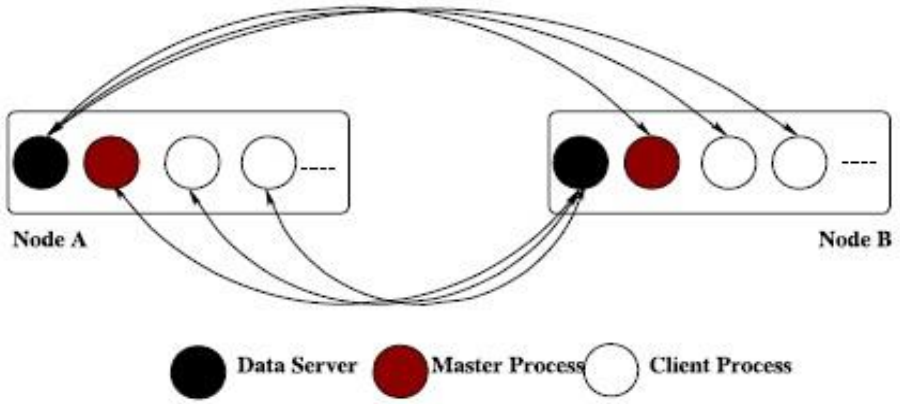


Figure 6.1. Communication structure in ARMCI.

memory) to achieve the best possible performance of the remote memory access/one-sided communication. Optimized implementations of ARMCI are available for the Cray Portals, Myrinet (GM and MX) [226], Quadrics [222], Giganet (VIA), and Infiniband (using OpenFabrics and Mellanox Verbs API) [142]. It is also available for leadership class machines including Cray XT4, XT5, XE6, and BlueGene/P [221].

Figure 6.1 shows the communication structure in ARMCI. The terminology between processes on the same node is differentiated to facilitate the implementation of one-sided communication primitives. The process with lowest rank on a node is called *master* and the rest of the processes on the node are called *clients*. The master process creates a thread, *data server*, which is used as an agent for remote asynchronous progress.

A request for global memory allocation is served by a shared memory segment visible to all processes on a node. One-sided communication occurs only between global address space data, precluding the requirement of client–client communication. The data server is used in designing protocols which may be efficiently implemented using copy based approach. Efficient protocols which require bulk atomic updates (such as accumulates) may also be designed using the data server. Depending on the workload, the network and the communication protocol, the data server *may or may not need to be active all the time*.

6.3 Design Methodology

In this section, we present the overall design of an energy efficient one-sided communication runtime system. We explore the alternatives for energy efficiency—DVFS and Interrupt driven execution and use them to design energy efficient protocols for one-sided communication primitives.

6.3.1 Mechanisms for Energy Efficiency

There are multiple mechanisms available for designing energy efficient one-sided communication protocols which are complementary. A combination of these protocols may be used as follows:

– Interrupt based execution allows multiple stages of communication protocols to transition using event driven mechanisms. As an alternative to polling—typically used in

high performance computing applications, this method allows much lower CPU utilization to save energy, particularly if enough communication slack is available to be exploited.

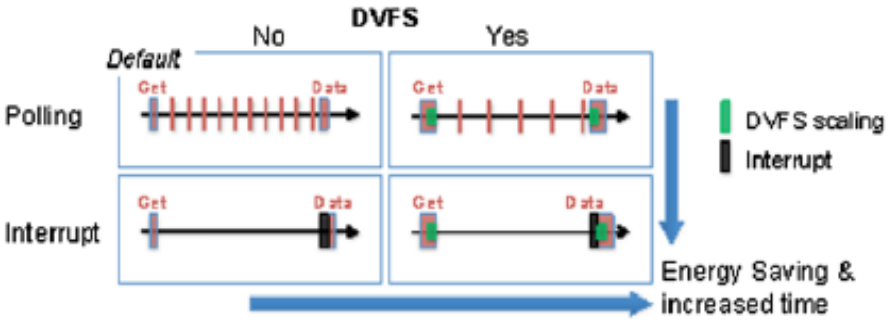


Figure 6.2: Mechanisms for energy optimizations.

– DVFS can improve energy efficiency by reducing the frequency and voltage of processors, typically on a per-core basis (frequency) and on a per-socket basis (voltage). Different communication protocols require varying CPU utilization during different phases. Energy efficiency can be achieved using DVFS, if the communication slack is much higher than the overhead of transitioning between frequency/ voltage states.

The use of interrupt based execution and DVFS is shown in Figure 6.2 using an example of a one-sided get operation. After the operation is executed, the data becomes available at a later point as shown on the time-line. The default case uses neither the interrupt nor DVFS mechanisms. Polling in this case is indicated by regular activities along the time-line between the get primitive and arrival of data (upper-left in the figure).

Combining DVFS and polling (upper-right), the frequency of polling is reduced due to reduced processor frequency but at an expense of transitions between DVFS states. Polling is completely reduced by using interrupts but can increase in latency results due to interrupt handling.

Keeping the mechanisms discussed above in mind, we design and implement an energy efficient one-sided communication runtime system. While our framework allows full DVFS scaling due to the limitations of our experimental setup, we are able to use frequency scaling only.

6.3.2 Energy Efficient Protocols for Contiguous Data Transfer

One-sided communication of contiguous data transfer is the primary case for using the Remote Direct Memory Access (RDMA) mechanism provided by most Interconnects such as InfiniBand [227], Quadrics [222], BlueGene [221], Cray Gemini. RDMA mechanism requires the source and target buffer to be registered for most networks. Let *source* and *target* represent the source and the target buffers respectively used for the one-sided communication primitive. Let *registered* be the function which checks whether a buffer has been registered. The following algorithm is executed at the client process:

if *registered(source) && registered(target)* ***then***

Use RDMA method

else

Copy Data in Intermediate Transmission Buffer

Send Data

end if

When the source and target buffers are registered, the overall communication slack is entirely the network data transfer. A combination of Interrupt based execution and DVFS is used depending on the expected communication slack. Section 6.4 helps us define the thresholds for using these mechanisms.

In the above algorithm, if either of the source or the target buffer is not registered, a copy-based communication protocol is used. Since data copy is compute intensive, the DVFS mechanism is used after the copy operation is complete. The interrupt driven execution is used after the data transfer request is completed.

6.3.3 Handling Noncontiguous Data Transfer

Handling noncontiguous data types for energy efficiency is of critical importance to many application domains. Some of these domains use distributed arrays and perform communication on Cartesian blocks of data. This results in uniformly noncontiguous (strided) data transfer. Energy efficient communication protocols for strided data transfer is pivotal for these applications.

Many communication protocols have been proposed for handling strided data communication. Some high-speed networks support strided communication natively by using scatter/gather mechanisms. However, these mechanisms result in high context memory utilization. Networks which provide high concurrency may also use pipelined data transfer. However, when individual data size is small, such a protocol results in

significant overhead.

Let *flatten* be a utility which converts a strided data to a contiguous data by copying it in *Intermediate Transmission Buffers*. A simplified protocol which is applicable to a wide variety of strided communication primitives is presented below:

Flatten the strided buffer

Copy Data in Intermediate Transmission Buffer

Send Data

The algorithm for *flattening the strided buffer* is a recursive operation, resulting in pipelined data transfer. The interrupt driven execution may be used in conjunction with DVFS if the pipelined buffer size is above a particular threshold to exploit communication slack.

6.3.4 Energy Efficient Asynchronous Agent

To provide asynchronous progress of one-sided communication operations on remote node(s), each node uses an asynchronous agent, such as the data server thread in ARMCI. The asynchronous agent is not involved when RDMA is used for data transfer between user-level buffers. The asynchronous agent is active when a copy based protocol is used for data transfer. Hence, the agent may be active only during these phases of communication. In the PASCoL design, we use a combination of interrupt driven execution and DVFS for the asynchronous agent. The frequency is scaled up after an interrupt has been received and scaled down just before blocking on the interrupt based

execution.

6.3.5 Discussion

In this section, we discuss the issues currently not considered in PASCoL. We specifically focus on atomic memory operations and synchronization methods:

Atomic memory operations*:* Atomic memory operations are widely used in many applications for load balancing, enabling passive synchronization etc. In PASCoL, we have not considered optimizing atomic operations except accumulate operations, which are typically performed on a large data block. Word-based atomic operations are latency sensitive and using DVFS/interrupt based execution may result in significant overhead. We plan to address this limitation in the future that may provide guidelines for leveraging the interrupt based execution with DVFS.

Synchronization methods*:* One-sided communication runtime systems provide active and passive modes of synchronization. With active synchronization, origin and target processes are involved in the synchronization. With passive synchronization, only the origin process is involved. ARMCI supports only active mode of synchronization. The synchronization operation may be optimized by possibly time-stamping the outstanding requests, providing an estimate of the communication slack. Interrupt based execution with DVFS may be used if the expected communication slack is above a threshold. Currently, PASCoL does not handle energy efficient synchronization. We plan to address this limitation in the near future.

6.4 Power and Performance Evaluation of PASCoL

In this section, we present a performance evaluation of PASCoL using synthetic benchmarks designed with ARMCI communication primitives. For one-sided communication primitives—put, get, accumulate, and put strided, we present the relative latency, relative energy consumption per megabyte of transfer, and relative power consumption of a combination of DVFS and Interrupt/polling methodologies. We begin with a description of the Experimental Test bed.

6.4.1 Experimental Test Bed

We use the Northwest ICE (NW-ICE) test bed at Pacific Northwest National Lab for power and performance evaluation. The NW-ICE cluster has 192 compute nodes, inter-connected with DDR InfiniBand network adapters and switches. Each NW-ICE node is an Intel Xeon E5345 dual socket quad core CPU with 2.33 GHz frequency. Each node has 16 GBytes of main memory with each core having a 32 kB cache size. Using DVFS, NW-ICE allows frequencies of 2.33 GHz and 1.9 GHz. By default all processes execute at 2.33 GHz frequency. The interface for changing the frequencies is through a memory resident file system.

ESDC Monitoring: Real-time data center energy efficiency depends on real-time data streaming from all the power consuming hardware in a data center, as well as data

acquisition and reduction software. PNNL has developed a real-time software tool, FRED (Fundamental Research in Energy Efficient Data Centers), to monitor, analyze, and store data from the ESDC-TB facility instrumentation. FRED's underlying technology is derived from PNNL's experience in developing power plant, distribution, and facility monitoring and diagnostic systems for applications ranging from nuclear power generation to building management. We use the real-time software tool of FRED to analyze the energy consumption for various synthetic benchmarks.

FRED consists of the ESDC-TB monitoring system, the Environmental and Molecular Science (EMSL) facility monitoring system, a data collector, a central database, and a web-based graphical user interface (GUI) client. The ESDC-TB monitoring system derives from PNNL's Decision Support for Operations and Maintenance (DSOM) software (R&D100Award), an advanced, flexible diagnostic monitoring application for energy supply and demand systems. The ESDC-TB monitoring system interfaces to auxiliary data acquisition systems that collect data specific to NW-ICE.

6.4.2 Performance Evaluation Methodology

In this section, we present the performance methodology for evaluating the power and performance of PASCoL. We design pure communication benchmarks using the one-sided communication primitives—put, get, accumulate, and put strided.

To study the impact of approaches proposed in the previous section, we design a shift communication pattern benchmark using each of these communication primitives. Unlike MPI based communication benchmark—which implicitly synchronizes the

communicating processes, the shift benchmark designed with one-sided communication primitives synchronizes the memory associated with communication. The following algorithm presents an example of designing the shift benchmark using put primitive:

```
start timer
for j = 0 to iterations do
  for i = 0 to numprocs do
    dest←myid +i
    put(data) to dest
    fence to dest
  end for
end for
end timer
```

Similarly, the shift benchmark is designed by replacing the corresponding put primitive with get, accumulate, and put strided primitives. A total of four combinations are used for comparison—polling, polling + DVFS, Interrupt, and Interrupt + DVFS. These combinations are used to compare the performance of evaluation metrics discussed below.

6.4.3 Evaluation Metrics

There are three evaluation metrics which are used for evaluation of PASCoL and comparing the performance of various approaches presented above. The fundamental metric is the latency observed by each of the approaches. Another metric of interest is the power consumption of the approaches. However, each of the above metrics may not be individually sufficient. We propose a derived metric—Energy consumed relative to volume of data transfer (energy/MByte). We specifically focus on the thresholds beyond which the power and energy/Mbyte may be improved without an increase in latency.

6.4.4 Results

In this section, we present the evaluation of PASCoL for each of the communication primitives using the metrics presented above, while comparing the

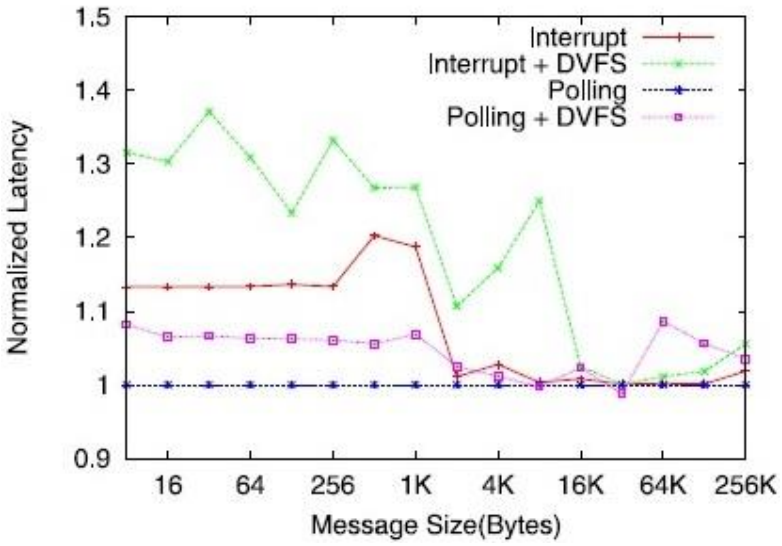


Figure 6.3: Normalized Latency for ARMCI put.

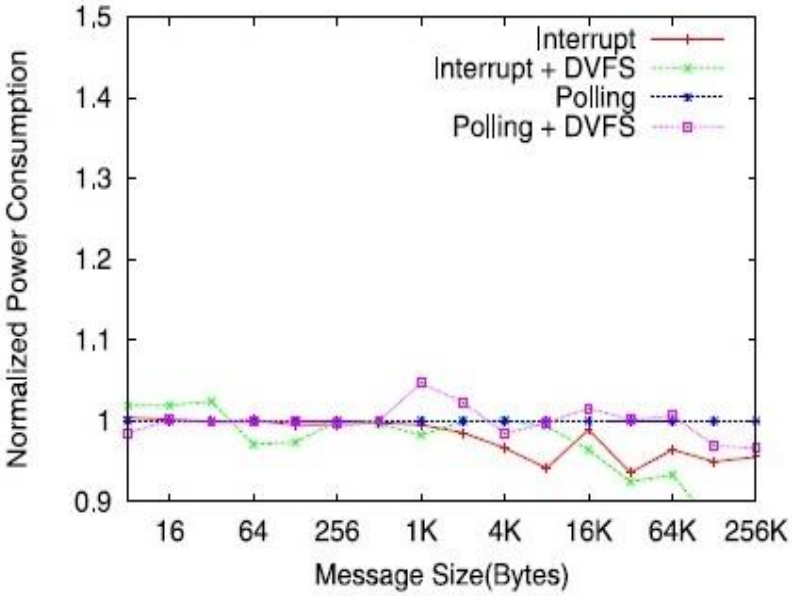


Figure 6.4: Normalized Power Consumption for ARMCI put.

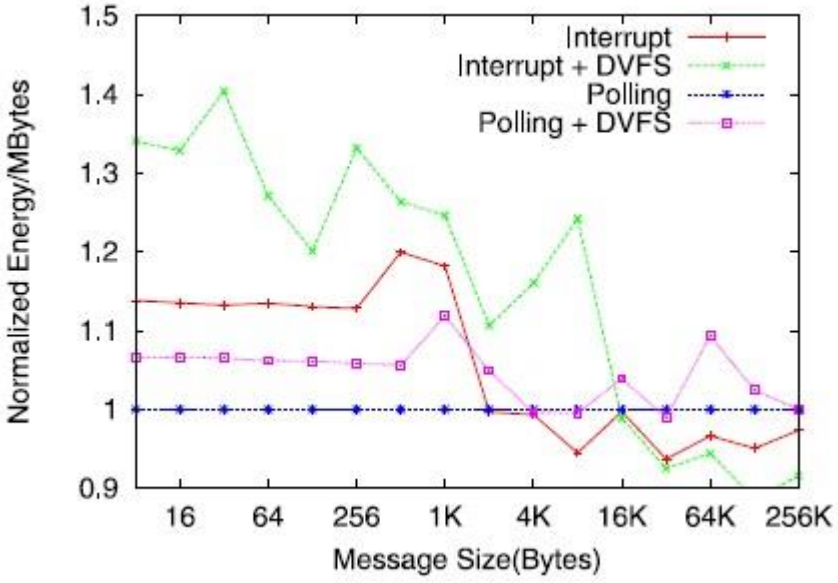


Figure 6.5: Normalized Energy/Mbytes for ARMCI put.

performance of the approaches—Polling, Polling + DVFS, Interrupt, Interrupt + DVFS. The performance results are normalized with the polling approach—the default methodology for most one-sided communication runtime systems.

Figures 6.3, 6.4, and 6.5 show the normalized latency, power consumption and Energy/Mbyte for the shift communication benchmark using the put one-sided primitive on 64 processes, respectively. The polling approach outperforms other approaches for small and medium size messages, due to their sensitivity to latency and significant overhead of using the Interrupt and DVFS mechanisms. We observe spikes for Interrupt and DVFS based approaches, due to the limitations of our current test bed, as the sampling is available once every 5 seconds.

With increasing message size, the latency for multiple approaches converges significantly (less than 5% difference). At 16 KBytes message size, we observe that the latency for all approaches (with a slight exception to Polling + DVFS), converges. A similar trend is observed in the relative power consumption of these approaches. For messages at 256 KBytes, the Interrupt + DVFS approach provides an improvement of 12% in power consumption in comparison to the polling approach. Smaller improvements are also observed in the power consumption of other approaches.

At the same time, Energy/Mbyte consumption for the Interrupts + DVFS approach improves significantly compared to the other approaches. Overall, we observe an improvement of 8% in the Energy/Mbytes using Interrupts with DVFS compared to the default polling case, while an improvement of 5% is observed compared to the Interrupts scheme. For large messages, the overhead incurred by interrupts is amortized by the overall time of data transfer. We observe that a threshold of 16 KBytes can be used for using Interrupts with DVFS without significant performance degradation.

Figures 6.6, 6.7, and 6.8 show the normalized latency, Power consumption and Energy/Mbyte for the shift communication benchmark using the get one-sided primitive on 64 processes, respectively. We observe the trends in the performance similar to the shift communication benchmark designed using the put communication primitive. The polling approach outperforms other approaches for all the evaluation metrics and small messages. The limitation of the sampling rates produces spikes, which are prominent for smaller messages and less for the larger messages. An improvement of about 11% is observed in the relative power consumption by Interrupt + DVFS approach, while much lesser improvements are observed for Interrupt and DVFS only approaches. The Interrupt

approach produces an out-liar at 4 KBytes message, which is still under observation.

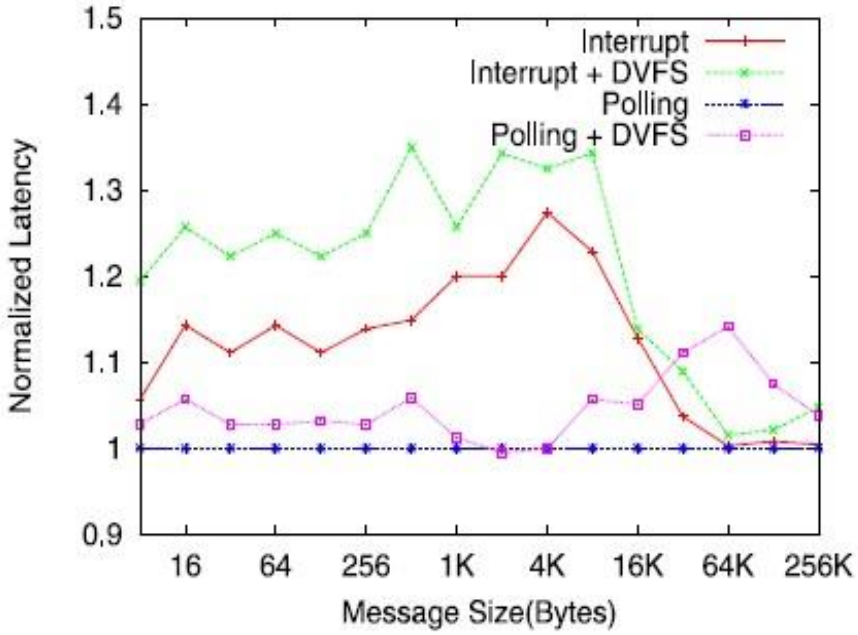


Figure 6.6: Normalized Latency for ARMCI get.

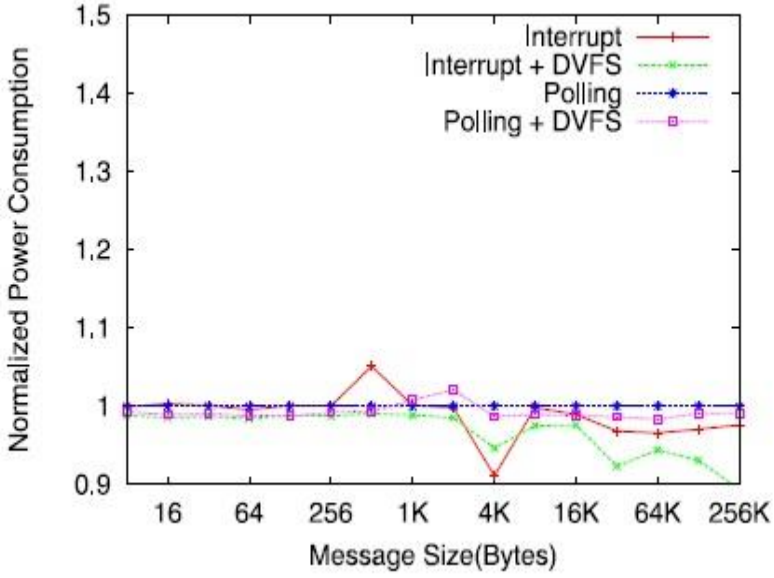


Figure 6.7: Normalized Power Consumption for ARMCI get.

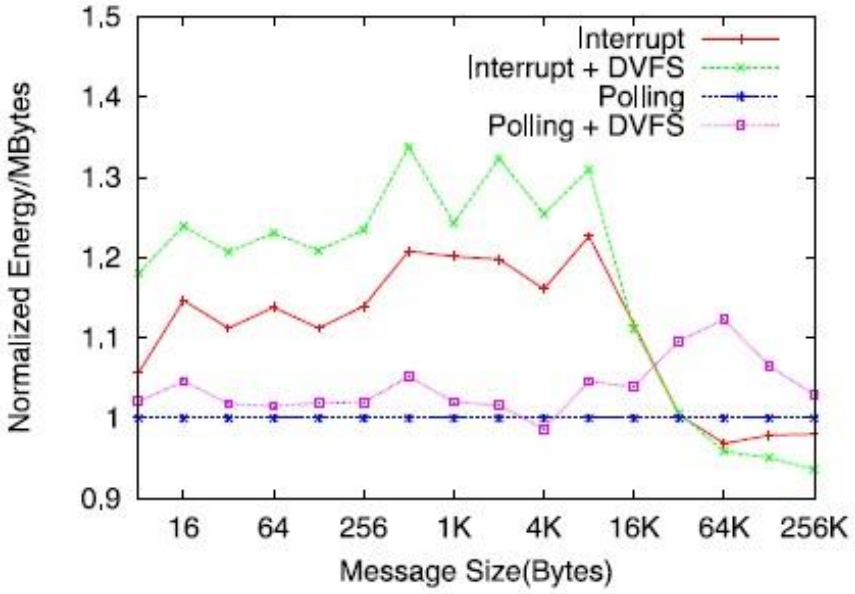


Figure 6.8: Normalized Energy/Mbytes for ARMCI get.

Similar trends are observed for energy/Mbytes metric, where interrupts + DVFS approach outperforms other approaches for larger messages, but incurs significant overhead for small messages.

Figures 6.9, 6.10, and 6.11 show the normalized latency, Power consumption and Energy/ Mbyte for the shift communication benchmark using the accumulated one-sided primitive on 64 processes, respectively. The shift communication benchmarks using put, and get primitives use RDMA and the associated communication protocol does not involve the asynchronous agent. The accumulated one-sided communication primitive uses the pipelined data transfer, and involves the asynchronous agent for remote progress.

As presented previously, the pipelined communication protocol flattens the buffer and uses the copy based approach. The copy phase and the atomic update phases of the

protocol are CPU intensive. Hence, we do not use DVFS during this phase. As a result,

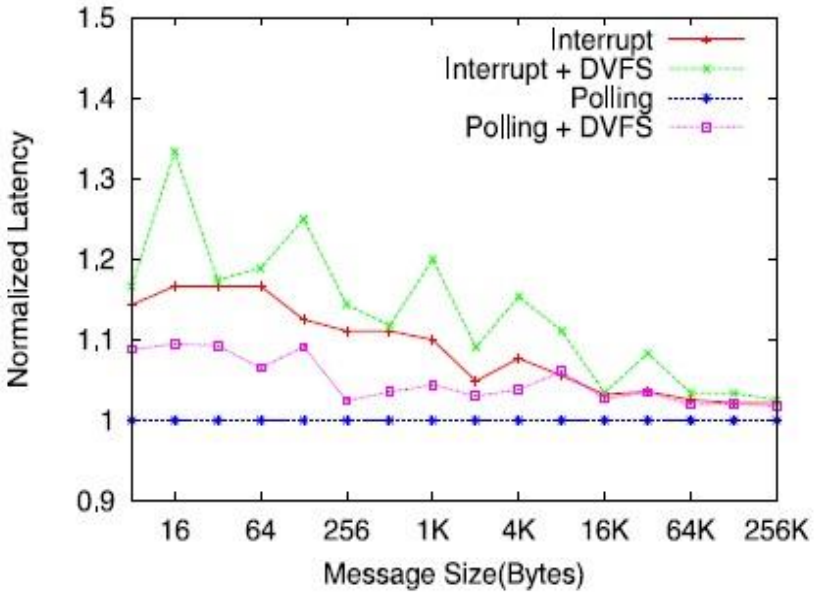


Figure 6.9: Normalized Latency for ARMCI accumulate.

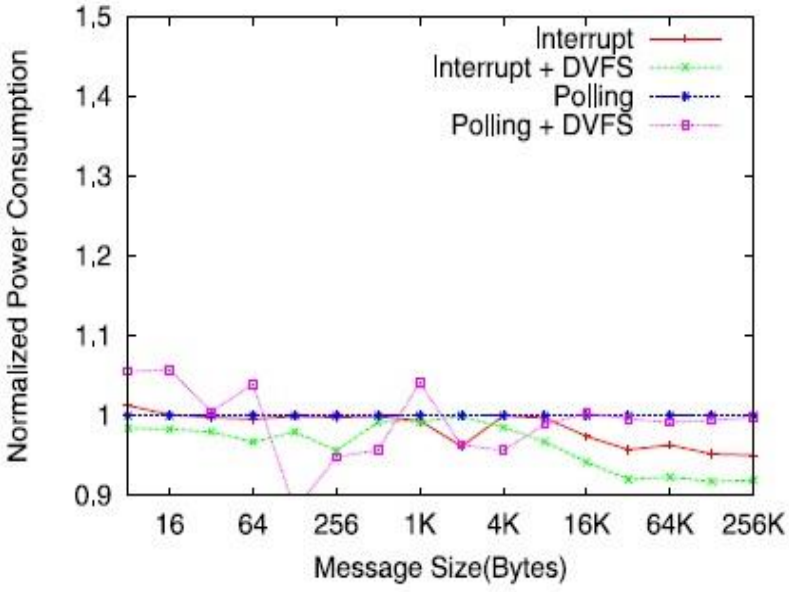


Figure 6.10: Normalized Power Consumption for ARMCI accumulate.

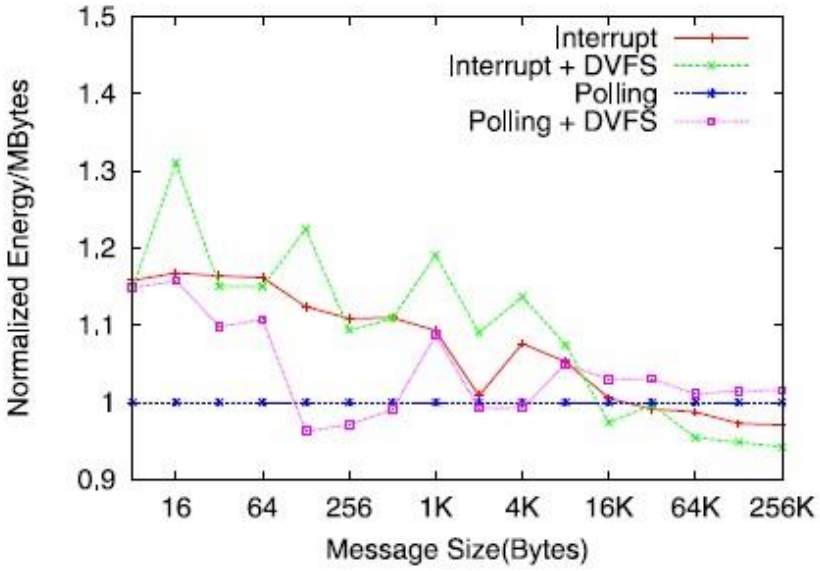


Figure 6.11: Normalized Energy/Mbytes for ARMCI accumulate.

the overall improvement in relative power consumption and energy/mbyte is reduced in comparison to the contiguous data transfer. The improvement in relative power consumption is 8%, while the improvement in Energy/Mbyte is about 6%.

Figures 6.12, 6.13, and 6.14 show the normalized latency, power consumption and energy/ Mbyte for the shift communication benchmark using the put strided one-sided primitive on 64 processes, respectively. The strided communication primitives use copy based approach for data transfer. Similar to the shift communication bench mark using accumulate primitive, the relative latencies converge for messages beyond 32 Kbytes and significant improvements in the relative power consumption and energy/Mbyte are also observed.

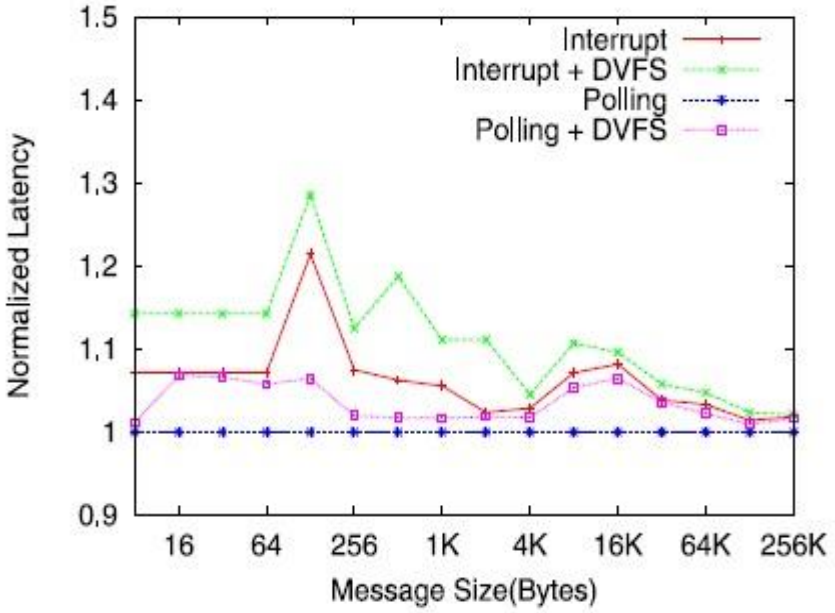


Figure 6.12: Normalized Latency for ARMCI put strided.

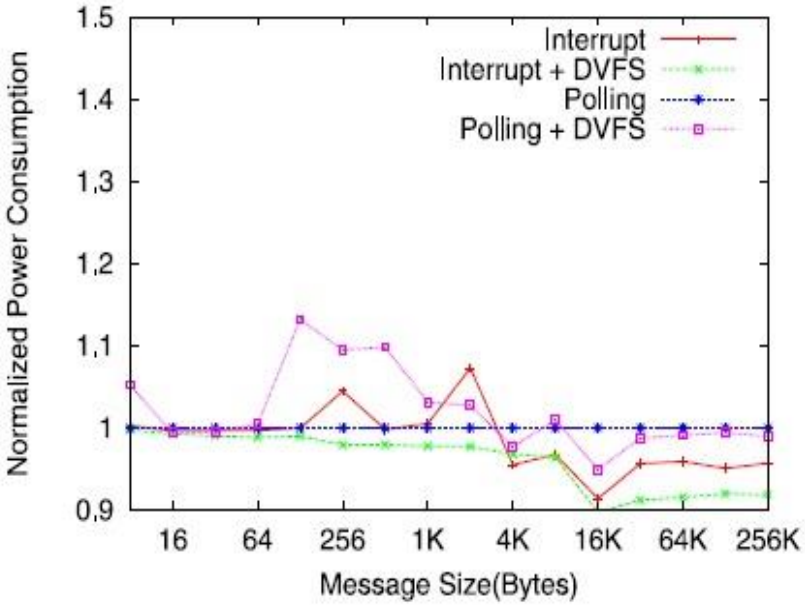


Figure 6.13: Normalized Power Consumption for ARMCI put strided.

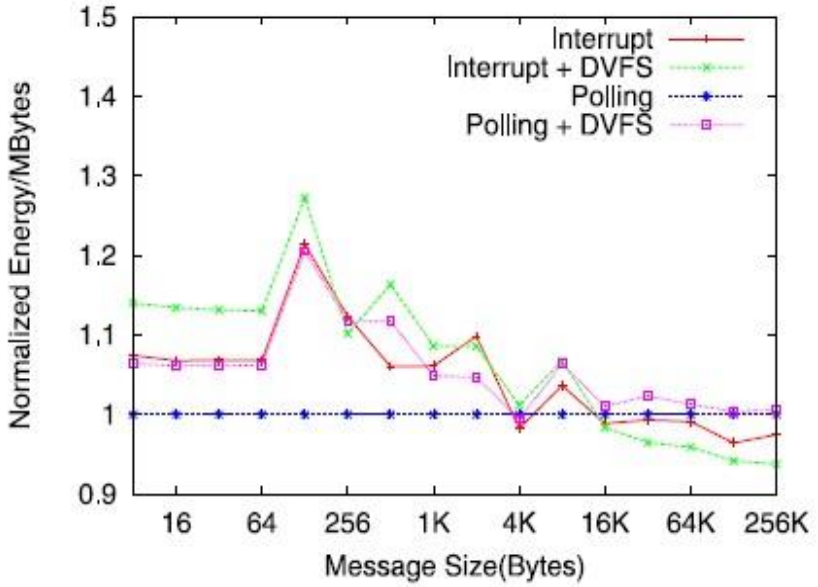


Figure 6.14: Normalized Energy/Mbytes for ARMCI put strided.

6. 4.5 Discussion

As presented in the previous section, a combination of interrupt based execution and DVFS mechanisms provide the maximum energy efficiency for large messages with each of the communication primitives. A summary of the results is presented in the Table 6.1 for each of the put, get, accumulate, and strided put primitives. In this table, we also indicate the increase in latency resulting from the DVFS transitions and the interrupt handling, and also the overall decrease in energy consumption. These metrics are presented relative to the default case of no DVFS and use of polling. Also listed in Table 6.1 is the message size at which increased energy efficiency is observed.

As we described in the previous section, the NW-ICE test bed used for the experimentation has its limitations. In particular, there are only two processor-core

frequency levels available—1.9 GHz and 2.33 GHz. This limits the potential energy

Table 6.1: Energy Efficiency Advantages for Different Communication Semantics

Primitive	Min message size (KB)	Increase in latency (%)	Decrease in energy (%)
Get	32	5	6
Put	16	5	6
Accumulate	16	3	5
Strided Put	16	2	6

efficiency that may be observed. Assuming that the processor core power consumption is directly proportional to the frequency then the difference in these two states represents a power difference of 22% and affects just the power used by the processors. Measurements made on NW-ICE on a rack basis indicate that an idle rack consumes 7.8 kW, and a rack containing all processor-cores performing the all-to-all benchmark consumes 9.4 kW resulting in a 16% difference. These two observations are inline with each other as the rack-based measurements include everything in the rack and not just the processors.

The energy improvements observed, as listed in Table 6.1 shows that a significant portion of the 16% maximum savings is being realized when using the Interrupt and DVFS energy saving mechanisms. Current state of the art processors including the Intel Nehalem series and the AMD Magny-Cours have a greater number of DVFS states in comparison to our experimental test bed. In addition, the overhead of transitioning between DVFS states is expected to reduce in future processor generations. These two factors impact expected energy savings in two ways:

– The greater number of DVFS states has a greater potential for energy savings for large-messages when using the combination of Interrupt and DVFS mechanisms for one-sided communications.

– The decrease in transition overheads should reduce the message size at which energy savings will occur.

6.5 Chapter Summary

In this work, we have designed a Power Aware One-Sided Communication Library (PASCoL) using Aggregate Remote Memory Copy Interface (ARMCI) [223], which leverages the architectural and network abstractions to exploit the communication slack for energy efficiency. We have laid down the issues involving various one-sided communication primitives and associated communication protocols for different datatypes, specifically focusing on contiguous and uniformly noncontiguous datatypes as a use case from many scientific applications. We have implemented our design and evaluated it using synthetic benchmarks on an InfiniBand Cluster. Our performance evaluation with benchmarks using various one-sided communication primitives has demonstrated that we can achieve significant energy efficiency with negligible performance degradation. The observed energy efficiency is close to the theoretical peak provided by the experimental test bed.

We plan to continue design and development of energy efficient one-sided communication protocols for different platform and high speed communication networks.

We also plan to evaluate the efficacy of these designs on large scale systems using scientific applications such as NWChem [90] and Subsurface Transport over Multiple Phases (STOMP).

Chapter 7

Conclusions and Future Work

7.1 Conclusions

This thesis presents theories, techniques and a software framework to quantitatively understand the relationship between power and performance for high performance systems at scale. I have designed three major efficiency-directed components in this thesis: 1) measuring, profiling and evaluating techniques for power/performance of large scale HPC systems; 2) modeling and analysis framework for building highly efficient systems; 3) energy/performance optimization runtime systems and tools. Specifically, our contributions and findings presented in this work include:

- **Measuring, profiling and evaluating techniques for power/performance of large scale HPC systems.** I evaluated the power and performance profiles of

the HPC Challenge benchmark suite on a multi-core-based HPC cluster. I used the PowerPack toolkit to collect the power and energy profiles and isolate them by components. I organized and analyzed the power and energy profiles according to each benchmark's memory access locality. I discovered that the power profiles of the HPCC benchmark suite reveal power boundaries for real applications. Applications with high spatial and temporal memory access locality consume the most power but achieve the highest energy efficiency. Applications with low spatial or temporal locality usually consume less power but also achieve lower energy efficiency.

- **Modeling and analysis framework for building highly efficient systems:** I present a system level energy efficiency model for various parallel applications and large scale parallel system architectures. I extend the concept of performance isoefficiency to iso-energy-efficiency and show how to build an accurate system level energy efficiency model step by step. Then I apply our analytical model to real scientific applications from various mainstream parallel applications and illustrate how to derive essential model parameters to predict total system energy consumption and efficiency for large scale parallel systems. I perform a thorough and detailed investigation of machine and application dependent parameters which have nontrivial impact on system energy efficiency. I apply my model to three scientific benchmarks representing different execution patterns to study the influential factors for system energy efficiency and explore how to scale these key factors to maintain efficiency. I also extended the methodology to modeling popular accelerators (i.e.: GPGPU) in

the system so my models can accommodate contemporary HPC systems with various hybrid architectures.

- **Energy/performance optimization runtime systems and tools:** I have designed a Power Aware One-Sided Communication Library (PASCoL) using the Aggregate Remote Memory Copy Interface (ARMCI), which leverages architectural and network abstractions to exploit communication slack for energy efficiency. I have laid down the issues involving various one-sided communication primitives and associated communication protocols for different datatypes, specifically focusing on contiguous and uniformly noncontiguous datatypes as a use case from many scientific applications. I have implemented our design and evaluated it using synthetic benchmarks on an InfiniBand Cluster. My performance evaluation with benchmarks using various one-sided communication primitives has demonstrated that I can achieve significant energy efficiency with negligible performance degradation. The observed energy efficiency is close to the theoretical peak provided by the experimental test bed.

7.2 Future Work

High Performance Computing (HPC) research is thriving. Nevertheless, HPC research remains a complex topic which often requires knowledge from various aspects of computer science such as architecture, systems, algorithms, programming languages and software. I have summarized several research topics that I believe will become the

essential challenges for the HPC research community in the coming years. In the near future, I want to extend my current research in the following directions:

- **Improving Energy Efficiency in Emergent Systems**: Undoubtedly, power is going to remain a major constraint on future HPC designs. My research will be centered on developing system and architecture level strategies to meet the future "energy wall" challenge. I plan to combine runtime stats (i.e. hardware performance counters and instantaneous system status), optimization strategies (i.e. multilevel DVFS, concurrency throttling and power-aware hardware design) with modeling techniques (i.e. statistical and machine learning) to create accurate, portable and agile energy–aware systems. Eventually, such systems should provide auto-tuning and self-improving functions. I also plan to implement power-aware features on top of the commonly used libraries and programming models (i.e. MPI, OpenMP, MapReduce) in order to hide the underlying complexity and help programmers write more energy efficient codes with less effort than required today.
- **Leverage Accelerators For HPC**: Advances in architectural design inspire new software-based research. In the past, I have been working on various aspects of new architectures including both homogenous (i.e. multicore NUMA) and heterogeneous architectures (i.e. GPU and MIC). It is critical that we continue to improve our understanding of the relationship between performance and power. Without a deep understanding of the tradeoffs, we

cannot hope to fully optimize system energy efficiency and fully utilize the emerging class of supercomputers. Issues that contribute to performance loss such as irregular memory access, data locality, multi-level parallelism, and inefficient parallel I/O must be addressed. For instance, NVIDIA Kepler's new functionality "Hyper-Q" slashes CPU idle time by allowing multiple CPU cores to simultaneously utilize a single GPU. With this feature, we can potentially design a more efficient kernel scheduler to achieve optimal performance or energy efficiency.

- **Exploiting Big Data in HPC**: I am interested in two aspects of big data research. First, I feel data locality must be exploited in big data solutions. In virtual machine environments, I plan to study data access patterns among processes and then merge the data that share similar patterns at run time on fewer virtual machines. In this way, I/O time can potentially be dramatically reduced. Second, I plan to study how to efficiently analyze big data using scalable approaches such as MapReduce. Power-Performance Analysis
- **Automation in Emergent Systems**: Due to the complexity of today's parallel codes, we need more automated mechanisms to identify the performance/power bottlenecks and idioms that represent regions of code with high latencies. I am planning to conduct research on this subject with the following goals: 1) automate identification of bottlenecks and their root causes; and 2) evaluate regions of code to automatically determine whether they will benefit from accelerators (e.g. GPUs).

Appendix

Table A: Parameters Used in Iso-energy-efficiency Model

Parameters	Machine Dependent Parameters
$\mathbf{W_c}$	Total on-chip computation workload
$\mathbf{W_m}$	Total off-chip memory access workload.
$\mathbf{W_{co}}$	Total parallel computation overhead
$\mathbf{W_{mo}}$	Total number of memory access overhead in parallelization
$\mathbf{M}$	Total number of messages packaged in parallelization
$\mathbf{B}$	Total number of bytes transmitted
p	Number of homogeneous processors available for computing the workloads
N	Problem size or total amount of work (in instructions or computations)
α	Corrector factor including components such as overlap among computation, memory access and network transmission, additional cost by code scaling, etc.
T_o	Total overhead time due to parallelism
T_1	Total sequential execution time of an application running on a single processor
$M(N)$	Memory utilization function. $M(N) = function(N)$
Parameters	**Application Dependent parameters**
Time related	
$\mathbf{t_c}$	$\frac{CPI_{on}}{f}$. [15].Average time per on-chip computation instruction (including on-chip caches and registers)
$\mathbf{t_m}$	Average memory access latency
$\mathbf{t_{msg}}$	Average start up time to send a message
$\mathbf{t_{Byte}}$	Average time of transmitting a 8-bits word
$\mathbf{T_{IO}}$	Total I/O access time
Power related	
$\mathbf{P_{c-on}}$	Average CPU power in running state
$\mathbf{P_{c-idle}}$	Average CPU power in idle state
$\mathbf{\Delta P_c}$	$\mathbf{P_{c-on} - P_{c-idle}}$
$\mathbf{P_{m-on}}$	Average memory power in running state
$\mathbf{P_{m-idle}}$	Average memory power in idle state
$\mathbf{\Delta P_m}$	$\mathbf{P_{m-on} - P_{m-idle}}$
$\mathbf{P_{IO-on}}$	Average IO device power in running state
$\mathbf{P_{IO-idle}}$	Average IO device power in idle state
$\mathbf{\Delta P_{IO}}$	$\mathbf{P_{IO-on} - P_{IO-idle}}$
$\mathbf{P_{other}}$	Average sum of other devices' power such as motherboard, System/CPU fans, NIC, etc.
$\mathbf{P_{total-idle}}$	Average system power on idle state
f	The clock *frequency* in clock cycles per second
θ	Energy efficiency or user desired energy efficiency. For *N* scaling case in IV, $0 < \theta \leq 1$; for *f* scaling case, $\theta > 0$ and possibly larger than 1 due to normalized to lowest frequency on one

	processor.

Bibliography

[1] T. Agerwala, "Exascale computing: The challenges and opportunities in the next decade," in *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, 2010, pp. 1-1.

[2] Steve Ashby, Pete Beckman , Jackie Chen , Phil Colella, B. Collins , Dona Crawford, *et al.*, "Report on Exascale Computing," Report on Exascale Computing, Department of Energy2010.

[3] M. Tolentino and K. W. Cameron, "The Optimist, the Pessimist, and the Global Race to Exascale in 20 Megawatts," *Computer*, vol. 45, pp. 95-97, 2012.

[4] S. Shuaiwen, S. Chun-Yi, G. Rong, A. Vishnu, and K. W. Cameron, "Iso-Energy-Efficiency: An Approach to Power-Constrained Parallel Computation," in *Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International*, 2011, pp. 128-139.

[5] W. Feng, "Making a Case for Efficient Supercomputing," *ACM Queue*, pp. 54-64, Oct, 2003.

[6] K. W. Cameron, R. Ge, and X. Feng, "High-performance, power-aware distributed computing for scientific applications " *Computer*, vol. 38, pp. 40-47, 2005.

[7] *2010 US department of energy annual report*. Available: http://www.eia.doe.gov/

[8] *"TOP500 Supercomputing Project"*. Available: http://www.top500.org

[9] (2012, November). *Green 500 list*. Available: http://www.green500.org/lists/2011/11/top/list.php

[10] (2011). *K Computer at RIKEN release*. Available: http://www.green500.org/lists/2011/11/top/list.php

[11] H. Wei, M. Allen-Ware, J. B. Carter, E. Cheng, K. Skadron, and M. R. Stan, "Temperature-Aware Architecture: Lessons and Opportunities," *Micro, IEEE*, vol. 31, pp. 82-86, 2011.

[12] J. R. Black, "Electromigration—A brief survey and some recent results," *Electron Devices, IEEE Transactions on*, vol. 16, pp. 338-347, 1969.

[13] D. K. Schroder and J. A. Babcock, "Negative bias temperature instability: Road to cross in deep submicron silicon semiconductor manufacturing," *Journal of Applied Physics*, vol. 94, pp. 1-18, 2003.

[14] W. Feng and K. W. Cameron, "The Green500 List: Encouraging Sustainable Supercomputing," *Computer*, vol. 40, pp. 50 - 55 Dec. 2007.

[15] D. A. Patterson and J.L.Hennessy, *Computer Architecture: A quantitative approach*, 3rd ed. San Francisco, CA: Morgan Kaufmann Publishers, 2003.

[16] J. Donald and M. Martonosi, "Leveraging Simultaneous Multithreading for Adaptive Thermal Control," in 2nd workshop on Temperature-Aware Computer Systems," in *TACS-2*, 2005.

[17] J. Donald and M. Martonosi, "Techniques for Multicore Thermal Management: Classification and New Exploration,," presented at the 33rd International Symposium on Computer Architecture (ISCA-33), 2006.
[18] M. D. Powell, M. Gomaa, and T. N. Vijaykumar, "Heat-and-Run: Leveraging SMT and CMP to Manage Power Density through Operating System,," presented at the International Conference on Architectural Support for Programming Languages and Operating Systems(ASPLOS), Boston, MA, 2004.
[19] H. K. Pyla, D. Li, and K. W. Cameron, "Poster: Thermal-aware High-performance Computing Using TEMPEST," in *19th IEEE/ACM International Conference on High Performance Computing and Communications (SC07)*, Reno, NV, 2007.
[20] A. Snavely and D. M. Tullsen, "Symbiotic Job scheduling for a Simultaneous Multithreading Processor," presented at the International Conference on Architectural Support for Programming Languages and Operating Systems(ASPLOS), 2000.
[21] K. Choi, R. Soma, and M. Pedram, "Dynamic voltage and frequency scaling based on workload decomposition," presented at the Proceedings of the 2004 international symposium on Low power electronics and design, Newport Beach, California, USA, 2004.
[22] O. Sarood, A. Gupta, and L. V. Kale, "Temperature Aware Load Balancing for Parallel Applications: Preliminary Work," presented at the Proceedings of the 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and PhD Forum, 2011.
[23] O. Sarood and L. V. Kale, "A 'cool' load balancer for parallel applications," presented at the Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, Seattle, Washington, 2011.
[24] H.K.Pyla, "Tempest: A Framework for High Performance Thermal-Aware Distributed Computing,," Masters in Computer Science, Computer Science, Virginia Tech, Blacksburg, 2007.
[25] D. M. Brooks, P. Bose, S. E. Schuster, H.Jacobson, P. N. Kudva, A. Buyuktosunoglu, *et al.*, "Power-Aware Microarchitecture: Design and Modeling Challenges for Next-Generation Microprocessors," *IEEE Micro*, 2000.
[26] K. Skadron, T. Abdelzaher, and M. R. Stan, "Control-Theoretic Techniques and Thermal-RC Modeling for Accurate and Localized Dynamic Thermal Management," presented at the Eighth International Symposium on High-Performance Computer Architecture, 2002.
[27] S. Murugesan, "Harnessing Green IT: Principles and Practices," *IEEE IT Professional*, pp. 24-33, January- Feburary 2008.
[28] R. S. C. D. Patel and C. E. Bash., "Smart cooling of datacenters.," in *IPACK'03: The PacicRim/ASME International Electronics Packaging Technical Conference and Exhibition.*, 2003.
[29] R. F. Sullivan, "Alternating Cold and Hot Aisles Provides More Reliable Cooling for Server Farms," Uptime Institute,, White Paper2000.
[30] S. Shuaiwen, M. Grove, and K. W. Cameron, "An ISO-Energy-Efficient Approach to Scalable System Power-Performance Optimization," in *Cluster*

Computing (CLUSTER), 2011 IEEE International Conference on, 2011, pp. 262-271.

[31] K. J. Nowka, G. D. Carpenter, E. W. MacDonald, H. C. Ngo, B. C. Brock, K. I. Ishii, *et al.*, "A 32-bit PowerPC system-on-a-chip with support for dynamic voltage scaling and dynamic frequency scaling," *Solid-State Circuits, IEEE Journal of*, vol. 37, pp. 1441-1447, 2002.

[32] G. Magklis, G. Semeraro, D. H. Albonesi, S. G. Dropsho, S. Dwarkadas, and M. L. Scott, "Dynamic frequency and voltage scaling for a multiple-clock-domain microprocessor," *Micro, IEEE*, vol. 23, pp. 62-68, 2003.

[33] R. McGowen, C. A. Poirier, C. Bostak, J. Ignowski, M. Millican, W. H. Parks, *et al.*, "Power and temperature control on a 90-nm Itanium family processor," *Solid-State Circuits, IEEE Journal of*, vol. 41, pp. 229-237, 2006.

[34] R. Ge, X. Feng, W.-c. Feng, and K. W. Cameron, "CPU MISER: A Performance-Directed, Run-Time System for Power-Aware Clusters," in *International Conference on Parallel Processing, ICPP*, Xi'an, China, 2007, p. 18.

[35] B. Rountree, D. K. Lownenthal, B. R. d. Supinski, M. Schulz, V. W. Freeh, and T. Bletsch, "Adagio: making DVS practical for complex HPC applications," presented at the Proceedings of the 23rd international conference on Supercomputing, Yorktown Heights, NY, USA, 2009.

[36] V. Spiliopoulos, S. Kaxiras, and G. Keramidas, "Green governors: A framework for Continuously Adaptive DVFS," in *Green Computing Conference and Workshops (IGCC), 2011 International*, 2011, pp. 1-8.

[37] N. Kappiah, V. W. Freeh, and D. K. Lowenthal, "Just In Time Dynamic Voltage Scaling: Exploiting Inter-Node Slack to Save Energy in MPI Programs," in *Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference*, 2005, pp. 33-33.

[38] M. Curtis-Maury, A. Shah, F. Blagojevic, D. S. Nikolopoulos, B. R. d. Supinski, and M. Schulz, "Prediction models for multi-dimensional power-performance optimization on many cores," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, New York, NY,US, 2008, pp. 250-259.

[39] D. Li, B. d. Supinski, M. Schulz, K. Cameron, and D. Nikolopoulos, "Hybrid MPI/OpenMP power-aware computing," in *2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS)*, 2010, pp. 1-12.

[40] C. Su, D. Li, D. Nikolopoulos, M. Grove, K. Cameron, and B. d. Supinski, "Critical Path-Based Thread Placement for NUMA Systems," presented at the 2nd International Workshop on Modeling, Benchmarking and Simulation of High Performance Computing Systems, Seattle, Washington, 2011.

[41] R. Jejurikar and R. Gupta, "Energy aware task scheduling with task synchronization for embedded real time systems," presented at the Proceedings of the 2002 international conference on Compilers, architecture, and synthesis for embedded systems, Grenoble, France, 2002.

[42] C. Jian-Jia and K. Chin-Fu, "Energy-Efficient Scheduling for Real-Time Systems on Dynamic Voltage Scaling (DVS) Platforms," in *Embedded and Real-Time Computing Systems and Applications, 2007. RTCSA 2007. 13th IEEE International Conference on*, 2007, pp. 28-38.

[43] Q. Deng, D. Meisner, L. Ramos, T. F. Wenisch, and R. Bianchini, "MemScale: active low-power modes for main memory," presented at the Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems, Newport Beach, California, USA, 2011.

[44] B. Diniz, D. Guedes, J. Wagner Meira, and R. Bianchini, "Limiting the power consumption of main memory," presented at the Proceedings of the 34th annual international symposium on Computer architecture, San Diego, California, USA, 2007.

[45] X. Fan, C. Ellis, and A. Lebeck, "Memory controller policies for DRAM power management," presented at the Proceedings of the 2001 international symposium on Low power electronics and design, Huntington Beach, California, United States, 2001.

[46] A. R. Lebeck, X. Fan, H. Zeng, and C. Ellis, "Power aware page allocation," presented at the Proceedings of the ninth international conference on Architectural support for programming languages and operating systems, Cambridge, Massachusetts, United States, 2000.

[47] L. Xiaodong, L. Zhenmin, Z. Pin, Z. Yuanyuan, S. V. Adve, and S. Kumar, "Performance-directed energy management for storage systems," *Micro, IEEE*, vol. 24, pp. 38-49, 2004.

[48] Intel, "Intel and Core i7 (Nehalem) Dynamic Power Management," ed: Intel, 2011.

[49] H. Nagasaka, N. Maruyama, A. Nukada, T. Endo, and S. Matsuoka, "Statistical power modeling of GPU kernels using performance counters," in *Green Computing Conference, 2010 International* 2010, pp. 115-122.

[50] Z. Yao and J. D. Owens, "A quantitative performance analysis model for GPU architectures," in *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on*, 2011, pp. 382-393.

[51] S. Song and K. W. Cameron, "Iso-energy-efficiency: An approach to scalable system power-performance optimization," in *International Conference for High Performance Computing, Networking,Storage and Analysis (SC11), Ph.D. forum*, Seattle, Washington, 2011.

[52] R. Ge, X. Feng, S. Song, and K. W. Cameron, "PowerPack: Energy Profiling and Analysis of High-Performance Systems and Applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 99, pp. 658-671, 2009.

[53] S. Song, R. Ge, X. Feng, and K. W. Cameron, "Energy Profiling and Analysis of the HPC Challenge Benchmarks," *International Journal of High Performance Computing Applications*, vol. 23, pp. 265-276, 2009.

[54] *The NAS Parallel Benchmarks*. Available: http://www.nas.nasa.gov/Resources/Software

[55] P. R. Luszczek, D. H. Bailey, J. J. Dongarra, J. Kepner, R. F. Lucas, R. Rabenseifner, *et al.*, "The HPC Challenge (HPCC) benchmark suite," presented at the Proceedings of the 2006 ACM/IEEE conference on Supercomputing, Tampa, Florida, 2006.

[56] *TOP500 List: The LINPACK Benchmark*. Available: http://www.top500.org/project/linpack

[57] S. Song and K. Cameron, "System-level power-performance efficiency modeling for emergent GPU architectures," presented at the Proceedings of the 21st

international conference on Parallel architectures and compilation techniques, Minneapolis, Minnesota, USA, 2012.

[58] S. Song, C.-y. Su, B. Rountree, and K. W. Cameron, "A Simplified and Accurate Model of Power-Performance Efficiency on Emergent GPU Architectures," in *27th IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, Boston, MA, US, 2013.

[59] A. Vishnu, S. Song, A. Marquez, K. Barker, D. Kerbyson, K. Cameron, *et al.*, "Designing Energy Efficient Communication Runtime Systems for Data Centric Programming Models," in *Procedings of IEEE GreenCom 2010*, Hangzhou, China, 2010.

[60] A. Vishnu, S. Song, A. Marquez, K. Barker, D. Kerbyson, K. Cameron, *et al.*, "Designing energy efficient communication runtime systems: a view from PGAS models," *The Journal of Supercomputing*, pp. 1-19, OCT, 2011.

[61] J. Nieplocha, B. Palmer, V. Tipparaju, M. Krishnan, H. Trease, E. Apr\, *et al.*, "Advances, Applications and Performance of the Global Arrays Shared Memory Programming Toolkit," *Int. J. High Perform. Comput. Appl.*, vol. 20, pp. 203-231, 2006.

[62] F. Xie, M. Martonosi, and S. Malik, "Intraprogram dynamic voltage scaling: Bounding opportunities with analytic modeling," *ACM Trans. Archit. Code Optim.*, vol. 1, pp. 323-367, 2004.

[63] R. Springer, D. K. Lowenthal, B. Rountree, and V. W. Freeh, "Minimizing execution time in MPI programs on an energy-constrained, power-scalable cluster," presented at the Proceedings of the eleventh ACM SIGPLAN symposium on Principles and practice of parallel programming, New York, New York, USA, 2006.

[64] V. W. Freeh and D. K. Lowenthal, "Using multiple energy gears in MPI programs on a power-scalable cluster," presented at the Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming, Chicago, IL, USA, 2005.

[65] L. Min Yeol, W. F. Vincent, and K. L. David, "Adaptive, Transparent Frequency and Voltage Scaling of Communication Phases in MPI Programs," in *SC 2006 Conference, Proceedings of the ACM/IEEE*, 2006, pp. 14-14.

[66] H. David, C. Fallin, E. Gorbatov, U. R. Hanebutte, and O. Mutlu, "Memory power management via dynamic voltage/frequency scaling," presented at the Proceedings of the 8th ACM international conference on Autonomic computing, Karlsruhe, Germany, 2011.

[67] J. Li and J. F. Martinez, "Dynamic power-performance adaptation of parallel computation on chip multiprocessors," in *The Twelfth International Symposium on High-Performance Computer Architecture*, 2006, pp. 77-87.

[68] N. S. Kim, T. Austin, D. Baauw, T. Mudge, K. Flautner, J. S. Hu, *et al.*, "Leakage current: Moore's law meets static power," *Computer*, vol. 36, pp. 68-75, 2003.

[69] S. Hong and H. Kim, "An integrated GPU power and performance model," presented at the Proceedings of the 37th annual international symposium on Computer architecture, Saint-Malo, France, 2010.

[70] R. Bergamaschi, G. Han, A. Buyuktosunoglu, H. Patel, I. Nair, G. Dittmann, *et al.*, "Exploring power management in multi-core systems," presented at the

Proceedings of the 2008 Asia and South Pacific Design Automation Conference, Seoul, Korea, 2008.

[71] 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," presented at the Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture, 2006.

[72] A. Y. Grama, A. Gupta, and V. Kumar, "Isoefficiency: measuring the scalability of parallel algorithms and architectures," in *Multiprocessor performance measurement and evaluation*, 1995, pp. 103-112.

[73] M. J. Quinn, *Parallel Programming in C with MPI and OpenMP*: TATA McGraw-Hill 2003.

[74] R. Ge and K. W. Cameron, "Power-Aware Speedup," in *proceedings of the 21st IEEE International Parallel and Distributed Processing Symposium*, 2007, pp. 56-56.

[75] S. W. Son, M. Kandemir, and A. Choudhary, "Software-Directed Disk Power Management for Scientific Applications," in *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, 2005, pp. 4b-4b.

[76] L. A. Barroso and U. Holzle, "The Case for Energy-Proportional Computing," *Computer*, vol. 40, pp. 33-37, 2007.

[77] E. V. Carrera, E. Pinheiro, and R. Bianchini, "Conserving Disk Energy in Network Servers," in *Proceedings of the 17th International Conference on Supercomputing*, 2003.

[78] F. Douglis, P. Krishnan, and B. N. Bershad, "Adaptive disk spin-down policies for mobile computers," in *MLICS '95: Proceedings of the 2nd Symposium on Mobile and Location-Independent Computing*, Berkeley, CA, USA, 1995, pp. 121-137.

[79] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, and H. Franke, "DRPM: dynamic speed control for power management in server class disks," in *Computer Architecture, 2003. Proceedings. 30th Annual International Symposium on*, 2003, pp. 169-179.

[80] G. Wang, A. R. Butt, and C. Gniady, "On the impact of disk scrubbing on energy savings," presented at the Proceedings of the 2008 conference on Power aware computing and systems, San Diego, California, 2008.

[81] J. Zedlewski, S. Sobti, N. Garg, F. Zheng, A. Krishnamurthy, and R. Wang, "Modeling hard-disk power consumption," in *FAST '03: Proceedings of the 2nd USENIX Conference on File and Storage Technologies.*, Berkeley, CA, USA, 2003, pp. 217-230.

[82] Q. Zhu, Z. Chen, L. Tan, Y. Zhou, K. Keeton, and J. Wilkes, "Hibernator: Helping disk array sleep through the winter," in *Proceedings of the 20th ACM Symposium on Operating Systems Principles (SOSP'05)*, 2005.

[83] M. D. Hill and M. R. Marty, "Amdahl's Law in the Multicore Era," *Computer*, vol. 41, pp. 33-38, 2008.

[84] J. Paul and B. Meyer, "Amdahl's Law Revisited for Single Chip Systems," *International Journal of Parallel Programming*, vol. 35, pp. 101-123, 2007.

[85] R. Ge, X. Feng, and K. W. Cameron, "Performance-constrained Distributed DVS Scheduling for Scientific Applications on Power-aware Clusters," presented at the Proceedings of the 2005 ACM/IEEE conference on Supercomputing, 2005.

[86] V. W. Freeh, F. Pan, N. Kappiah, D. K. Lowenthal, and R. Springer, "Exploring the Energy-Time Tradeoff in MPI Programs on a Power-Scalable Cluster," presented at the Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Papers - Volume 01, 2005.

[87] I. Goiri, R. Beauchea, K. Le, T. D. Nguyen, M. E. Haque, J. Guitart, *et al.*, "GreenSlot: Scheduling energy consumption in green datacenters," in *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*, 2011, pp. 1-11.

[88] K. Le, R. Bianchini, J. Zhang, Y. Jaluria, J. Meng, and T. D. Nguyen, "Reducing electricity cost through virtual machine placement in high performance computing clouds," in *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*, 2011, pp. 1-12.

[89] R. Gonzalez and M. Horowitz, "Energy dissipation in General purpose microprocessors," in *IEEE International Symposium on Low Power Electronics*, Oct,1995.

[90] M. Valiev, E. J. Bylaska, N. Govind, K. Kowalski, T. P. Straatsma, H. J. J. v. Dam, *et al.*, "NWChem: a comprehensive and scalable open-source solution for large scale molecular simulations," *Comput. Phys. Commun.*, p. 181, 2010.

[91] X. Ma, M. Rincon, and Z. Deng, "Improving Energy Efficiency of GPU based General-Purpose Scientific Computing through Automated Selection of Near Optimal Configurations," University of Houston2011.

[92] S. Hong and H. Kim, "An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness," presented at the Proceedings of the 36th annual international symposium on Computer architecture, Austin, TX, USA, 2009.

[93] L. A. Barroso, "The Price of Performance," *Queue*, vol. 3, pp. 48-53, 2005.

[94] A. K. Mishra, S. Srikantaiah, M. Kandemir, and C. R. Das, "CPM in CMPs: Coordinated Power Management in Chip-Multiprocessors," presented at the Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, 2010.

[95] R. Bergamaschi, I. Nair, G. Dittmann, H. Patel, G. Janssen, N. Dhanwada, *et al.*, "Performance modeling for early analysis of multi-core systems," presented at the Proceedings of the 5th IEEE/ACM international conference on Hardware/software codesign and system synthesis, Salzburg, Austria, 2007.

[96] C. h. Hsu and W. c. Feng, "A power-aware run-time system for high-performance computing," in *Supercomputer'05*, 2005.

[97] H. Chung-Hsing, U. Kremer, and M. Hsiao, "Compiler-directed dynamic voltage/frequency scheduling for energy reduction in microprocessors," in *Low Power Electronics and Design, International Symposium on, 2001.*, 2001, pp. 275-278.

[98] B. Rountree, D. K. Lowenthal, M. Schulz, and B. R. de Supinski, "Practical performance prediction under Dynamic Voltage Frequency Scaling," in *Green Computing Conference and Workshops (IGCC), 2011 International*, 2011, pp. 1-8.

[99] B. Rountree, D. K. Lowenthal, S. Funk, V. W. Freeh, B. R. de Supinski, and M. Schulz, "Bounding energy consumption in large-scale MPI programs," in *Supercomputing, 2007. SC '07. Proceedings of the 2007 ACM/IEEE Conference on*, 2007, pp. 1-9.

[100] R. Balasubramonian, D. H. Albonesi, A. Buyuktosunoglu, and S. Dwarkadas, "Memory hierarchy reconfiguration for energy and performance in general purpose architectures," in *Proc. of the 33rd Annual Intl. Sym. on Microarchitecture*, 2000, pp. 245-257.

[101] A. S. Dhodapkar and J. E. Smith, "Managing multiconfiguration hardware via dynamic working set analysis," in *Proc. of the 29th Annual Intl. Sym. on Computer Architecture*, 2002, pp. 233-244.

[102] R. Preissl, T. Kockerbauer, M. Schulz, D. Kranzlmuller, B. Supinski, and D. J. Quinlan, "Detecting Patterns in MPI Communication Traces," in *Parallel Processing, 2008. ICPP '08. 37th International Conference on*, 2008, pp. 230-237.

[103] C.-H. Hsu and U. Kremer, "The design, implementation, and evaluation of a compiler algorithm for CPU energy reduction," in *PLDI '03: Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation, ACM*, New York, NY, USA, 2003, pp. 38-48.

[104] R. Bertran, M. Gonzalez, X. Martorell, N. Navarro, and E. Ayguade, "Decomposable and responsive power models for multicore processors using performance counters," presented at the Proceedings of the 24th ACM International Conference on Supercomputing, Tsukuba, Ibaraki, Japan, 2010.

[105] U. Hoelzle and L. A. Barroso, *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*: Morgan and Claypool Publishers 2009.

[106] C. Lefurgy, K. Rajamani, F. Rawson, W. Felter, M. Kistler, and T. W. Keller, "Energy management for commerical servers," *IEEE Computer*, pp. 39-48, 2003.

[107] K. Lim, J. Chang, T. Mudge, P. Ranganathan, S. K. Reinhardt, and T. F. Wenisch, "Disaggregated memory for expansion and sharing in blade servers," presented at the Proceedings of the 36th annual international symposium on Computer architecture, Austin, TX, USA, 2009.

[108] H. Huang, P. Pillai, and K. G. Shin, "Design and implementation of power-aware virtual memory," presented at the Proceedings of the annual conference on USENIX Annual Technical Conference, San Antonio, Texas, 2003.

[109] X. Li, Z. Li, F. David, P. Zhou, Y. Zhou, S. Adve, *et al.*, "Performance directed energy management for main memory and disks," presented at the Proceedings of the 11th international conference on Architectural support for programming languages and operating systems, Boston, MA, USA, 2004.

[110] N. Aggarwal, J. F. Cantin, M. H. Lipasti, and J. E. Smith, "Power-Efficient DRAM Speculation," in *High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on*, 2008, pp. 317-328.

[111] I. Hur and C. Lin, "A comprehensive approach to DRAM power management," in *High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on*, 2008, pp. 305-316.

[112] R. Crisp, "Direct RAMbus technology: the new main memory standard," *Micro, IEEE*, vol. 17, pp. 18-28, 1997.

[113] W. Felter, K. Rajamani, T. Keller, and C. Rusu, "A performance-conserving approach for reducing peak power consumption in server systems," presented at the Proceedings of the 19th annual international conference on Supercomputing, Cambridge, Massachusetts, 2005.
[114] H. Hanson and K. Rajamani, "What Computer Architects Need to Know About Memory Throttling," ed, 2010.
[115] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le, "RAPL: Memory power estimation and capping," in *Low-Power Electronics and Design (ISLPED), 2010 ACM/IEEE International Symposium on*, 2010, pp. 189-194.
[116] J. Lin, H. Zheng, Z. Zhu, H. David, and Z. Zhang, "Thermal modeling and management of DRAM memory systems," presented at the Proceedings of the 34th annual international symposium on Computer architecture, San Diego, California, USA, 2007.
[117] J. Lin, H. Zheng, Z. Zhu, E. Gorbatov, H. David, and Z. Zhang, "Software thermal management of dram memory for multicore systems," presented at the Proceedings of the 2008 ACM SIGMETRICS international conference on Measurement and modeling of computer systems, Annapolis, MD, USA, 2008.
[118] D. Meisner, B. T. Gold, and T. F. Wenisch, "PowerNap: eliminating server idle power," presented at the Proceedings of the 14th international conference on Architectural support for programming languages and operating systems, Washington, DC, USA, 2009.
[119] E. Elnozahy, M. Kistler, and R. Rajamony, "Energy-Efficient Server Clusters
Power-Aware Computer Systems." vol. 2325, B. Falsafi and T. Vijaykumar, Eds., ed: Springer Berlin / Heidelberg, 2003, pp. 179-197.
[120] A. N. Udipi, N. Muralimanohar, N. Chatterjee, R. Balasubramonian, A. Davis, and N. P. Jouppi, "Rethinking DRAM design and organization for energy-constrained multi-cores," presented at the Proceedings of the 37th annual international symposium on Computer architecture, Saint-Malo, France, 2010.
[121] J. H. Ahn, N. P. Jouppi, C. Kozyrakis, J. Leverich, and R. S. Schreiber, "Future scaling of processor-memory interfaces," presented at the Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, Portland, Oregon, 2009.
[122] Z. Hongzhong, L. Jiang, Z. Zhao, E. Gorbatov, H. David, and Z. Zhichun, "Mini-rank: Adaptive DRAM architecture for improving memory power efficiency," in *Microarchitecture, 2008. MICRO-41. 2008 41st IEEE/ACM International Symposium on*, 2008, pp. 210-221.
[123] F. A. Ware and C. Hampel, "Improving Power and Data Efficiency with Threaded Memory Modules," in *Computer Design, 2006. ICCD 2006. International Conference on*, 2006, pp. 417-424.
[124] K. Puttaswamy, C. Kyu-Won, P. Jun Cheol, V. J. Mooney, III, A. Chatterjee, and P. Ellervee, "System level power-performance trade-offs in embedded systems using voltage and frequency scaling of off-chip buses and memory," in *System Synthesis, 2002. 15th International Symposium on*, 2002, pp. 225-230.
[125] H. Zheng, J. Lin, Z. Zhang, and Z. Zhu, "Decoupled DIMM: building high-bandwidth memory system using low-speed DRAM devices," presented at the

Proceedings of the 36th annual international symposium on Computer architecture, Austin, TX, USA, 2009.

[126] V. Delaluz, M. Kandemir, N. Vijaykrishnan, A. Sivasubramaniam, and M. J. Irwin, "DRAM Energy Management Using Sof ware and Hardware Directed Power Mode Control," presented at the Proceedings of the 7th International Symposium on High-Performance Computer Architecture, 2001.

[127] P. Vivek, W. Jiang, Y. Zhou, and R. Bianchini, "DMA-aware memory energy management," in *High-Performance Computer Architecture, 2006. The Twelfth International Symposium on*, 2006, pp. 133-144.

[128] A. Vahdat, A. Lebeck, and C. S. Ellis, "Every joule is precious: the case for revisiting operating system design for energy efficiency," presented at the Proceedings of the 9th workshop on ACM SIGOPS European workshop: beyond the PC: new challenges for the operating system, Kolding, Denmark, 2000.

[129] H. Huang, K. G. Shin, C. Lefurgy, and T. Keller, "Improving energy efficiency by making DRAM less randomly accessed," presented at the Proceedings of the 2005 international symposium on Low power electronics and design, San Diego, CA, USA, 2005.

[130] P. Zhou, V. Pandey, J. Sundaresan, A. Raghuraman, Y. Zhou, and S. Kumar, "Dynamic tracking of page miss ratio curve for memory management," presented at the Proceedings of the 11th international conference on Architectural support for programming languages and operating systems, Boston, MA, USA, 2004.

[131] M. E. Tolentino, J. Turner, and K. W. Cameron, "Memory-miser: a performance-constrained runtime system for power-scalable clusters," presented at the Proceedings of the 4th international conference on Computing frontiers, Ischia, Italy, 2007.

[132] M. E. Tolentino, J. Turner, and K. W. Cameron, "Memory MISER: Improving Main Memory Energy Efficiency in Servers," *Computers, IEEE Transactions on*, vol. 58, pp. 336-350, 2009.

[133] Q. Zhu and Y. Zhou, "Power Aware Storage Cache Management," *IEEE Transactions on Computers (IEEE-TC),*, vol. 54, pp. 587-602, 2005.

[134] C. Gniady, Y. C. Hu, and Y.-H. Lu, "Program counter based techniques for dynamic power management," in *HPCA'04*, 2004, pp. 24-35.

[135] C. S. Ellis, "The case for higher-level power management," in *Hot Topics in Operating Systems, 1999. Proceedings of the Seventh Workshop on*, 1999, pp. 162-167.

[136] C.-H. Hwang and A. C.-H. Wu, "A predictive system shutdown method for energy saving of event-driven computation," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 5, pp. 226-241, 2000.

[137] A. Weissel, Bj, #246, r. Beutel, and F. Bellosa, "Cooperative I/O: a novel I/O semantics for energy-aware applications," *SIGOPS Oper. Syst. Rev.*, vol. 36, pp. 117-129, 2002.

[138] C. Weddle, M. Oldham, J. Qian, and A.-I. A. Wang, "PARAID: A Gear-Shifting Power-Aware RAID," in *FAST'2007*, 2007.

[139] L. Yung-Hsiang, L. Benini, and G. De Micheli, "Operating-system directed power reduction," in *Low Power Electronics and Design, 2000. ISLPED '00. Proceedings of the 2000 International Symposium on*, 2000, pp. 37-42.

[140] L. Yung-Hsiang, L. Benini, and G. De Micheli, "Power-aware operating systems for interactive systems," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 10, pp. 119-134, 2002.

[141] R. Zamani, A. Afsahi, Y. Qian, and C. Hamacher, "A feasibility analysis of power-awareness and energy minimization in modern interconnects for high-performance computing," in *CLUSTER '07: Proceedings of the 2007 IEEE International Conference on Cluster Computing*, Washington, DC, USA, 2007, pp. 118-128.

[142] S. S. K. Kandalla, E. P. Mancini, and D. K. Panda, "Designing Power-Aware Collective Communication Algorithms for InfiniBand Clusters," presented at the Technical Report, The Ohio State University, 2010.

[143] J. Liu, D. Poff, and B. Abali, "Evaluating high performance communication: a power perspective," in *ICS '09: Proceedings of the 23rd international conference on Supercomputing*, New York, NY, USA, 2009, pp. 326-337.

[144] X. Ma, M. Dong, L. Zhong, and Z. Deng, "Statistical Power Consumption Analysis and Modeling for GPU-based Computing," in *SOSP Workshop on Power Aware Computing and Systems (HotPower '09)*, Big Sky, MT, 2009.

[145] "Report to Congress on Server and Data Center Energy Efficiency," US Environmental Protection Agency2007.

[146] *Power Scorecard. Electricity from Coal*. Available: http://www.powerscorecard.org/tech_detail.cfm?resource_id=2

[147] N. Tolia, Z. Wang, M. Marwah, C. Bash, P. Ranganathan, and X. Zhu, "Delivering energy proportionality with non energy-proportional systems: optimizing the ensemble," presented at the Proceedings of the 2008 conference on Power aware computing and systems, San Diego, California, 2008.

[148] M. E. Femal and V. W. Freeh, "Boosting data center performance through non-uniform power allocation," in *International Conference on Autonomic Computing*, 2005.

[149] Q. Zhu, J. Zhu, and G. Agrawal, "Power-Aware Consolidation of Scientific Workflows in Virtualized Environments," presented at the Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, 2010.

[150] N. Vijaykrishnan, M. Kandemir, M. Irwin, H. Kim, and W. Ye, "Energy-driven integrated hardware-software optimizations using SimplePower," in *Proceedings of 27th International Symposium on Computer Architecture*, Vancouver, British Columbia, 2000.

[151] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, "The design and use of simplepower: A cycle-accurate energy estimation tool," pp. 340-345, 2000.

[152] D. C. Burger and T. M. Austin, "The SimpleScalar Toolset, Version 2.0," *Computer Architecture News*, vol. 25, pp. 13-25, 1997.

[153] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: a framework for architectural-level power analysis and optimizations," in *ISCA '00: Proceedings of the 27th annual international symposium on Computer architecture*, New York, NY, USA, 2000, pp. 83-94.

[154] D. Brooks, P. Bose, V. Srinivasan, and M. K. Gschwind, "New methodology for early-stage, microarchitecture-level power-performance analysis of microprocessors," *IBM Journal of Research and Development*, 2003.

[155] Y. Ding, K. Malkowski, P. Raghavan, and M. Kandemir, "Towards Energy Efficient Scaling of Scientific Codes," in *IEEE International Symposium on Parallel and Distributed Processing*, 2008, pp. 1 - 8

[156] G. Cai and C. Lim, "Architectural Level Power/Performance Optimization and Dynamic Power Optimization," in *Proceedings of Cool Chips Tutorial at 32nd ISCA*, 1999.

[157] A. Dhodapkar, C. H. Lim, G. Cai, and W. R. Daasch, ""TEM2P2EST: A Thermal Enabled Multimodel Power/Performance Estimator," in *Proceedings of the First International Workshop on Power-Aware Computer Systems*, 2000.

[158] H.-S. W. Xinping, H. s. Wang, X. Zhu, L. s. Peh, and S. Malik, "Orion: A power-performance simulator for interconnection networks," in *the 35th International Symposium on Microarchitecture (MICRO)*, Istanbul, Turkey, 2002.

[159] S. Gurumurthi, A. Sivasubramaniam, M. J. Irwin, N. Vijaykrishnan, and M. Kandemir, "Using Complete Machine Simulation for Software Power Estimation: The SoftWatt Approach," in *Proceedings of Eighth International Symposium on High-Performance Computer Architecture (HPCA'02)*, Boston, Massachusettes, 2002.

[160] J. Flinn and M. Satyanarayanan, "PowerScope: A Tool for Profiling the Energy Usage of Mobile Applications," in *Proceedings of the Second IEEE Workshop on Mobile Computer Systems and Applications*, 1999.

[161] *NWICE system at Pacific Northwest National Lab*. Available: http://www.er.doe.gov/ascr/News/NewsRoundup8-08.html

[162] C. Isci and M. Martonosi, "Runtime Power Monitoring in High-End Processors: Methodology and Empirical Data," in *the 36th annual IEEE/ACM International Symposium on Microarchitecture*, 2003.

[163] V. Tiwari, D. Singh, S. Rajgopal, G.Mehta, R.Patel, and F.Baez, "Reducing Power in High-Performance Microprocessors," in *the 35th Conference on Design Automation*, 1998.

[164] R. Joseph and M. Martonosi, "Run-time power estimation in high performance microprecessors," *International symposium on low power electronics and design*, 2001.

[165] F. Bellosa, "The Benefits of Event-Driven Energy Accounting in Power-Sensitive Systems," in *9th ACM SIGOPS European Workshop*, Kolding, Denmark, 2000.

[166] D. Bedard, R. Fowler, M. Y. Lim, and A. Porterfield, "PowerMon 2: Fine-grained, Integrated Power Measurement," RENCI Technical Report TR-09-042009.

[167] A. Lewis, S. Ghosh, and N.-F. Tzeng, "Run-time energy consumption estimation based on workload in server systems," in *Proceeding of HotPower'08 Proceedings of the 2008 conference on Power aware computing and systems* Berkeley, CA, USA 2008.

[168] A. Kerr, G. Diamos, and S. Yalamanchili, "Modeling GPU-CPU workloads and systems," presented at the Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units, Pittsburgh, Pennsylvania, 2010.

[169] S. S. Baghsorkhi, M. Delahaye, S. J. Patel, W. D. Gropp, and W.-m. W. Hwu, "An adaptive performance modeling tool for GPU architectures," presented at the Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, Bangalore, India, 2010.

[170] K. W. Cameron and X.-H. Sun, "Quantifying Locality Effect in Data Access Delay: Memory logP," in *Proceedings of IEEE International Parallel and Distributed Processing Symposium (IPDPS 2003)*, Nice, France, 2003.

[171] B. M. Maggs, L. R. Matheson, and R. E. Tarjan, "Models of Parallel Computation: A Survey ad Synthesis," in *Proceedings of 28th Hawaii International Conference on System Sciences (HICSS)*, Honolulu, HI, 1995.

[172] D. E. Culler, R. Karp, D. A. Patterson, A. Sahay, E. Santos, K.Schauser, *et al.*, "LogP: A Practical Model of Parallel Computation," *Communications of the ACM*, vol. 39, pp. 78-85, 1996.

[173] S. Fortune and Wyllie, "Parallelism in random access machines," in *Proceedings of 10th Annual ACM Symposium on Theory of Computing*, San Diego, CA, 1978.

[174] L. G. Valiant, "A Bridging Model for Parallel Computation," *Communications of the ACM*, vol. 33, pp. 103-111, 1990.

[175] A. Alexandrov, M. F. Ionescu, K. Schauser, and C. Scheiman, "LogGP: Incorporating Long Messages into the LogP model," in *Proceedings of Seventh Annual Symposium on Parallel Algorithms and Architecture*, Santa Barbara, CA, 1995, pp. 95-105.

[176] M. I. Frank, A. Agarwal, and M. K. Vernon, "LoPC: Modeling Contention in Parallel Algorithms," in *Proceedings of Sixth Symposium on Principles and Practice of Parallel Programming*, Las Vegas, NV, 1997, pp. 276-287.

[177] F. Ino, N. Fujimoto, and K. Hagihara, "LogGPS: A Parallel Computational Model for Synchronization Analysis," in *Proceedings of PPoPP'01*, Snowbird, Utah, 2001, pp. 133-42.

[178] C. A. Moritz and M. I. Frank, "LoGPC: Modeling Network Contention in Message-Passing Programs," in *Proceedings of SIGMETRICS'98*, Madison, WI, 1998, pp. 254-63.

[179] T. Kielmann and H. E. Bal, "Fast Measurement of LogP Parameters for Message Passing Platforms.," in *Proceedings of the 15 IPDPS 2000 Workshops on Parallel and Distributed Processing*, 2000.

[180] K. W. Cameron and R. Ge, "Predicting and Evaluating Distributed Communication Performance," in *Proceedings of the 2004 ACM/IEEE Supercomputing Conference*, 2004.

[181] K. W. Cameron, R. Ge, and X.-H. Sun., "lognP and log3P: Accurate Analytical Models of Point-to-Point Communication in Distributed Systems," *IEEE TRANSACTIONS ON COMPUTERS*, vol. 56, 2007.

[182] B. Tu, J. Fan, J. Zhan, and X. Zhao, "Accurate Analytical Models for Message Passing on Multi-core Clusters," in *Proceedings of the 2009 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing* 2009.

[183] N. Jiang, J. Pisharath, and A. Choudhary, "Characterizing and improving energy-delay tradeoffs in heterogeneous communication systems," *Signals, Circuits and Systems, 2003. SCS 2003. International Symposium* vol. 2, pp. 409-412, 2003.

[184] K. W. Cameron, R. Ge, and X. Feng, "High-Performance, Power-Aware Distributed Computing for Scientific Applications," *Computer,* vol. 38, pp. 40-47, 2005.
[185] X. Feng, R. Ge, and K. W. Cameron, "Power and Energy Profiling of Scientific Applications on Distributed Systems," presented at the Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Papers - Volume 01, 2005.
[186] J. Pjeivac-Grbovi, T. Angskun, G. Bosilca, G. E. Fagg, E. Gabriel, and J. J. Dongarra, "Performance analysis of MPI collective operations," *Cluster Computing,* vol. 10, pp. 127-143, 2007.
[187] M. M. Qiang Wu, Douglas W. Clark, V. J. Reddi, Dan Connors, Youfeng Wu, Jin Lee, David Brooks, "A Dynamic Compilation Framework for Controlling Microprocessor Energy and Performance," presented at the Proceedings of the 38th annual IEEE/ACM International Symposium on Microarchitecture, Barcelona, Spain, 2005.
[188] K.C.Louden, *Compiler Construction: Principles and Practice, 1st edition,* 1997.
[189] T. A. Jelena Pjeivac-Grbovi, George Bosilca, Graham E. Fagg, Edgar Gabriel, Jack J. Dongarra, "Performance analysis of MPI collective operations," *Cluster Computing,* vol. 10, pp. 127-143, 2007.
[190] (2010). *MPICH2*. Available: http://phase.hpcc.jp/mirrors/mpi/mpich2/
[191] (2010). *TAU: Tuning and Analysis Utilities*. Available: http://www.cs.uoregon.edu/research/tau/home.php
[192] L. McVoy. *LMbench - Tools for Performance Analysis*. Available: http://www.bitmover.com/lmbench/
[193] (2010). *InfiniBand Trade Association*. Available: http://www.infinibandta.org/
[194] S. S. Shende and A. D. Malony, "The TAU Parallel Performance System," *International Journal of High Performance Computing Applications,* vol. 20, pp. 287-311, 2006.
[195] B. Subramaniam and W.-c. Feng, "Load-Varying LINPACK: A Benchmark for Evaluating Energy Efficiency in High-End Computing," in *GreenCom 2010*, Hangzhou,China, 2010.
[196] NVIDIA. (NOV 2011). *Tesla c2075 device.* Available: http://www.nvidia.com/object/workstation-solutions-tesla.html
[197] NVIDIA. (2011, Fermi official whitepaper. Available: http://developer.nvidia.com/cuda-toolkit-41
[198] J. S. Vetter, R. Glassbrook, J. Dongarra, K. Schwan, B. Loftis, S. McNally, *et al.*, "Keeneland: Bringing Heterogeneous GPU Computing to the Computational Science Community," *Computing in Science & Engineering,* vol. 13, pp. 90-95, 2011.
[199] NVIDIA, "CUDA SDK 4.1."
[200] J. C. Gordon, A. T. Fenley, and A. Onufriev, "An analytical approach to computing biomolecular electrostatic potential. ii. validation and applications," *The Journal of Chemical Physics,* vol. 129, 2008.
[201] A. Danalis, G. Marin, C. McCurdy, J. S. Meredith, P. C. Roth, K. Spafford, *et al.*, "The Scalable Heterogeneous Computing (SHOC) benchmark suite," presented at

the Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units, Pittsburgh, Pennsylvania, 2010.

[202] S. Huang, S. Xiao, and W. Feng, "On the energy efficiency of graphics processing units for scientific computing," in *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, 2009, pp. 1-8.

[203] S. Collange, D. Defour, and A. Tisserand, "Power Consumption of GPUs from a Software Perspective," presented at the Proceedings of the 9th International Conference on Computational Science: Part I, Baton Rouge, LA, 2009.

[204] K.Ramani, A.Ibrahim, and d.sHIMIZU, "PowerRed: A flexible simulation framework for power efficiency exploration in gpus," University of Utah2007.

[205] J. W. Sheaffer, D. Luebke, and K. Skadron, "A flexible simulation framework for graphics architectures," presented at the Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware, Grenoble, France, 2004.

[206] A. Kerr, G. Diamos, and S. Yalamanchili, "A characterization and analysis of PTX kernels," in *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, 2009, pp. 3-12.

[207] S. Ryoo, C. I. Rodrigues, S. S. Stone, S. S. Baghsorkhi, S.-Z. Ueng, J. A. Stratton, *et al.*, "Program optimization space pruning for a multithreaded gpu," presented at the Proceedings of the 6th annual IEEE/ACM international symposium on Code generation and optimization, Boston, MA, USA, 2008.

[208] J. Meng and K. Skadron, "Performance modeling and automatic ghost zone optimization for iterative stencil loops on GPUs," presented at the Proceedings of the 23rd international conference on Supercomputing, Yorktown Heights, NY, USA, 2009.

[209] L. Yixun, E. Z. Zhang, and S. Xipeng, "A cross-input adaptive framework for GPU program optimizations," in *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, 2009, pp. 1-10.

[210] S. Collange, M. Daumas, D. Defour, and D. Parello, "Barra: A Parallel Functional Simulator for GPGPU," in *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2010 IEEE International Symposium on*, 2010, pp. 351-360.

[211] J. Sim, A. Dasgupta, H. Kim, and R. Vuduc, "A performance analysis framework for identifying potential benefits in GPGPU applications," presented at the Proceedings of the 17th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming, New Orleans, Louisiana, USA, 2012.

[212] NVIDIA. (2012). *CUDA Programming Environment*. Available: http://www.nvidia.com/object/cuda home new.html

[213] NVIDIA. *Fermi Occupancy Calculator*. Available: http://developer.nvidia.com/cuda-toolkit-41

[214] S. J. Russel and P.Norvig, *Artificial Intelligence: A Modern Approach*: Pearson Education, 2003.

[215] NVIDIA. (2011, NVIDIA management library. Available: http://developer.nvidia.com/nvidia-management-library-nvml

[216] NVIDIA. (2011, CUPTI user's guide. Available: http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUPTI _Users_Guide.pdf
[217] K. Asanovic, R. Bodik, J. Demmel, T. Keaveny, K. Keutzer, J. Kubiatowicz, *et al.*, "A view of the parallel computing landscape," *Commun. ACM,* vol. 52, pp. 56-67, 2009.
[218] S. lab. (2013). *SCAPE Software*. Available: http://scape.cs.vt.edu/software/
[219] A. M. Aji, M. Daga, and W.-c. Feng, "Bounding the effect of partition camping in GPU kernels," presented at the Proceedings of the 8th ACM International Conference on Computing Frontiers, Ischia, Italy, 2011.
[220] W.Huber, A.V.Heydebreck, and M.Vingron, *Analysis of microarray gene expression data, Handbook of Statistical Genetics*, 2nd ed., 2003.
[221] S. Kumar, G. Dozsa, G. Almasi, P. Heidelberger, D. Chen, M. E. Giampapa, *et al.*, "The deep computing messaging framework: generalized scalable message passing on the blue gene/P supercomputer," presented at the Proceedings of the 22nd annual international conference on Supercomputing, Island of Kos, Greece, 2008.
[222] F. Petrini, W.-c. Feng, A. Hoisie, S. Coll, and E. Frachtenberg, "The Quadrics Network (QsNet): High-Performance Clustering Technology," presented at the Proceedings of the The Ninth Symposium on High Performance Interconnects, 2001.
[223] J. Nieplocha and B. Carpenter, *ARMCI: A Portable Remote Memory Copy Library for Distributed Array Libraries and Compiler Run-Time Systems*: Springer-Verlag, 1999.
[224] P. Husbands, C. Iancu, and K. Yelick, "A performance analysis of the Berkeley UPC compiler," presented at the Proceedings of the 17th annual international conference on Supercomputing, San Francisco, CA, USA, 2003.
[225] G. Shah and C. Bender, "Performance and Experience with LAPI -- A New High-Performance Communication Library for the IBM RS/6000 SP," presented at the Proceedings of the 12th. International Parallel Processing Symposium on International Parallel Processing Symposium, 1998.
[226] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, *et al.*, "Myrinet: A Gigabit-per-Second Local Area Network," *IEEE Micro,* vol. 15, pp. 29-36, 1995.
[227] V. Velusamy and C. Rao, "Programming the Infiniband Network Architecture for High Performance Message Passing Systems," in *ISCA*, 2003.