Energy and Performance Models Enabling Design Space Exploration using Domain Specific Languages

Mariam Umar

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
Changhee Jung
Calvin J. Ribbens
Eli Tilevich
Jeffrey S. Vetter

April 9, 2018
Blacksburg, Virginia

Keywords: Performance Modeling, Energy Modeling and Estimation, High Performance Computing, Exascale Architectures. Domain Specific Languages

Energy and Performance Models Enabling Design Space Exploration using Domain Specific Languages

Mariam Umar

ABSTRACT

With the advent of exascale architectures maximizing performance while maintaining energy consumption within reasonable limits has become one of the most critical design constraints. This constraint is particularly significant in light of the power budget of 20 MWatts set by the U.S. Department of Energy for exascale supercomputing facilities. Therefore, understanding an application's characteristics, execution pattern, energy footprint, and the interactions of such aspects is critical to improving the application's performance as well as its utilization of the underlying resources.

With conventional methods of analyzing performance and energy consumption trends scientists are forced to limit themselves to a manageable number of design parameters. While these modeling techniques have catered to the needs of current high-performance computing systems, the complexity and scale of exascale systems demands that large-scale design-space-exploration techniques are developed to enable comprehensive analysis and evaluations.

In this dissertation we present research on performance and energy modeling of current high performance computing and future exascale systems. Our thesis is focused on the design space exploration of current and future architectures, in terms of their reconfigurability, application's sensitivity to hardware characteristics (e.g., system clock, memory bandwidth), application's execution patterns, application's communication behavior, and utilization of resources. Our research is aimed at understanding the methods by which we may maximize performance of exascale systems, minimize energy consumption, and understand the trade offs between the two.

We use analytical, statistical, and machine-learning approaches to develop accurate, portable and scalable performance and energy models. We develop application and machine abstractions using Aspen (a domain specific language) to implement and evaluate our modeling techniques. As part of our research we develop and evaluate system-level performance and energy-consumption models that form part of an automated modeling framework, which analyzes application signatures to evaluate sensitivity of reconfigurable hardware components for candidate exascale proxy applications. We also develop statistical and machine-learning based models of the application's execution patterns on heterogeneous platforms. We also propose a communication and computation modeling and mapping framework for exascale proxy architectures and evaluate the framework for an exascale proxy application. These models serve as external and internal extensions to Aspen, which enable proxy exascale architecture implementations and thus facilitate design space exploration of exascale systems.

This work is supported in part by National Science Foundation (Grant # 1422712), Virginia Tech and Oak Ridge National Laboratory.

Energy and Performance Models Enabling Design Space Exploration using Domain Specific Languages

Mariam Umar

GENERAL AUDIENCE ABSTRACT

Performance monitoring and modeling has been an extensively researched topic over the last decade. The traditional approaches of manually modeling performance and energy worked well for previous generation computers. With the prevalence of complex high-performance computers, clusters and the anticipation of future exascale architectures, the conventional modeling approaches will not be sufficient. A number of reasons limit the conventional modeling approaches, e.g, complexity of current and future architectures, increase in number of performance parameters to monitor, diversity in the architecture etc. This issue will worsen with the advent of exascale architectures that encompasses complex micro-architectures along with the increases in scale that have never been encountered in the computing industry before.

In this dissertation, we focus on two primary aspects of performance and energy modeling in the context of current high performance computing and future exascale architectures. We focus on adapting conventional modeling approaches to comprise the properties of accuracy, scalability, portability and independence of architectures. Centered around performance and energy improvements, we also develop design space exploration techniques that study the effects of application performance improvement in terms of reconfigurable hardware. We also quantitatively measure the effects of application performance sensitivity with changing hardware configurations – using analytical and machine learning modeling techniques. We explore theoretical exascale architecture, and validate it for performance limits. We develop a communication and computation model for the proxy exascale architecture and test it for strong and weak scaling for co-design for molecular dynamics.

This work is supported in part by National Science Foundation (Grant # 1422712), Virginia Tech and Oak Ridge National Laboratory.

To

my parents Shafqat & Shamim,

my husband Umar,

my brother Hassan

and my son Ahmed

for their love, support, and encouragement.

ACKNOWLEDGEMENTS

I am thankful to Allah for giving me the ability to foster in my graduate studies and persevere in the face of challenges. I pray that I use my knowledge and experiences for the benefit of all.

There are a number of people who have played a vital role in enabling my success. I am grateful to all for their support.

I will forever be grateful to my family, who have unconditionally supported me through all the highs and lows of my life. My parents have been my most enthusiastic supporters. Their prayers, encouragement, and advice has always enabled me in achieving my ambitions. They nurtured my belief that I can achieve anything with focus and diligence. It is their sheer innocence and simplicity that taught me the importance of staying humble while being tenacious as I work towards my goals. I would always find comfort in my regular conversations with them. My husband, Umar, has also been an integral source of support and encouragement. He has celebrated my successes and held me in times of failure. He always reassured me that I have the right attitude and ability to achieve my ambitions. He has played the roles of a wonderful husband, best friend, professional mentor, brainstorming buddy, confidante and the list goes on and on. I will never be able to summarize my appreciation for his affection and genuine love. My 4-year-old son, Ahmed, has been my greatest source of joy. His lively and joyful nature along with his generous love has always helped me overcome the stresses of graduate studies. In an attempt to be a role model for him, I have learned to stay strong and maintain a positive attitude in the face of hardships. He is one of the few reasons that has allowed me to further develop my ability to persevere. His inquisitive nature and interesting point of views have always been a tremendous joy and a source of inspiration. My brother's continued support and motivation has always motivated me to pursue my dreams. His words of encouragement have always been a beacon of light for me. My mother in-law and father in-law have always prayed for my success and I will always be thankful for their support.

Dr. Kirk Cameron, my thesis advisor, has been my most vocal champion. His sincere advice, encouragement, and genuine desire to see me succeed has been a tremendous source of motivation for me. I will forever be grateful to him for providing me the opportunity to work at the SCAPE laboratory, where I was able to develop my professional skills and conduct meaningful as well as state of the art research, all the while not having to worry about research funding and other tribulations. He always encouraged me to maintain a positive attitude in the face of adversity. He showed all the students, by setting an example, how to conduct oneself, stay upbeat, and create a respectful, healthy, and professional work environment. In addition to developing technical skills under his supervision, I learned many soft-skills by observing his professional and encouraging attitude, where he would never compromise on the quality of research. I have flourished tremendously under his supervision and I would not have grown in a similar manner had I been working with any other research group.

I am also thankful to Dr. Calvin Ribbens who has played a significant role towards my success. I am grateful to him for his sincere guidance and support. When I was stranded at a point where I could see no way out, his support and encouragement proved to be the turning point, and rest is history. I will forever be grateful to him.

Dr. Eli Tilevich is perhaps the most accomplished and well-rounded professor that I have met during my tenure at Virginia Tech. I have always felt that he genuinely celebrates students' successes. He has always been approachable and has never shied away from helping students as they face different trials during their studies. His courses have always been meaningful. He is the manifestation of what a mentor should be.

I am grateful to Dr. Jeffrey Vetter, who has played a phenomenal role towards my success and enabling me to conduct research. I am thankful to him for providing me the opportunity to work alongside

accomplished scholars at the Oak Ridge National Laboratory (ORNL). In spite of his busy schedule, he has always been available and accessible to his students. Throughout my research, Dr. Vetter provided me with quality feedback and guidance that enabled me in conducting meaningful research. I am also grateful to Jeremy Meredith who guided me during my early days at ORNL. His patience and encouragement enabled me to understand the intricacies of Aspen (software that I used extensively as part of my research). I remember the days when in spite of his schedule, he would make time for our regular meetings. Dr. Shirley Moore at ORNL has also played a significant role in honing my research, particularly during the last two years of my research. With her support, I was able to obtain access to various compute resources that furthered my research.

I have also been blessed to have wonderful people as friends. Dr. Uzma, my sister in law has always been a source of inspiration for me. Faria, my sister in law, has encouraged me to keep going in spite of all the trials and tribulations. Speaking with my dear friend, Dr. Ayat, has always been a source of relief and energy. My interactions with Ashima also allowed me to stay upbeat throughout my doctoral studies. I am grateful to all the current members and alumni of the SCAPE lab, who were always available to help and a source of inspiration. I am also thankful to my little friends, my niece and nephew Noor and Abdullah, for their loving friendship.

I am indebted to the Department of Computer Science at Virginia Tech, for providing me with the opportunity to pursue my doctoral studies and providing me with all the necessary resources.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Performance and energy consumption are considered to be among the primary design parameters for high performance computing systems. These considerations are expected to continue as we move towards the exasale computing era, which will entail increased complexity of architectures and large-scale systems.

It is a well known fact that the number of transistors in microprocessors have continued to grow over the last four decades (Figure 1.1). It is also well established that the approach of obtaining performance gains by increasing system clock frequency is a thing of the past and that significant performance gains in the future are to be expected from increasing levels of parallelism (Figure 1.2). With the increase in number of cores in high-performance-computing (HPC) systems, to achieve increased parallelism, there is a substantial increase in system-wide power consumption. This increase in power consumption is to the extent that supercomputers consume about the same power as cities[1]. It is for such reasons that the United States Department of Energy's (DOE) design criteria for exascale systems includes a power limit of

[1] In 2015, Tianhe-2, the 3.12 million processor supercomputer would consume the same power, of 17.8 megawatts, as the city of Tupelo, Mississippi, with a population of about 36,000 people [71].

20 megawatts, with a theoretical peak performance of at least one exaflops [44]. It is apparent that the competing goals of maximizing performance and minimizing energy consumption make the challenge of designing exascale systems much more difficult. Thus, it is not only important to understand the performance and energy consumption patterns of exascale systems but also to acknowledge the trade offs between the two.

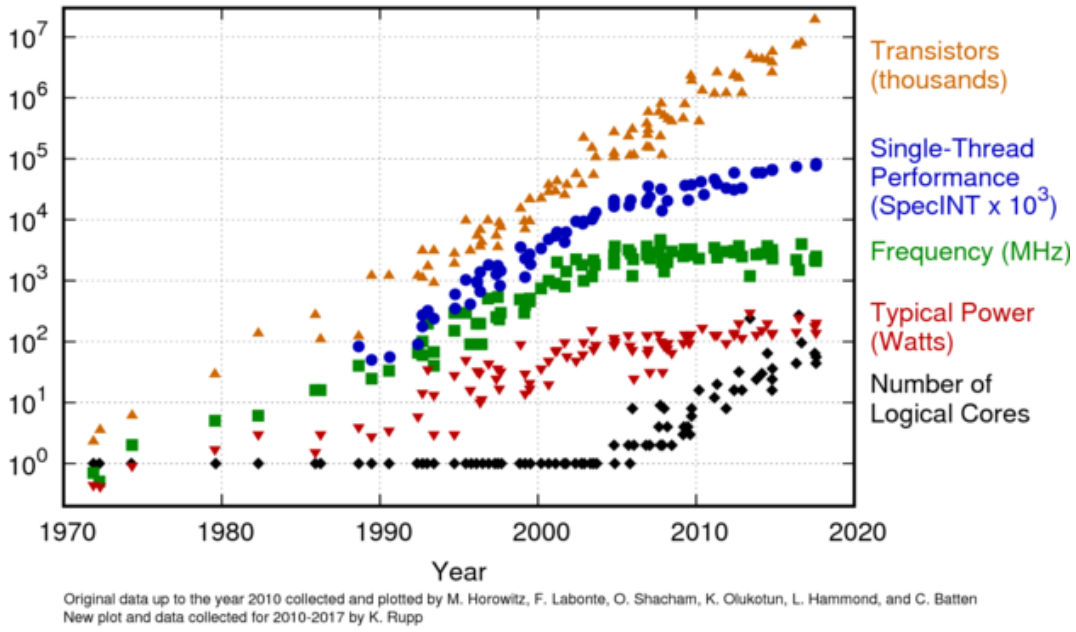


Figure 1.1: 42 years of microprocessor trend data (source: [140])

1.1 Research Challenges

With the conventional methods of analyzing performance and energy consumption trends, scientists have typically limited themselves to a manageable number of design parameters [44]. The general trend to improve performance of an application is to apply application-specific software optimizations, hardware-specific optimization or a select combination of them. However, with future exascale architectures, heterogeneity is a common theme; heterogeneity alone provides several opportunities to explore optimizations. Thus, the drastic increase in available

design parameters, complexity, and scale of exascale systems demands that large-scale design-space-exploration techniques are developed to enable comprehensive analysis of performance, energy consumption, and their trade offs.

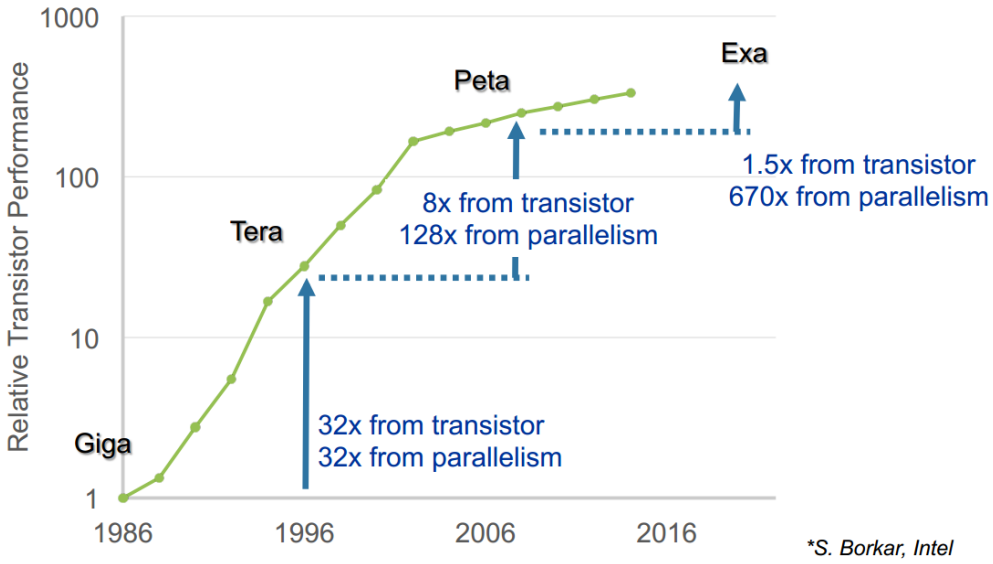


Figure 1.2: Performance from parallelism (source: [117])

The challenges of design-space exploration are further exacerbated by the lack of available exascale architectures; as the research and proposals are still in their infancy, production systems are not available for development and evaluation.

In addition, existing HPC applications are not representative of the exascale applications [44]. Therefore, to cater to the lack of representative applications, the Exascale Computing Project team (ECP) [44] has proposed a variety of exascale proxy applications (e.g., [47, 87, 121]). These proxy applications are expected to mimic the behavior of exascale systems and thus enable evaluation and future development.

Therefore, in order to evaluate performance and energy consumption trends, we first need to have abstract machine models that are representative of exascale systems. Next, we need to be in a position to evaluate the proposed exascale proxy applications using the abstract machine

models. Finally, with the aforementioned resources in place, we need to efficiently model performance patterns, understand energy consumption behaviors, determine their trade offs, and propose application-machine configurations that maximize performance and minimize energy consumption.

1.2 Performance and Energy Modeling

Traditionally, a number of approaches have been used to study and analyze power-performance trade-offs for parallel and distributed systems. A well-established approach is to study and implement performance and power models using simulators. Simulators have been used extensively to study and analyze the behavior of applications on multiple versions of parallel and high-performance computing systems. For example [53, 68, 91] have used simulations to take advantage of detailed functional or cycle-accurate analysis of current systems as well as projection for future systems. While simulators have some advantages, for example the ability to imitate the behavior of hardware when it does not exists, nevertheless they incur overheads in terms of speed, accuracy, and in some cases require implementation of the complete software stack.

As with simulators, emulators [63, 82, 107] possesses similar capabilities, by mimicking the hardware behavior. Although emulators provide much more accurate predictions, they lack portability and are unable to estimate application behaviors for different architecture than the one implemented in the emulating environment, until they are implemented as part of the emulator. In some cases, their overheads can be a challenge as computations are scaled up.

Empirical modeling methods (such as [28, 115, 164, 171]) use partial but direct measurements to predict application behavior. These methods use measurements such as execution time, communication costs, and measures of various performance counters. As the measurements

are obtained directly from the underlying hardware the confidence of the predictions is much higher. Although the process of collecting empirical results involves an overhead until the process can be automated. Due to their inherent nature, such methods are portable.

Analytical modeling (statistical, machine-learning, black-box modeling etc.) are typically developed using a training data set and evaluated on a testing data set. However, analytical models lose confidence in their predictions when input parameters are varied beyond the range of their training data.

Considering the benefits and challenges of the aforementioned approaches, we propose mechanisms for modeling that are accurate, portable, efficient, and applicable to current and future architectures.

1.3 Design-Space Exploration

The notion of developing frameworks that focus on improving performance and enabling efficient energy consumption, while taking into account both applications and hardware is paramount in identifying near-optimal configurations. A well-balanced hardware software co-design approach would target the optimization space for both application and hardware in order to maximize performance and energy gains. Considering the scope and complexity of exascale architectures, such co-design techniques would require automation and enable system developers to make intelligent decisions in meeting performance goals.

There is a dramatic increase in the scale of design parameters to evaluate when considering both application parameters and machine configurations. These parameters include aspects such as optimal number of ranks, communication strategies, data layout and placement, accelerator configuration, work distribution approaches, and so forth. Thus, it is not only important

to accurately and efficiently model performance and energy, but also to be able to efficiently explore the myriad of design parameters when evaluating their trade offs.

To explore design space of hardware and software in isolation, application developers have relied on both compiler based solutions (e.g., [10, 28, 94]) and runtime solutions (e.g., [36, 141, 161, 168]). While these approaches are useful, they come at the cost of a burden on the programmer [172]. In addition, these methods either apply to the application or the hardware, but not both. Though these solutions can provide performance and energy gains, they are typically limited in terms of portability, and scalability. In the context of exascale architectures, these solutions when applied alone are not sufficient to maximize performance and energy gains [30, 34, 48, 89, 90, 122]. Moreover, there is an inherent inability to reconfigure in most of the hardware configurations in current architectures.

Towards this end, we propose hardware-software co-design techniques in the context of current and future architectures, that evaluate the effect of the application sensitivity towards various hardware configurations. We also explore the effect of a scalable communication mapping and modeling technique that studies the computation and communication aspects of a proxy application on an exascale proxy architecture implemented in Aspen domain specific language.

1.4 Aspen – A Domain Specific Language

To evaluate performance and energy models on future exascale architectures, we need to be able to test them using a scalable, reconfigurable and portable platform. As mentioned earlier, simulators and emulators have inherent characteristics that preclude them from use at exascale.

Aspen, a domain specific language, provides a framework to analytically model the application and underlying hardware using abstractions. Aspen has been proven to be scalable, accurate,

portable, and efficient as compared to alternate options [146, 147].

In this dissertation, we develop, evaluate, and validate performance and energy consumption models as well as our approach towards design-space exploration using Aspen. We develop application and machine abstractions using Aspen and validate their behavior patterns. We develop performance and energy models using Aspen and ensure correctness with empirical evaluation at smaller scales and at current cluster scales. We leverage Aspen's ability to scale up to evaluate metrics at exascale. We extend Aspen to include performance and energy models and use sensitivity analysis to determine opportunities for optimization. We also explore the benefits of Aspen in implementing exascale proxy architectures and study the effect of computation and communication modeling, as well as their mapping on exascale clusters.

1.5 Research Considerations

After studying state of the art and related research in the area of performance and energy modeling, we believe that there is a dire need for considering application execution patterns and underlying hardware configurations while implementing performance and energy models. Moreover, we appreciate that communication and computation will be effected by extreme scale as well as the complexity and heterogeneity of compute nodes. Therefore, an in-depth analysis of communication patterns at scale is also required.

In this context, we consider the following aspects of performance-energy modeling and design space exploration of current and future architectures:

- Capturing application signatures at compile or execution time to enable performance and energy modeling
- Developing and validating analytical, statistical, and empirical modeling techniques that

are accurate, scalable, efficient, and reconfigurable

- Exploring the design-space for current and future architectures while considering application and hardware characteristics
- Developing methods that enable automation of design-space exploration and subsequent adaptation, such as mapping of proxy applications to proxy architectures while considering communication and computation characteristics

1.6 Research Contributions

In this dissertation, we focus on the development of performance and energy models for exascale systems, in light of the goals set by ECP [44]. We do so by proposing hardware and software co-design approaches for maximizing performance and minimizing energy consumption. We develop automated techniques for design-space exploration by building on Aspen DSL and incorporating extensions where needed. Our research contributions in this dissertation, also summarized in Figure 1.3, are:

- Analytical and machine-learning-based models of performance and energy consumption. The models are developed by identifying significant counters that reflect system (application and hardware) behavior, and mapping these counters to corresponding hardware components. We analyze scalability of the models using extensions implemented for Aspen. We also validate our findings with empirical analysis using Cori's supercomputing facilities at NERSC.
- Determining sensitivity of performance and energy consumption to various hardware configurations. We also validate our findings with empirical analysis using compute clusters at Virginia Tech.

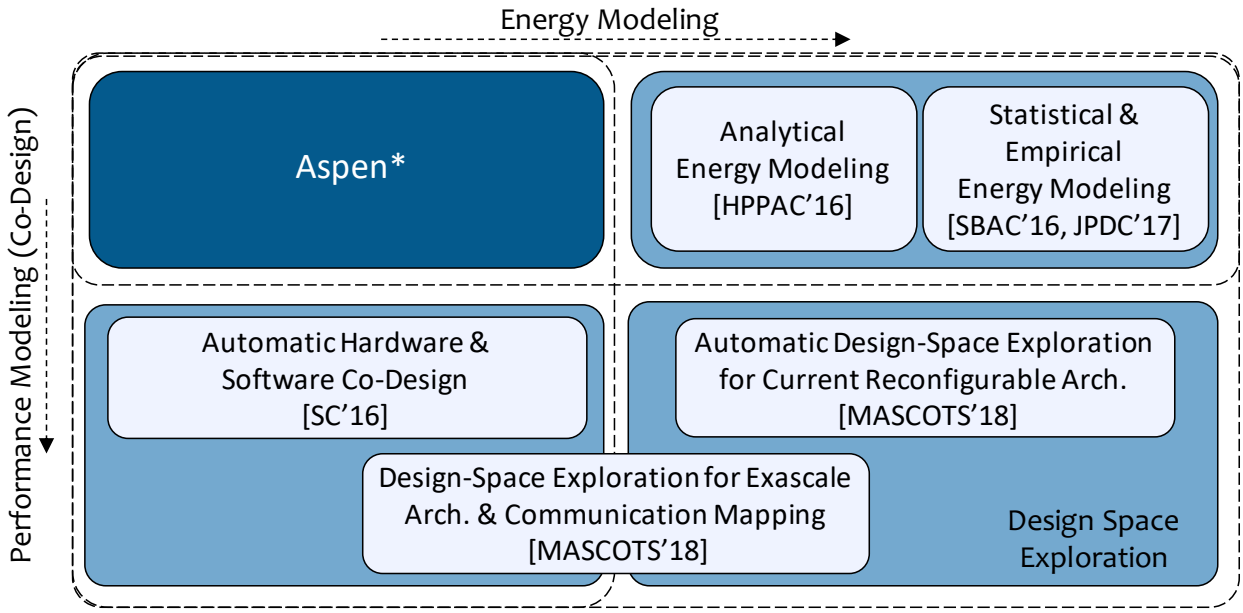


Figure 1.3: Research contributions and their relationship with Aspen

- Analytical modeling of computation and communication for exascale proxy systems (i.e., proxy applications and candidate exascale architectures), as defined by the ECP. This includes extensions to Aspen in the form of implementations of proxy applications and architectures. We also validate our findings by using the bounds defined by ECP.

- A modeling framework that automatically generates application and machine models as Aspen's abstractions for design space exploration, and evaluates its sensitivity to CPUs and GPUs, in a heterogeneous system. The framework also recommends near-optimal configurations using what-if analysis.

1.7 Organization of Dissertation

This thesis is organized as follows. Chapter 2 presents the state of the art research in relation to our thesis contributions. These include performance and energy modeling, hardware software co-design, hardware sensitivity to computation patterns, and a comparison of various techniques used to map the communication and computation of application onto hardware to achieve maximum scalability.

In Chapter 3, we propose an empirical and analytical energy estimation model using hardware performance measurements techniques. These serve as a guideline to the user as to which device/accelerators should be selected to enable energy savings. We compare and contrast the empirical and analytical energy estimation models with an analytical energy model developed in Aspen's grammar. We also quantitatively and qualitatively compare as to which approach provides better energy estimation with respect to speed, accuracy, initial setup cost and portability. We implement our two models on five proxy applications with varying input sizes, to determine the usability of the models in multiple execution scenarios.

We propose Prometheus in Chapter 4, a composable hardware-software optimization framework. As part of Prometheus, we develop a combination of analytical and machine-learning techniques to capture application characteristics. These application characteristics are subsequently used to determine the hardware-software configuration for near-optimal performance and energy consumption. Prometheus leverages Aspeńs ability to represent application and machine behavior and develop an automated optimization search space exploration framework around Aspen. We test Prometheus on four widely used proxy applications, LULESH, CoMD, CG and CoEVP. We demonstrate that Prometheus identifies near-optimal hardware software configurations and we validate the experimental results with the empirical measurements collected on compute clusters at Virginia Tech and NERSC's Cori.

We provide details regarding the implementation of proxy homogeneous exascale architecture in Aspen. We validate the proxy architecture for theoretical bounds against the values proposed by ECP in Chapter 5. We implement communication and computation modeling and mapping framework on top of the proxy architecture for CoMD.

Chapter 2

Related Work

Our research focuses on modeling and design-space exploration to estimate and predict performance and energy consumption of HPC applications and therefore help to improve the overall utilization of the hardware. In this section, we review the state of the art research that pertains to performance and energy modeling, as well as supporting frameworks and domain specific languages.

2.1 Performance Modeling

Estimating the performance of a scientific application can be done in multiple ways, including simulation, profiling and analytical models. A large body of work has been based on analytical models often expressed with various formulations such as LogP [35] and its variants, the BSP model [159], Amdahl's law [6], Gustafson's law [69] and so forth. Amdahl's law considers modeling for a fixed problem size, while Gustafson's law says that a larger problem size can be solved in less time if we add more processing workforce.

LogP [35] models the communication performance between two entities. It models the communication performance using latency between two processes, overhead incurred between two communicating parties when they are not performing any other operation. It also models the minimum time interval between consecutive message transmission and reception, and total number of processes involved in communication. LogP has the ability to analyze a variety of algorithm styles. Moreover, it is not tied to any programming style or communication protocol. It provides modeling details for shared memory and distributed memory, and a variety of message passing paradigms. LogP has been extended into a number of variants, such as logGP [5], that extends logP model into sending and receiving long messages. It quantifies the time per byte for a long message as well as available bandwidth for a communicating party. Another variant is logfP [75] that sends and receives consecutive messages in one communication.

BSP (Bulk-Synchronous Model) specifies the communication between a set of processors with local memory, and provide specifications for barrier synchronization. It specifies the point-to-point messaging using a combination of local computation, communication and bonded by a synchronization scheme. BSP has been extended to many variants e.g., BSP [39] and D-BSP [14].

Artificial neural networks have been used in conjunction with compiler techniques to provide performance modeling. COBAYN [10] is a compiler based optimization and auto-tuning technique that uses machine learning algorithm for auto-tuning purposes. Its primary goal is to dynamically improve application performance using Bayesian networks. A similar approach for compiler based approach for application optimization that uses machine learning techniques is proposed by Sameer et al. [94], which uses artificial neural networks to guide compiler about the sequence in which optimizations need to be applied to avoid complexity and to achieve better overall application optimization. Curtis-Maury [37] uses artificial neural networks (ANN) to identify concurrency configurations of multiprocessors systems. They use ANN to profile

the behavior of their system and model performance and energy consumptions based on the execution pattern.

These ANN based frameworks provide a more realistic parallel machine to the programmer when creating parallel algorithms. They also force a parallel program to be developed in a rigorously structured format and define several parameters so that the performance of a program can be analytically estimated. In addition, they can be combined with application requirements, such as the required number of floating point operations to solve the problem. Hoefler et al. used this approach by defining a simple six-step process to create application performance models [74]. However, analytical approaches require considerable manual analysis of the algorithm, which may be challenging on large parallel systems. Some study power modeling for GPUs using linear regression modeling techniques has been studied [104, 124]. These linear regression models assume a linear relationship between power consumed and performance counters. Hong et al. [77] implements a system level analytical power model that uses uses the relationship between performance and power to find the optimal number of nodes to run an application on GPU.

Simulators and profiling tools provide an alternative approach to model and predict the performance. For example, simulators focused on the processor or node level architectures such as CACTI, SimpleScalar, ORION, SoftWatt and GEMS [20, 110, 132, 169] are extensively used in computer architecture to make predictive analysis and to guide design space exploration. Although simulators provide a low-level, high fidelity models of the target architecture, they are less useful for gaining insights as to how parallel software perform at scale. There are other simulators that are focused on parallel applications such as the Wisconsin Wind Tunnel [123], PROTEUS [22] and the PACE toolkit [83]. Usually, these tools take source code as input or the executable itself and construct the simulation model. While these tools can reduce the overhead associated with the development of analytical models, the execution time required

to simulate each program instruction is a significant barrier to the adoption of such tools.

There are also several performance profiling tools that gather information during one or more runs of a program and present the dynamics of its execution (for example, frequency and duration of function calls, frequency of operation counts within blocks of code, memory consumption). The collected data is usually obtained through code instrumentation (either at the level of source code, at compile time, or during runtime), operating system monitors, and performance counters. Examples of such tools include PowerPack [62], VampirTrace [125], gprof [66], PAPI [108], SimplePower [166,179], SimpleScalar [25], PowerTimer [23], TAU [143], Scalasca [175], and HPC-Toolkit [3]. Profiling tools can provide accurate performance projections of a program in a given platform. However, they provide limited analyses and typically require that the target architecture be available at the target scale. Direct system performance and energy measurements are also used to direct modeling and trade-off strategies [80, 86, 153].

2.1.1 Qualitative comparison of performance prediction approaches

Table 2.1 summarizes a comparison of various prediction methods geared towards understanding application performance and energy behavior. We compare following classes of performance prediction methods: analytical modeling, simulation, emulation, empirical-measurement based modeling, domain specific languages.

Table 2.1: Comparison and contrast of prediction methods (✓: yes, ✗: no, ✓/✗: subjective).

	Analytical-based Modeling [28, 64, 81, 94, 115, 145, 159]	Simulation [20, 21, 82, 110]	Emulation [63, 82, 107]	Empirical-Measurement Based Modeling [16, 28, 115, 164, 171]	Domain-Specific Languages [12, 13, 114, 147, 157]
Accurate	✓/✗	✗	✓	✓	✗
Portable	✗	✓	✗	✗	✓/✗
Minimal Overhead	✓/✗	✗	✗	✗	✗
Scalable	✗	✓/✗	✗	✗	✓/✗
Automatic	✓/✗	✗	✗	✓/✗	✓/✗
Independent of Application	✗	✓/✗	✓/✗	✗	✓/✗
Easy to Use	✓	✓	✓	✓/✗	✓

We compare these methods in terms of the following qualitative parameters:

- **Accuracy**: degree of errors (prediction vs. empirical observation)
- **Portability**: ability to accommodate future architectures
- **Overhead**: amount of time and computational effort
- **Scalability**: valid results with strong and weak scaling
- **Automatic**: degree of user intervention required
- **Independent of Kernel/Application**: applicable to a broad range of applications and hardware configurations
- **Ease of Use**: complexity of method and degree of effort required in obtaining predictions

Analytical-based modeling methods (such as [28, 33, 64, 81, 94, 115, 145]) describe the characteristics of the application and hardware using mathematical models and estimated behavior. These methods include but not limited to statistical, machine learning, and black-box modeling techniques. These models are typically developed using a training data set and therefore their estimations are valid only within the range that the training data set covers. Thus, we lose confidence in the predictions when parameters are varied beyond the range of training data — e.g., we have low confidence in performance prediction when scaling estimates are made using number of cores or input sizes beyond which the model was developed. Similarly, a change in architecture would force researchers to develop a model from scratch and not be able to adapt to the changes.

Simulators (such as [20, 21, 82, 110]) provide good flexibility to incorporate changes in the architectures. However, they do present a challenge in terms of high development costs, use

and overheads. As with analytical models, the predictions loose confidence when they are used for predictions beyond the scope of which they were designed (e.g., scaling).

As with simulators, **emulators** (such as [63, 82, 107]) posses similar capabilities — by mimicking the hardware behavior. Although emulators provide much more accurate predictions, they lack portability and are unable to estimate application behaviors for new architectures, until they are implemented as part of the emulator. In some cases, their overheads can be a challenge as computations are scaled up.

Empirical modeling methods (such as [16, 28, 115, 164, 171]) use partial but direct measurements to predict application behavior. These methods use measurements such as execution time, communication costs, and measures of various performance counters. As the measurements are obtained directly from the underlying hardware the confidence of the predictions is much higher. Although the process of collecting empirical results involves an overhead until the process can be automated. Due to their inherent nature, such methods are portable.

2.2 Miscellaneous optimizations

There are several other notable contributions towards power and performance modeling. These include research by Vetter et al. [146] that presents cost based abstract machine model that attempt to understand the vulnerability of models in the context of domain specific languages [180].

In addition to simulation and profiling tools, the majority of research in performance and power modeling relies on ad hoc analytical models. Song et al. [144] propose an iso-energy-efficiency model for determining the problem size and clock frequency to achieve a desired level of energy-efficiency on a system. However, the perspective is hardware centric. In contrast, our work attempts to provide scientists with insight into applications as well as hardware,

in a modular, flexible and reproducible way.

There are several analytical approaches that have developed detailed GPU-specific models [76, 77, 103], however, they require in-depth knowledge of the GPU micro architecture. Finally, several application-specific studies exist have not been generalized yet [45, 155].

Domain Specific Languages and in particular co-design frameworks [12, 13, 114, 147, 157] have been developed to explore the combined effects of hardware and software optimizations. The manner in which these frameworks are developed dictate their characteristics, such as accuracy, portability, overheads, scalability, automaticity, independence of kernels and ease of use.

Most of the domain specific languages require representation of the application and hardware in their specific grammatical semantics. The level of details of representation may vary but they are always generated manually. Moreover, the experimental design is also manual.

A detailed comparison of various domain specific languages is provided in (§2.5).

2.3 Energy modeling

Energy modeling is a well researched topic for current as well as future architectures. Various approaches have been used to models energy, including analytical, empirical, simulation-based or a combination of them. We provide a summary of a range of the related work in this section.

Ge et al. includes parallelism effects and processor frequency when generating an analytical energy model [58]. They use a classification criteria to divide execution time into multiple groups according to runtimes and parallelism. They also introduce a performance metric called power-aware speedup to quantify the classification.

Ge et al. extended the same modeling approach for multi-core cluster [61]. They included the aspects of allocated cores per computing nodes into the analytical model and adjusted the power aware speedup formulation accordingly. Dynamic Voltage and Frequency Scaling (DVFS) [136] is a commonly used approach to manage power and energy consumption in Intel based architectures. DVFS enables the user to choose among a range of operating frequencies [177] and therefore provide a trade-off between improved performance and energy efficiency. DVFS has been explored at the application level to improve performance of processor [61, 139] as well as memory by Deng et al. [43]. Mishra et al. [119] propose a two-tier global power monitoring system, the first tier allocates a target budget to the chip multi-processor and the lower tier makes sure that the multi-processor never exceeds that power budget. DVFS has also been studied at microprocessor level in terms of CMOS technology [17, 106, 129]. Isci et al. [79] presents a monitoring and control system based on DVFS for chip and core level based on power and temperature used by the application.

In addition to applying DVFS on a chip level, power and energy management has also been employed on large scale HPC systems. Spiliopoulos et al. and Lim et al. [100, 148] propose approaches to monitor the system workload and accordingly adjust the power/performance curve to achieve the best prediction for the large scale system. Dynamic Voltage SCaling (DVS) has also been used to provide power/performance trade-offs on power aware cluster by Ge et al. [60]. Freeh et al. [53] studies the effect of trade-offs between performance and energy in the context of large scale systems using MPI framework. Rountree et al. [138] predicts the performance of the system using performance counters and linear programming techniques under the influence of DVFS. Their optimization let the cores on the critical path explore the energy saving opportunities by slowing down the non- critical nodes proportionally.

Power-performance tradeoffs using simulators have been studied extensively. For example, some approaches [53, 68, 91] have used simulations to take advantage of detailed functional-

or cycle-accurate analysis of current and future system hardware design. The granular detail comes at the price of speed, and sometimes a full software stack (i.e.,OS, middle ware, and communication libraries) is required to ascertain parallel and distributed system power-performance trade-offs. In contrast, abstract models by Hong et al. [78] that use performance counters as proposed by Frank et al.([16]), can aid with runtime adaptation, though the models are typically application agnostic and are primarily useful for identifying hardware bottlenecks. Unfortunately, the cumbersome nature of simulations make them less approachable for runtime adaptation. While the speed of empirical and analytical approaches based on hardware counters can help with runtime adaptation, the lack of detail about the underlying application results in runtime decisions of a generic nature. Such decisions are typically less than optimal for the broad set of applications common in high-performance systems.

The flexibility of having several configurations to choose from, combined with a fine granularity of performance- and power-related parameters is useful in in developing models. Examples of widely used simulators include by Brooks et al. [24]. and by Li for GPUs [99]. To address the limitation of simulation speeds, emulation based approaches can be used to estimate energy consumption as proposed by Hong et al. [78]. A drawback of simulation approaches, in contrast with empirical-based approaches, is the need to incorporate the internal details of the hardware to reflect its real-world behavior. This requirement inhibits simulator development as well as its usage.

Analytical models come in many forms. Some models, such as logP by Culler et al., [35], captures the fundamental aspects of hardware configurations and are combined with details of the algorithm or software to analyze performance. Such models can be extended to consider other hardware costs as well [149]. These models are related to the proposed work in a manner that they combine aspects from both hardware-related empirical measurements and details of the application. They are then used by the model to predict the performance cost. In contrast,

we design a fine-grained, application specific, model and implement it either as part of or atop Aspen.

Analytical models of power and energy are more recent but follow a similar structure. Some models, such as the iso-energy-efficiency model by Song et al. [144], are based on analytical performance models with extensions to include power and energy.

Belllosa et al. were among the first to use empirical performance measurements to estimate application power and energy usage [16,116]. Other scholars have also adopted this approach with some success, for example Dongarra et al. [46]. This approach has also been extended to include accelerators such as GPUs [64, 145] and to analyze offloading trade offs as proposed by Satoshi et al. [72].

A detailed survey on GPU energy estimation contains several related approaches as one proposed by Vetter et al. [120]. Here empirical performance data is collected for stress-test benchmarks combined with power measured with external system probes. The model consists of correlating events to an empirical cost in terms of power or energy (e.g., joules per instruction). In some cases, regression or neural networks are used to determine event correlation. These approaches are limited to measurable performance events and the granularity with which power can be measured. Patki et al. selects best configuration of the device for running an application based on profiled configurations [130].

2.4 Power/Energy Management

Effective power modeling and management is a widely researched topic for current and future architectures because of the number of improvements it provides (e.g., reduced energy consumption and effective resource management). We consider related research in terms of

performance, energy management, as well as performance and energy modeling.

Hardware architects also perform energy management and modeling as part of their product design. Academic researchers develop analytical, empirical and machine learning modeling based on the applications execution on the hardware provided by vendors. We discuss few approaches to enables efficient performance and energy management here:

2.4.1 Energy Management: Vendor-Provided Approaches

In recent HPC and enterprise systems, architects and vendors have put special emphasis on adopting hardware techniques to improve power and energy measurement and maintenance techniques. For example, dedicated power management units (PMU) and Voltage-Frequency Islands (VFI) are enabled in Nehalem [67], Sandy Bridge [135] and IBM POWER7 [170] architectures to enable fine grained power management at runtime. Voltage frequency island is a circuit optimization technique for multi-processor SoC architecture that enables various voltage and frequency scaling for different power management domains on a chip. Voltage and frequency scaling provides significant opportunities for power management on the system. These power management units also enable fine-grained offline management of VFIs.

2.4.2 Energy Management: Approaches Combining Software in Conjunction with Hardware

An unoptimized application execution on hardware can be a hurdle in achieving full advantage of power saving capabilities provided by hardware. Although, software can be optimized to its full capacity to explore full performance and energy capabilities of a system, but it comes with an added overhead on the programmer to exploit the full potential of the application.

A widely used technique to enable power/energy management is Dynamic Voltage and Frequency Scaling (DVFS) [134, 176], that is available as part of most Intel [134] and AMD [32] processors. With DVFS, the operating frequency is linearly reduced to modulate the per-core processor frequency and hence have the cumulative effect of reducing the overall energy consumption.

Model-State Registers (MSRs) have been deployed in recent architectures of Intel and ARM systems that provide more control of performance and energy tuning to the user. The power and energy management of Intel's Sandy Bridge architectures and beyond have been partitioned into multiple domains (e.g., clock, memory) to enable more fine-grained energy measurements and management. All these architectures encompass MSRs that allow management of energy in kernel mode.

Intel has also enabled energy modeling using MSRs through Intel's RAPL (Running Average Power Limit) [70]. Intel's RAPL enables power management of individual cores of a processor within a socket. The RAPL interface enables energy consumption calculation on real clusters. It also provides power limiting capabilities on specific subsystems. The RAPL interface can be programmed using MSRs. RAPL enables reading and writing MSRs in order to enable register values accessed during application execution. It enables power/energy measurements for three domains, package, power plane 0 and DRAM package. It has the ability to provide measurements for power, energy, and time.

Similarly, GPUs also enable energy management [88] and measurement using a combination of vendor specified software models along with hardware capabilities (e.g., nvidia-smi, NVML [2], PowerMizer [181]). PowerMizer power downs idle components of GPU and PCIe bus interface if not in use.

2.4.3 Energy Management: Software-Based Approaches

There has been notable research towards measuring and managing energy consumption using compile time frameworks and runtime systems. Some of these approaches use code instrumentation and execution profiles to determine runtime characteristics of energy consumption.

CPUSpeed [152] adjusts the power draw of the system using profiled processor utilization. CPUSpeed is now a part of Linux kernel module. With the advent of multi-processor systems, MPI is typically used as the dominant framework to exploit parallelization of the applications. For example, Cameron et al. use PMPI (an MPI profiling framework) to explore energy saving options [26, 59]. Fab et al. [49] uses information provided by the system i.e., CPU utilization to estimate the dynamic power usage.

Rountree et al. developed a framework that uses a linear programming solver in order to determine an energy saving bound on the system [137]. The output of their framework is a schedule for the application that is generated using applications communication trace and a profiled power characteristics of the cluster. The overarching goal of their research is to generate a schedule for the application that specifies how long an application must run in order to stay under the energy budget of the cluster. Their methodology utilizes a combination of empirical measurements that require an execution trace of the application at each discrete frequency.

They further explored this research area using Dynamic Voltage Frequency Scaling algorithm [139], which classifies the program execution into critical path regions. These critical paths are then used to predict execution time of the entire application. The first assumption in their work is to rely on MPI point-to-point communication in order to generate task blocking strategies. Another assumption is to limit the number of MPI messages. This however limits the full performance potential of the application.

DVFS is also used to explore the impact of load imbalance in parallel programs. Freeh et al. presents one such approach [52], which relies upon MPI communication library calls to determine which node is the bottleneck in each iteration of the application's kernel. The knowledge of blocking nodes is used as background information to set the DVFS levels for each individual nodes. Each node is setup at an appropriate DVFS level so that they all reach the synchronization point at nearly the same time. Similarly Curtis-Maury et al. [36], explores the impact of integrating DVFS and DCT for reducing dynamic power consumption of a system. They propose a multi-dimensional on-line performance predictor that uses DVFS and DCT in the background to solve the complexities of frequency scaling on a multi core high performance computing system. A similar approach is proposed by Koller et al. [92] for large clusters of nodes. They use application signatures to determine the power drawn by the data centers. In this process, they use application specific parameters, such as throughput, to determine the applications performance.

Similarly Lim et al. [100] enables energy optimization and management using profiling of MPI communication calls. Cameron et al. propose *CPUMiser*: a system-wide application aware DVFS based runtime system [61] that enables generic power aware computing without much overhead. They divide the program execution into several execution-time slices, each of which depends upon the current frequency used by the program. They also use performance counters [16] and empirical measurements to determine which frequencies to set for the next time slice.

Similarly, memory is also considered to be an important energy consuming hardware components. Qiang et al. [176] propose a compiler based energy management technique that enables optimization by identifying memory bounded regions in the code.

2.5 Domain Specific Languages and Design Space Exploration

Our performance and energy models are designed for Aspen Domain Specific Language [147] to explore and develop performance models for current and future architectures. While Aspen is originally designed to explore the architectures and capabilities of exascale architectures, it has reliable performance modeling capabilities that can be developed further. Aspen domain specific language takes inspiration from source code annotation to trigger and monitor application performance at runtime. An inspiration for performance modeling using source code annotations is by Vetter et al. [163]. Vetter et al. [163], provide a mean for determining performance analyses from source codes. This performance analysis can be compared with the user expectations to improve overall execution. Their framework allows users to insert annotations inside the source code that can collect data implicitly to verify assertion conditions. This framework can serve as a continuous monitoring tool for the application, where it shows no warning if the constraints are satisfied and throw an exception if a constraint is violated.

SKOPE [114] has the ability to generate analytical models that define data transfer between tasks and it uses empirical modeling techniques to understand performance scalability between the tasks. Similar to SKOPE is the framework by Tallent et al. in Palm [150]. It uses similar code annotation techniques as proposed by Vetter et al. [163] for abstraction and expressing complexity of the code. Song et al. presents Compass [97] that uses static analysis of the application to generate the performance model using performance prediction techniques using Aspen. Aspen has also been used explore the design space of applications in the context of future architectures [162]. Aspen is a also used in conjunction with CODES simulator to generate scalable workloads extracted from original applications in Durango [27]. Durango is tested for LULESH proxy application.

Palm also uses a combination of static and dynamic analysis tools to explore the application signatures. Panorama [40], explores another aspect of design space exploration by modeling the performance of complex scientific work flows. Panorama integrates the extreme-scale simulation with analytical modeling to understand the overall performance of scientific work flows. Mandal et al. [109] deals with large-scale systems failures and anomalies by presenting an end-to-end prototype system for modeling and analyzes the execution of complex scientific workflows. Their framework interfaces between Pegasus workflow [41] and Aspen [147] to provide an integrated modeling framework. Automation is a critical issue in the co-design of extreme scale systems. CatWalk [174] aims to automate the performance modeling activities of exascale computing project.

Lee et al. [96] propose a co-design framework for GPUs that explores the design space of hardware with potential applications to investigate the interactions among them. ExaSAT [157] is a framework that provides valuable insights in hardware and software trade-offs and enables user to perform deep-code analysis using proxy applications.

TiDA [158] is an abstraction library that takes inspiration from Aspen and ExaSAT to hide details of data composition, cache, data locality and memory affinity. Its main emphasis is on hiding data communication cost. It is especially useful for applications with stencil computations that uses tiling based data access patterns.

2.6 Communication and Computation Modeling

Future exascale architectures are expected to be extremely scalar and complex as compared to current high-performance computing system. Therefore, either we have to shape the existing performance and energy models to provide the scale needed for extreme-scale architectures or we design new modeling techniques for communication and computation that scales very

well with minimum overhead. Some communication modeling and mapping frameworks have been introduced e.g., Qilin. Qilin [102] proposes an automatic mapping of computation to a heterogeneous CPU-GPU architecture. Qilin mapping algorithms have been designed to cautiously distribute the work between the devices in order to achieve the improvement in time and energy consumption. ScalaTrace [128] reduces the cost of calculating the communication behavior of large-scale systems. They compress intra and inter node communication of MPI events that contains the information of an application's communication behavior. They also include the capability to replay the traces generated by the applications communication behavior. Liu et al. [101] proposes power and time efficient communication and computation modeling techniques by applying DVFS on CPU and GPU. Ahmed et al. [4] presents a communication model for predicting performance of extreme scale applications on Cray's Gemini 3-D torus interconnect network. They integrated their analytical model with MPI that provides a granularity of packet level on the interconnect.

In terms of modeling communication on a large scale system, it is important to consider both intra and inter node communication. Kumar et al. [95] studies the detailed analysis of intra and inter node communication of parallel wavefront stencil algorithm. Gahvari et al. [54] and Chan et al. [29] studies the analysis of expected latency and bandwidth requirements for exascale architectures using LogGP modeling of fast fourier transform. Faraj et al. [50] studies the effects of MPI collective algorithms at a scale of 16K nodes on IBM BG/P system. Some tracing tools also capture the communication events in large-scale systems e.g., DUMPI [82], as an integral component of Structural Simulation Toolkit and collect the communication traces of mini-applications. Similar to DUMPI is an extension in CODES simulator called TRACER, that predicts the network performance and interconnect behavior by simulating the messages on ROSS. A compiler based solution to identify communication behavior is proposed by Shao et.al [142], that uses a matching mechanism of an application compilers traces with the the

already characterized communication pattern. Their framework also distinguishes between static and dynamic communication patterns.

Simulators are also employed to study and model the communication patterns on large scale systems. Durango [27] combines Aspens benefits with ROSS compiler to generate the scalable workload, with added features for the communication on extreme scale, using the real applications. A number of other simulators are also deployed to study the communication behaviors e.g., Omnet++ [160], SST [82], Dimemas [133], Bigsim [182], Booksim [85].

In the next chapter, we will present two performance and energy modeling techniques for Aspen. The first one is a statistical and empirical performance and energy model that uses hardware performance monitoring techniques and empowers Aspen by determining the resource efficient device for application execution. The other model is a system-level analytical energy model that uses activity factors for processor, memory and communication to estimate the energy consumption. We also quantitatively and qualitatively compare both modeling techniques to determine which modeling technique should be preferred over the other depending upon the execution scenario.

Chapter 3

Performance and Energy Modeling

3.1 Introduction

With the anticipated complexity and scale of exascale architectures, maintaining efficient power consumption is one of the fundamental parameters of concern for the research community. Hence, it is of paramount importance to understand the power-performance trade-offs for both current and future architectures.

Current high-performance computing systems are designed to support execution of a wide range of parallel and distributed applications. The optimization strategies include application-specific software approaches, hardware-specific optimizations or a combination of them. In case of hardware-specific optimizations, some applications may benefit from high-performance accelerators while others may not. In the case of a heterogeneous system, an application can be adapted to use the best available hardware to optimize performance while staying under the power budget. Any optimization and hardware adaptation framework requires an in-depth understanding and analysis of the application performance-energy tradeoffs to enable dynamic

mapping of the application to the best suitable configuration of hardware. In order to provide the best performance of an application with respect to hardware, optimization strategies must be efficient, accurate, portable and scalable to accommodate a range of application and hardware configurations and to enable runtime adaptation on current as well as future exascale architectures.

In this chapter, we propose an empirical and analytical energy estimation model using hardware performance monitoring techniques, which serves as a guideline to the user as to which device should be selected in order to enable energy savings. We also validate the empirical and analytical energy estimation model in order to present the user with a summary of which approach to use i.e., embedded versus a framework that is independent of the constructs of Aspen. We also quantitatively and qualitatively compare the approaches in terms of energy estimation and evaluate their speed, accuracy, initial setup cost and portability. We test our two models on five proxy applications with varying input sizes, to establish the usability of the models in multiple execution scenarios.

This chapter presents the following contributions:

- Creation of accurate, fast and portable analytical models of power-performance trade offs that leverages empirical system measurements and analytical modeling
- a system level energy model "Aspen's Embedded Energy Model" (AEEM)) embedded in Aspen that uses performance constructs of Aspen's application and machine models to generate estimations (§3.4)
- Creation of the "Algorithmic and Categorical Energy Modeling and Estimation" (ACEE) runtime framework that uses statistical techniques for modeling power-performance trade offs in high-performance applications and systems (§3.4)

- Evaluation of five proxy application and multiple use-cases for ACEE and AEEM that apply application-specific analytical power performance modeling to runtime analyses (§5.2)
- Comparison and contrast of the two models to present a guideline for the user as to which type of model is better for a scenario (§3.8).
- Extending Aspen by implementing and testing a cache model

3.2 Introduction to Aspen DSL

Aspen (Abstract Scalable Performance Engineering Notation) [147] is a domain specific language (DSL) that provides specifications of various analytical performance models, most notably LogP [35] and BSP [159] using well-defined grammar. It also provides abstractions of application and machines in a grammatically complete manner. Moreover, the ability to include any application and to test its execution on a prototype abstraction for current and future exascale systems makes it a framework of choice for high-performance computing. Aspen provides abstractions for both application and machine using structures of the application along with resource usage characteristics of the hardware. The independent generation of abstract models for application and hardware enables analysis of application along with hardware or in isolation. It also provides study of current and future hardware architecture. These capabilities of generation, analysis and execution of performance models enables Aspen to rapidly generate intra-nodal models and conduct performance analysis as well as scalability analysis over various cluster configurations.

Aspen's grammatical semantics consist of two main components:

- *Application model:* An abstract application model, as shown in Figure 3.1, explains the

characteristics of the application including control flow and resource usage requirements. Examples of resource consumption descriptions include the number of floating point operations, memory loads and stores. Control flow in the model can represent constructs like iterations and sequential dependencies in the application model.

- *Machine model:* An abstract machine model, as shown in Figure 3.2, specifies machine characteristics, and hardware resources. These features include floating point units, cache and memory hierarchy, and interconnects. The model represents the machine in a hierarchy, from a full stack of clusters down to an individual node.

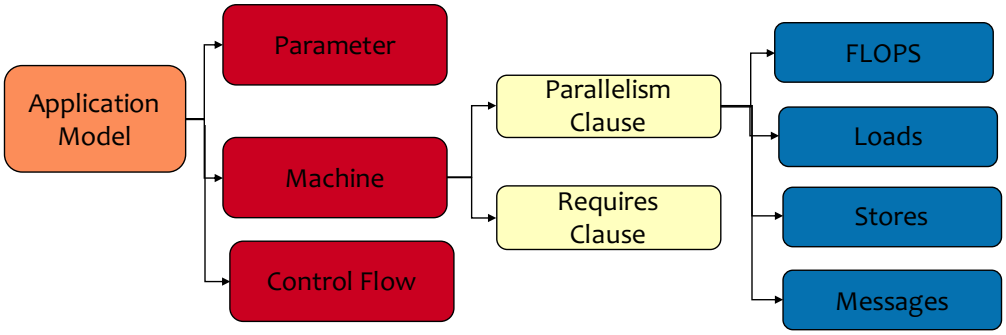


Figure 3.1: Components of an application model, adapted from [147]

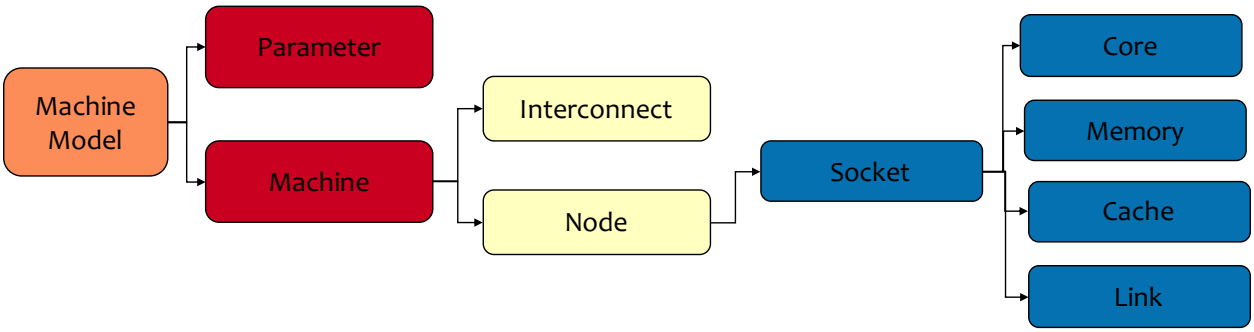


Figure 3.2: Components of a machine model, adapted from [147].

Aspen can analytically model runtime of an application on homogeneous as well as heterogeneous architectures, along with testing theoretical peak performance of a hardware architecture. It generates predictions by constructing a symbolic expression of application components

and their corresponding resource usage. It then maps it through the machine model to generate a symbolic expression of the application runtime. Aspen also has the ability to resolve resource conflicts while modeling the execution of an application on an abstract machine model.

3.3 Extending Aspen: Development and Generation of Cache Models

Aspen generates runtime predictions by constructing a symbolic expression of application components (kernels, Flops, memory accesses etc,) and their corresponding resource usage (used Flops, memory bandwidth utilization etc,), then mapping it through the machine model to generate a symbolic expression of the application runtime. Aspen also has the ability to resolve resource conflicts while modeling the execution of an application on an abstract machine model.

Before the machine model parameters can be substituted into the runtime expression, the application resource usage must be mapped onto the available machine resources. For example, *load* and *store* requirements need to be mapped onto the *cache* and *memory* hierarchy [118]. In the case of a *single inclusive cache hierarchy*, the mapping of *load* and *store* requirements can be done by means of a *cache model* that takes *cache size, associativity*, and *replacement policy* into consideration. The input to the cache model is the *memory access pattern* of the application, and the output is the *predicted memory traffic*. This predicted memory traffic can be used to calculate the memory time for the *analytical estimate* of the runtime and to determine the *memory activity factor* for the analytical energy model described in section 3.4.

For input to a cache model, the usual way to represent an application's memory access pattern is in the form of *reuse distances*. For a regular memory access pattern, such as that displayed

by an affine loop, the reuse distance can be expressed in Aspen for loads and stores by using a trait. For more complicated memory access patterns, we use *reuse distance distributions (RDDs)*. Aspen's mapping component can use a *reuse distance* trait together with a *probabilistic model* for a set associative cache to predict whether or not the modeled memory operations hit in cache. In the case that an RDD is supplied, Aspen hands this off to third-party cache modeling software, such as the last-level cache model described in [15], to get an estimate of the memory traffic generated by the loads. The RDD can be specified as a parameter in the Aspen application model or can be input from a file. An example for Matrix Multiply that shows the two ways of modeling reuse distances is shown in Listing 4.1. In this example, the reuse distances for the loads and stores are specified as traits, so these will be used. If these traits were not given, the closest enclosing reuse distance *cumulative distribution function (CDF)* would be used. Reuse distance distributions can be specified at kernel granularity.

Listing 3.1: Cache modeling in Aspen

```
// Simple Matrix-Matrix Multiply
model matmul {
  param n = 1000 // Input size, iteration count
  param nTimes = 5 //Iteration count
  param wordSize = 4 // Word size in Bytes
  param rddCDF[] = {0.5, 0.75, 1.0} // Data reuse parameters
  param rddVals[] = {2.0, 2000, 1001000}

  data matA as Matrix(n, n, wordSize) // Data structures
   ...

  // The matrix multiply kernel
  kernel matmul {
    execute [1] "mainblock"
    {
      ... //Flops specification
      ... //Loads and store specification
      loads [n^3-n^2] of size [wordSize] from matA
            as rd(n*n+n)
      ...
```

```
        loads [n^3—n^2] of size [wordSize] from matB
                as rd(2*n)
        ...
        loads [n^3—n^2] of size [wordSize] from matC
                as rd(2)
        stores [n^3] of size [wordSize] to matC
                as rd(2)
    }
  }

  kernel main {
        iterate [nTimes] { call matmul } // Control Flow
                }
}
```

3.4 Energy Modeling in Aspen

To enable power-performance modeling in Aspen, we present two approaches here. The first approach, *ACEE*; uses statistical modeling to combine significant performance counters as a way to estimate energy. The statistical modeling is done outside Aspen and helps Aspen make decisions regarding scheduling of the application in order to conserve energy of the system while maintaining performance goals. The second approach, *AEEM*, uses the modeling semantics of Aspen to estimate the energy consumed by the application on a hardware platform.

3.4.1 ACEE, Aspen's Extrinsic Energy Modeling and Estimation Approach

ACEE (*Algorithmic and Categorical Energy Estimation*) is an application and device specific modeling framework atop Aspen. It uses information from performance counters in order to observe the characteristics of the application and uses this information to generate a statistical estimation of energy use. It interacts externally with the Aspen toolkit. The overarching goal of

ACEE is to implement and validate a technique that generates energy models and estimations independent of Aspens grammar and should not require any changes to the Aspen DSL or to the application and machine models created inside Aspen. ACEE incorporates empirical and analytical modeling techniques to observe application's characteristic and behavior at runtime in order to generate energy consumption patterns.

We use resource consumption of an application in order to construct an application's characteristics signature. There are numerous ways to capture an application's signature, we use performance counters in this work. Performance counters [171], [2] have been studied by a number of researchers to understand application execution characteristics. Given the number and types of counters available in today's architectures, it is important to choose the right technique to capture the behavior of performance counters at the application runtime. We use LIKWID [154] to measure performance counter data.

LIKWID is an open source tool that has the ability to collect both core and uncore events at runtime. It has a wide list of counters that the user can choose from. LIKWID also provides some preset groups of counters, e.g., all counters corresponding to *Number of micro-operations* can be collected using a group configuration. Similar groups exist for *Translation look aside buffer, caches, memory bandwidth* and *latency*, etc. LIKWID also enables measurement of energy consumption values. The list of counters can be extended or updated with the change of architecture. We use *nvprof*, *NVML* and *nvidia-smi* to collect counters for GPU. Tables 3.1 and 3.2 shows the set of performance counters that we have used in our experiments. We explain the working of ACEE by presenting the steps below:

Table 3.1: Performance counters for CPU

Performance counter	**Description**
UOPS	UOPs execution info
UOPS_RETIRE	UOPs retirement
TLB_INSTR	L1 instructions TLB miss rate/ratio
TLB_DATA	L2 data TLB miss rate/ratio
UNCORECLOCK	All clocks
UOPS_EXEC	UOPs execution
UOPS_ISSUE	UOPs issueing
L2	L2 cache bandwidth in MBytes/s
RECOVERY	Recovery duration
L3CACHE	L3 cache miss rate/ratio
L3	L3 cache bandwidth in MBytes/sec
ICACHE	Instruction cache miss rate/ratio
NUMA	Local and remote memory accesses
ENERGY	Power and energy consumption
CLOCK	Power and energy consumption of clock
FLOPS_DP	Double precision MFLOP/s
QPI	QPI link layer data
CBOX	CBOX related data and metrics
FALSE_SHARE	False sharing
FLOPS_AVX	Packed AVX MFLOP/s
CYCLE_ACTIVITY	Cycle activities
L2CACHE	L2 cache miss rate/ratio
MEM	Main memory bandwidth in MBytes/s
DATA	Load to store ratio
MEM_SP	Single precision memory bandwidth in MBytes/s
BRANCH	Branch prediction miss rate/ratio
MEM_DP	Double precision memory bandwidth in MBytes/s
FLOPS_SP	Single precision MFlop/s

Table 3.2: Performance counters for GPU

Performance counter	Explanation
gld_request	Global memory requests
l1_global_loads_hit	L1 hits for global loads per SM
l1_global_load_miss	L1 miss for global loads per SM
global_store	Global store instructions executed per warp per SM
l_load	Local load
l_store	Local store
exec_inst	Instructions executed
Branch	Branch executed
divergent_branch	Divergent branches executed
shared_load	Number of executed shared load instructions per warp
l1_shared_bank_conflict	Number of shared bank conflicts per SM
T_hits	Texture 0 cache sector queries
T_miss	Texture 0 cache sector misses
l1_shared_bank_conflcit	L1 shared load and store bank conflict
shared_load	Shared memory load and store conflicts

Step-1: Capturing Application and Device Characteristics

Our framework takes advantage of empirical modeling techniques in order to reuse the data that has already been collected and to provide faster results whenever we can. ACEE stores all energy estimated values in a log file corresponding to the application, its input size and characteristics along with hardware specifications.

We start by doing a quick search of comparing the current application and hardware with existing profiled results and observe if the exact match exists. If an exact match exists, we use the stored results. If a match does not exist, then we proceed to collect the application and device characteristics. We use the application and its data size as input parameters to

our framework. The working of the ACEE in terms of state-transition diagram is shown in Figure 3.3.

We begin by executing the application on CPU and GPU. This enables us to collect performance counters on each device simultaneously. In our experiments, we consider a heterogeneous platform consisting of a CPU and GPU, our framework can accommodate other accelerators as well. It is important to note that we collect counters on each device multiple times, e.g., running FFT on CPU and GPU required approximately 50 samples in order to satisfy the confidence interval criteria [105]. We also collect multiple samples of energy consumed by each device to satisfy the statistical conditions. We use RAPL [70] for CPU and NVML [2] for GPUs to collect the empirical energy consumed by each device for validation purposes. The hardware counter information is stored in a log file along with execution time and consumed energy.

Step-2: Selecting the device

After the required samples of hardware performance counters are collected, we short-list the device upon which we can apply regression modeling. The criteria for short-listing the device can be energy efficiency, efficient execution time or a combination of the two. The default selection criteria is to short-list the most energy efficient device. In the worst situation, where users do not want to short-list the device and choose to generate regression model for each device tested, then this step of short-listing the device can be excluded.

Algorithm 1 shows the steps required to select a device in a pool of devices of heterogeneous architecture.

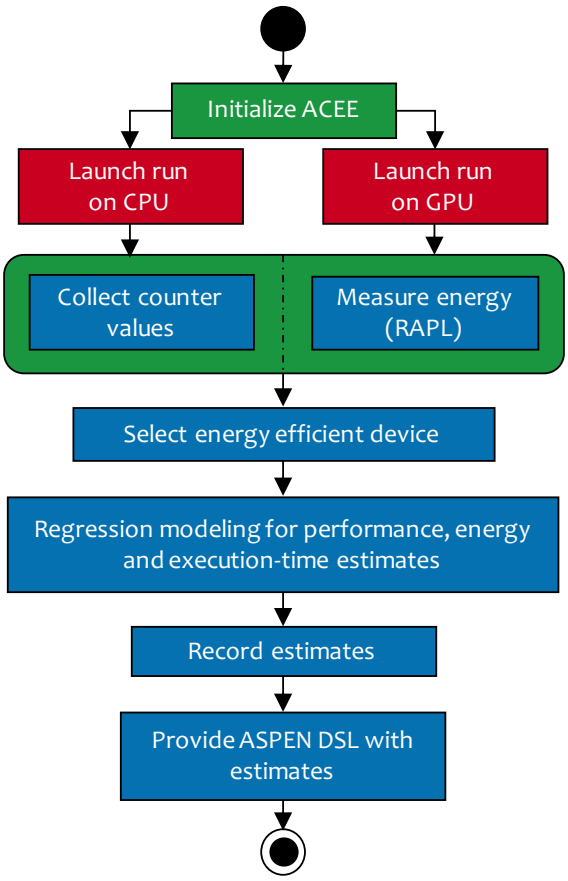


Figure 3.3: ACEE's state transition diagram showing the process by which we estimate energy and find the most energy efficient device.

Step-3: Energy estimation model generation

The selected device from the previous step will be used to generate regression analysis. The log file containing the profiled performance counters is used to generate multi-variable linear regression. Before applying linear regression, we assure that all assumptions required to generate best fit model for energy estimation are met.

- We collected enough samples of the application run in order to satisfy the confidence interval criteria.
- The relationship between dependent and independent variables is linear. We used scatter

Input: Application type, input size
Result: Application's estimated energy consumption
if *energy estimation record exists* **then**
    return record;
else
    `/* Run application on CPU */`
    Collect CPU hardware counters using LIKWID;
    Query RAPL for energy consumed;
    `/* Run application on GPU */`
    Collect GPU hardware counters using NVPROF;
    Query NVML for energy consumed;
    Compare energy consumption and performance;
    Short-list best energy efficient device;
    Apply linear regression to estimate energy consumption of chosen device;
    Record results;
end

Algorithm 1: Short-listing of device to be tested for regression modeling for ACEE

plot to demonstrate the linearity of variables. In our scatter plot all the points either lie on the line or around it, which showed the linearity in relationship.

- All the variables under consideration for regression analysis were multivariate normal. A histogram is used to test multi-variate normality. A QQ-plot is also used in our experiments to depict normality. Figure 3.4 shows a QQ-plot from our experiments with a sample size of 50. As can be seen all the points either lie on the line or around it, we also present a quantile box plot which shows normal distribution.

- There was little to no correlation among independent variables. In statistics, correlation matrices are generally used to find correlation between independent variables. A Pearsons bivariate correlation generated between independent variables showed the total correlation value to be less than 1.

- There was no autocorrelation in the data. Generally residuals are used to test autocorrelation, where a set of independent residuals shows no autocorrelation.

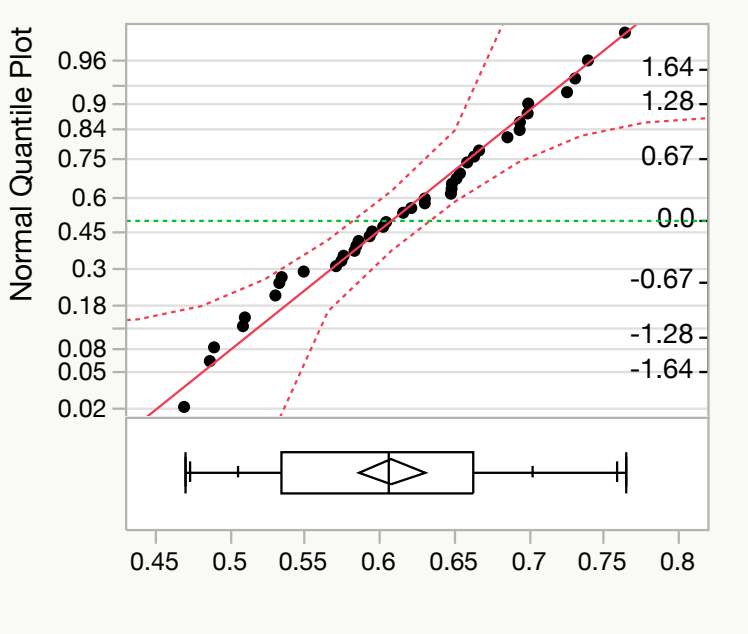


Figure 3.4: QQ-Plot for a sample size of 50 elements

- The error between independent and dependent variables was same across all numbers of independent variables.

After fulfilling all the assumptions of linear regression, and short-listing of device, we apply multi-variable regression analysis corresponding to performance counters and energy consumption on the testing data. We divide the full data set into two parts, training data set and testing data set. In the initial part of regression, we estimate the parameters on training data and we later assess the accuracy of the model on testing data. The overall result of the linear regression provides us the estimated energy for the selected device. The following equation shows the mathematics of the regression analysis:

$$F(X) = \begin{cases} \sum_{i=1}^{n} \sum_{j=1}^{m} \beta_{ij} X_{ij} + \alpha_{ij} & \text{where } \beta_{ij} \neq 0 \\ \sum_{i=1}^{n} \sum_{j=1}^{m} \beta_{ij} Z_{ij} + \alpha_{ij} & \text{Otherwise} \end{cases}$$

where β is the performance counter coefficient, X is performance counter type, α is the intercept, $F(X)$ is estimated energy value, n is iteration count/number of samples of performance counters, m is the number of hardware counters and Z are the performance counters whose value is never zero even when system is idle, e.g., clock frequency. In the other situation, when all the counters corresponding to application execution are zero, still the value of F(X) will not be zero, as some counters e.g., compute cycles are never zero and they represent the idle energy of the system.

X_{ij} indicates the values of collected performance counters, and it is replaced by all significant counters specified by regression analysis. In short, it can be written as:

$$X(ij) = Val(c(ij)))\epsilon\Big[c1, c2, c3....cm, 1 \leqslant j \leqslant m\Big] \tag{3.1}$$

where $c1, c2, c3$ represents performance counter#1, performance counter#2, performance counter#3, and m is the total number of counters available in the sample set.

After generation of the multi-variable linear regression model, we send the estimated energy values to Aspen. This will help Aspen in the execution of the application on the energy efficient device selected by regression analysis and enable Aspen to accommodate the application characteristics in order to stay under an energy/power budget.

3.4.2 AEEM: Aspen's Intrinsic Energy Estimation Model

The second model that we present is AEEM (Aspeńs Embedded Energy Model), which is an embedded model that uses Aspen's grammatical semantics and performance parameters, and map them to provide energy estimation. As Aspen has the ability to generate a number of performance models and parameters, e.g., FLOPS taken by the application, theoretical maxi-

mum FLOPs of the architecture, memory loads and stores, communication between different kernels of an application and communication between kernel and hardware, so we can map this information to a meaningful energy modeling. Inspired by Gemel et al. [56], we can map the energy consumed by a kernel into energy consumed on the system resources as well as map the communication between kernels.

Aspen provides information about operation counts, memory accesses, and communication for every application kernel. We can estimate the amount of energy required by a given kernel i as:

$$E_i = E_i^{system} + E_i^{comm} \tag{3.2}$$

where E_i^{system} is the energy required by kernel i to process the required Flops, loads, and stores clauses, where as E_i^{comm} is the energy spent to transfer the required messages.

Energy results from integrating power over time. For the processor and memory subsystems, we consider both static and dynamic power, which is:

$$E_i^{system} = T_i \cdot (P_i^{static} + P_i^{dynamic}) \tag{3.3}$$

where T_i is the runtime required by kernel i to process the required Flops, loads and stores clauses. This runtime is estimated using the Aspen performance model that includes the capability to produce bounds on predicted runtime by the kernel.

Static system power can be decomposed into the power of each contributing subsystem at idle state. In this work, we include power consumed by processor to compute the Flops, and the power consumed by the memory subsystem for the loads and stores clauses. Thus,

$$P_i^{static} = P_{cpu}^{idle} + P_{memory}^{idle} \tag{3.4}$$

where P_{cpu}^{idle} and P_{memory}^{idle} are the idle wattages for the processor and memory subsystems, respectively. These architectural parameters can be estimated using hardware specifications or can be measured empirically. Subsequently, their values can be modeled as constants in Aspen's machine model.

Dynamic system power can be approximated using dynamic-power-dissipation models based on activity factors (α_{cpu}^i and α_{memory}^i) extracted from the Aspen performance model for each subsystem. Thus,

$$P_i^{dynamic} = \alpha_{cpu}^i \cdot P_{cpu}^{dynamic} + \alpha_{memory}^i \cdot P_{memory}^{dynamic} \tag{3.5}$$

where $P_{cpu}^{dynamic}$ and $P_{memory}^{dynamic}$ are active wattages for the processor and memory subsystems respectively. Their values can be estimated using hardware specifications and modeled as constants in the Aspen machine model. In this work, we consider:

$$\begin{aligned} P_{cpu}^{dynamic} &= \text{TDP}_{cpu} - P_{cpu}^{idle} \\ P_{memory}^{dynamic} &= \text{TDP}_{memory} - P_{memory}^{idle} \end{aligned} \tag{3.6}$$

where TDP_{cpu} and TDP_{memory} are the thermal design power (TDP) for the processor and the memory subsystem, respectively.

As advised by Gamel *et al.* [57], we compute the activity factors using number of FLOPS ($flops^i$) and the number of memory accesses per second (mem_bw^i). These activity factors

are computed with respect to the kernel i and the normalization factor that represents the maximum capacity. The activity factors are represented as:

$$\begin{aligned} \alpha^i_{cpu} &= \frac{flops^i}{MAX_{flops}} \\ \alpha^i_{memory} &= \frac{mem\_bw^i}{MAX_{mem\_bw}} \end{aligned} \tag{3.7}$$

For the communication subsystem, energy consumption depends on the volume of messages transfered by kernel i ($message\_volume^i$), as well as on the network characteristics — network bandwidth (net_bw) and the active wattage of the network interface card ($P^{dynamic}_{nic}$). As with other subsystems, $P^{dynamic}_{nic}$ can be modeled as an architectural parameter in the Aspen machine model, which can be estimated using hardware specifications or can be measured empirically. Therefore, the energy consumed by a kernel i due to data communication can be expressed as:

$$E^{comm}_i = \frac{message\_volume^i}{net\_bw} \cdot P^{dynamic}_{nic} \tag{3.8}$$

Finally, the energy consumption of an entire application can be computed in the following manner:

$$E = \sum_{i=1}^{k} (call_i \cdot E_i) \tag{3.9}$$

where k is the number of application kernels, $call_i$ indicates the number of calls to kernel i, and E_i is the total energy required by kernel i.

3.5 Example Model Usage

In this section, we present the usage of the two models on Matrix Multiply application.

3.5.1 ACEE sample output

After running matrix multiply on CPU and GPU simultaneously and collecting performance counters, execution times, and energy usage, ACEE derives a regression model, the values of the model are shown for the CPU in Table 3.3. Table 3.3 lists coefficients (beta) with the corresponding counters, where intercept α is 153.3.

We used *SAS-JMP, R* and *Python+Numpy* in combination to conduct stepwise linear regression. A combination of these statistical and machine learning tools [113] automates the process of identifying inconsequential parameters and includes dominant variables in the model. We verified results by implementing the same using python scripts for Matrix-Multiply and compared the model and significance estimates. We studied significance of all the parameters for the Matrix Multiply. Figure 3.6 shows the normality of residuals generated by regression analysis. Similarly Figure 3.5 indicates the relationship between actual energy values and predicted energy values for matrix multiply, and it has an R-squared value of 99.4.

Table 3.3: Regression variables for Matrix-Multiply on CPU

β	X
23.45	UOPS
194.45	NUMA
-2.45	CLOCK
...	...

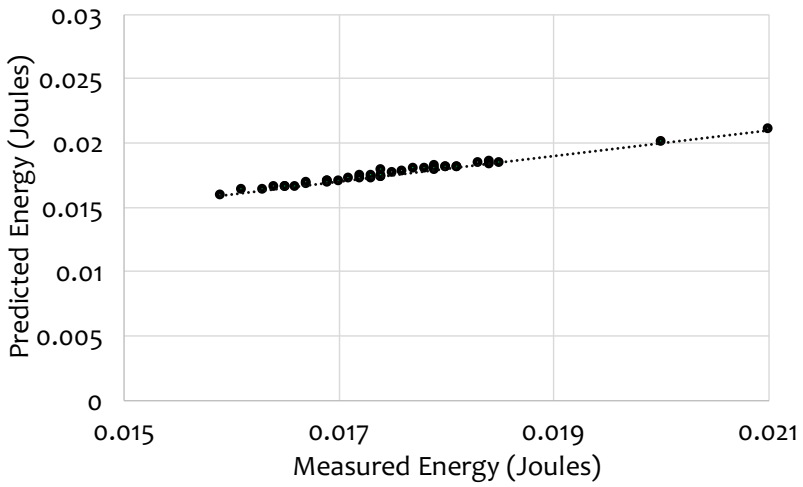


Figure 3.5: Scatter plot between measured and predicted energy values

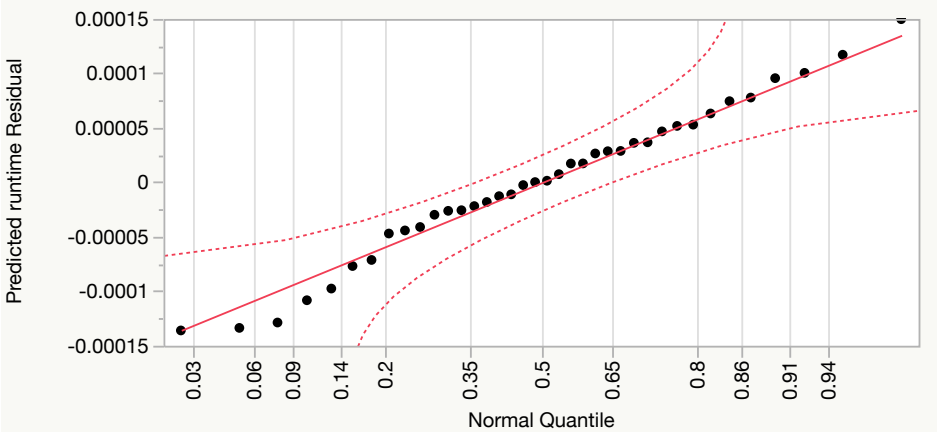


Figure 3.6: QQ-Plot for residuals of regression analysis

AEEM

We tested the same application, i.e., Matrix Multiply 1M×1M, using AEEM. We implemented three levels of energy estimation for an application, energy estimation of hardware (theoretical peak values of energy), total energy consumed by the application and the energy consumption by the individual kernels. Listing 3.2 shows the results of AEEM on the CPU for hardware energy estimation using theoretical performance, for the application at a node level, and for the application on a per-kernel basis. They are:

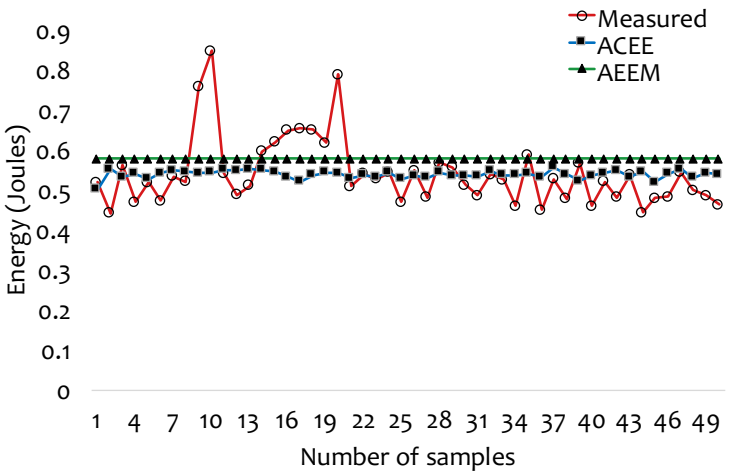
1. Hardware energy estimation using peak theoretical performances, i.e., energy taken for single precision, single precision/SIMD, single precision/SIMD/FMAD for Intel Xeon_-E5_2637 socket and NVidia_K20x.
2. Energy estimation of an application on core level.
3. Energy estimation of an application on node level per kernel basis

Listing 3.2: Sample Output from AEEM

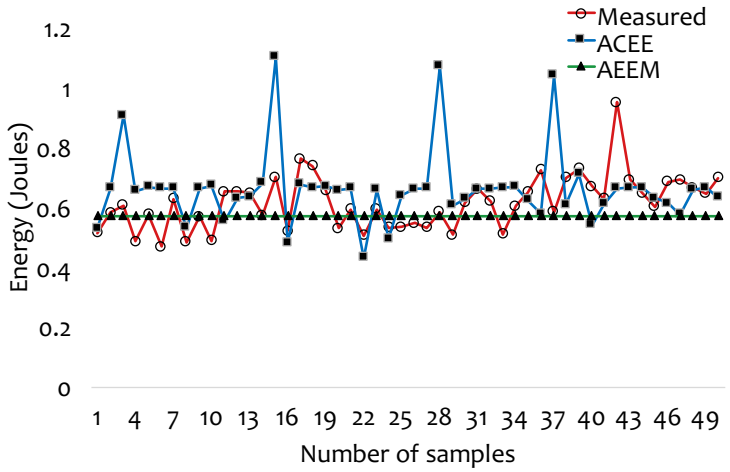
```
Theoretical peaks per socket:
    socket:intel_xeon_e5_2637
        single precision GFlops:130J,
            sp/SIMD GFlops:130J,
            peak dp GFlops:130J
            dp/SIMD GFlops:130J,
        peak memory bw:5J
Energy per core :
        Intel_Xeon_e5_2637 energy:25623.3J
Energy per Kernel:
    Kernel matmul
        ---socket:intel_xeon_e5_2637
            energy :5124.65J
```

3.6 Validation of AEEM

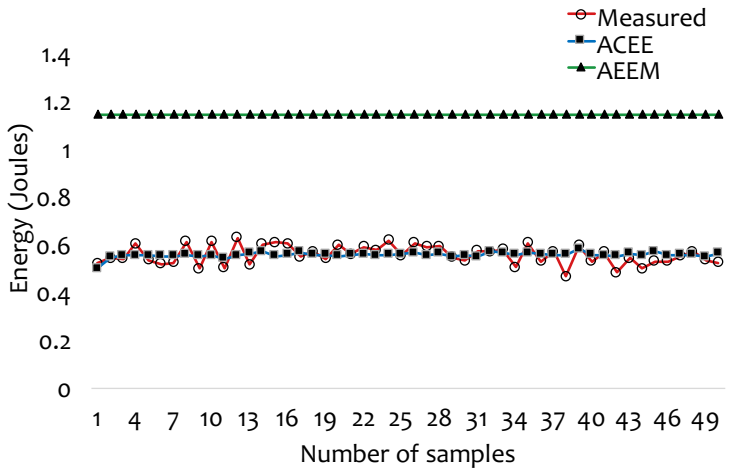
In this section we illustrate the advantages of using an automated tool such as Aspen to explore performance and energy trade-offs on current systems and to analyze the power requirements



(a) Small



(b) Medium



(c) Large

Figure 3.7: Comparison between measured, ACEE and AEEM for Fast Fourier Transform with varying input sizes, running on CPU

Table 3.4: Hardware parameters for Keeneland's processing elements and the local node's processing elements.

	Xeon X5660	Tesla M2090	Xeon E5-2620	Tesla K20c
Number of Cores	6	512	6	2496
Core Clock	2.8 (GHz)	1300 (Mhz)	2 (GHz)	706 (Mhz)
Peak GFlop/s SP/DP	134/67	1331/665	96/48	3500/1750
Memory Clock (Mhz)	1066 (DDR3)	3700 (GDDR5)	1066 (DDR3)	1300(GDDR5)
Memory Bandwith (GB/s)	25.5	177	25.5	208
P_{cpu}^{idle} (W)	10	30	10	29
$P_{cpu}^{dynamic}$ (W)	80	220	85	196
P_{memory}^{idle} (W)	0.5	0.5	0.5	0.5
$P_{memory}^{dynamic}$ (W)	4.5	1.5	4.5	1.5
$P_{nic}^{dynamic}$ (W)	9.69	9.69	-	-

of the applications. In the analysis presented below we use two computer systems, an abstract model of Keeneland [1] and a node of our local cluster at Virginia Tech, which consists of a dual-socket Intel Xeon E5-2620 CPU and an NVIDIA Tesla K20c GPU. Table 3.4 presents the detailed configurations of the machine.

3.6.1 Machine Model Evaluation

We have implemented a micro-benchmark that evaluates the energy draw for the processors as well as the memory subsystems for each type of processor on a heterogeneous cluster. For example, the energy efficiency for Flops (GFlops/Joules) of each of Keeneland's processing elements (an Intel-Xeon X5660 CPU and an NVIDIA Tesla M-2090 GPU) and of our local node's processing elements (an Intel Xeon E5-2620 CPU and an NVIDIA Tesla K20c GPU) is shown in Figure 3.8. Table 3.4 shows the details for these processors.

GPU performance (e.g., M2090 and K20c) is modeled using SIMD width[1] — a parameter in As-

[1]This approach has also been adopted by Dongarra et al. [126].

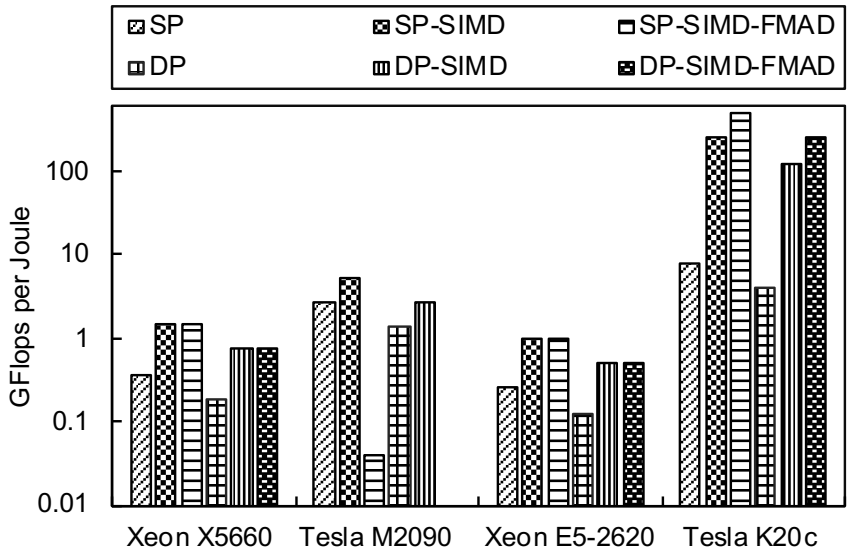


Figure 3.8: Achievable GFLOPS per Joule for a given type of floating point operation: SP (single precision), DP (double precision), SIMD and FMAD

pen, in the form of `number of cores` × `core clock` × 2 `floating point instructions per cycle (single precision)`. We observed that with Keeneland the GPU is in general, an order of magnitude more energy efficient than the CPU, particularly for SIMD operations. In our local node, this difference is approximately of two orders of magnitude. Figure 3.9 shows an evaluation for the memory subsystems of the two machines in terms of GBytes/Joules. In both machines, the GPU is able to offer significantly more Gbytes per Joule than the CPU, and the difference is more pronounced in our local node.

3.6.2 3D-FFT evaluation

In order to evaluate the accuracy of our analytical performance-energy model (AEEM), we examine the estimated energy for the 3D-FFT kernel on both types of processing elements (GPU and CPU) on our local node. For reference, we collect the measured energy obtained from running the FFTW 3.3.4 library for the CPU, and the CUFFT 7.5 for the GPU. The measured

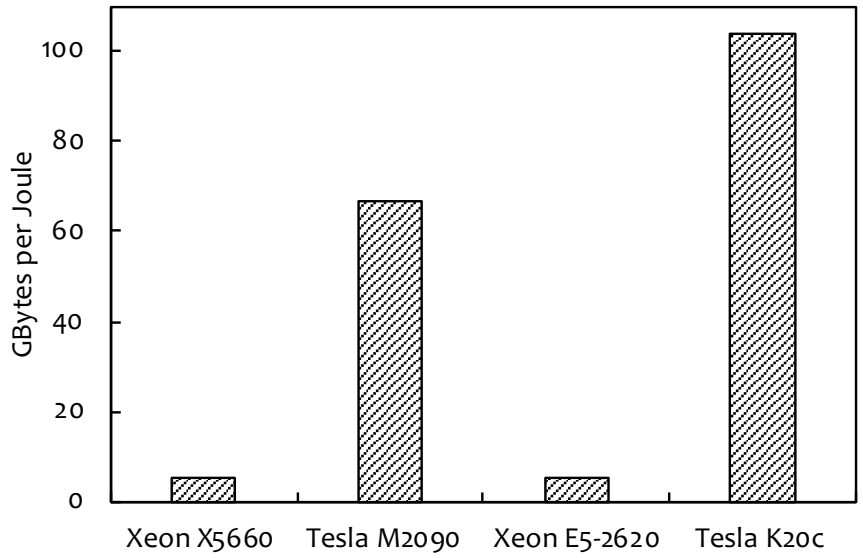


Figure 3.9: Achievable GBytes per Joule for a given type of floating point operation: SP (single precision), DP (double precision), SIMD and FMAD

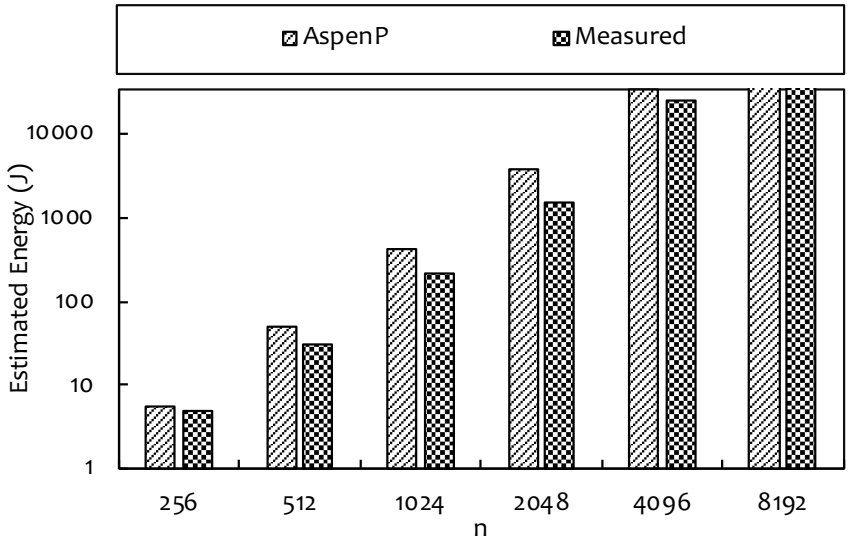


Figure 3.10: Estimated and measured energy for 1D-FFT kernel on CPU (Intel Xeon E5-2620)

energy for validation were obtained using the Running Average Power Limit (RAPL [70]) energy sensors for the the CPU, and the NVIDIA Management Library (NVML [2]) for the GPUs. Figures 3.10 and 3.11 show the result for the parallel execution of n^2 instances of 1-D FFT kernel for the CPU and GPU, respectively, as the problem size varies from $n = 256$ to $n = 8192$.

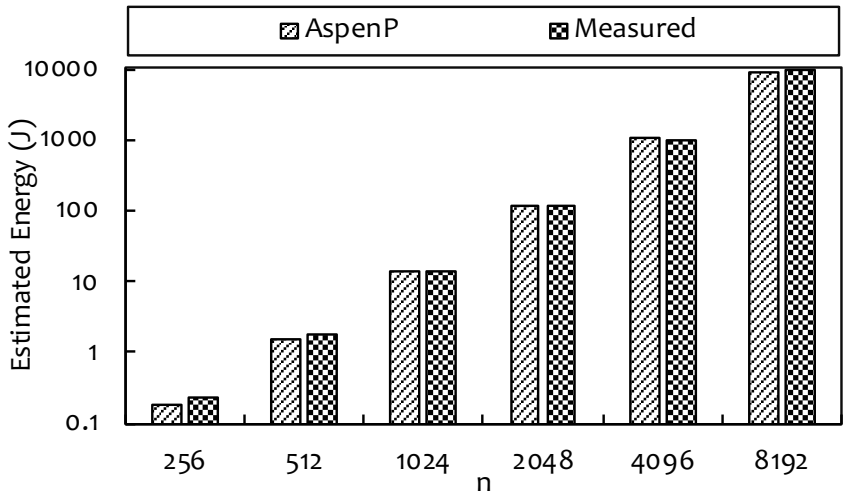


Figure 3.11: Estimated and measured energy for 1D-FFT kernel on GPU (Tesla K20c)

In both type of processing elements, we observe a high degree of overlap between the measured and the estimated energy, i.e., the proposed energy model is accurate enough to compare energy-performance tradeoffs.

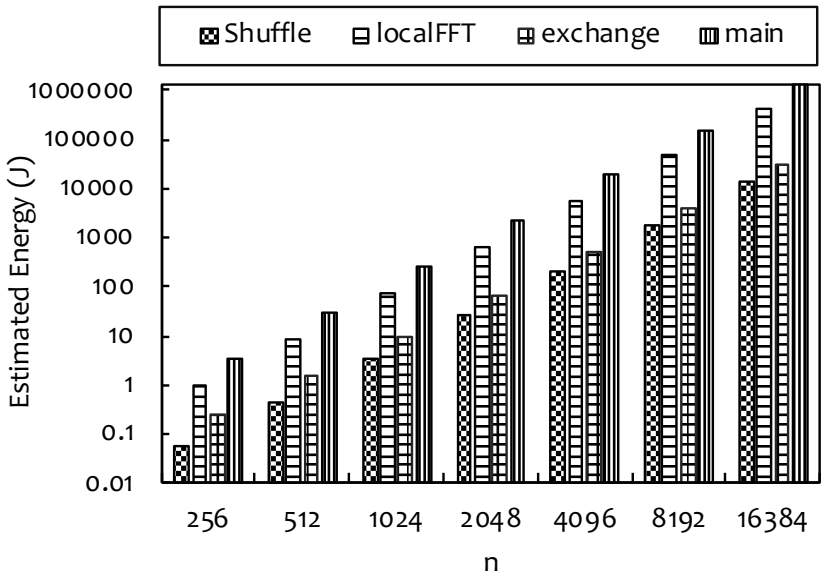


Figure 3.12: Estimated energy for 3D-FFT kernels on CPU (Intel-Xeon X-5660)

We now focus on the analysis of the 3D-FFT in the Keeneland machine, and provide some

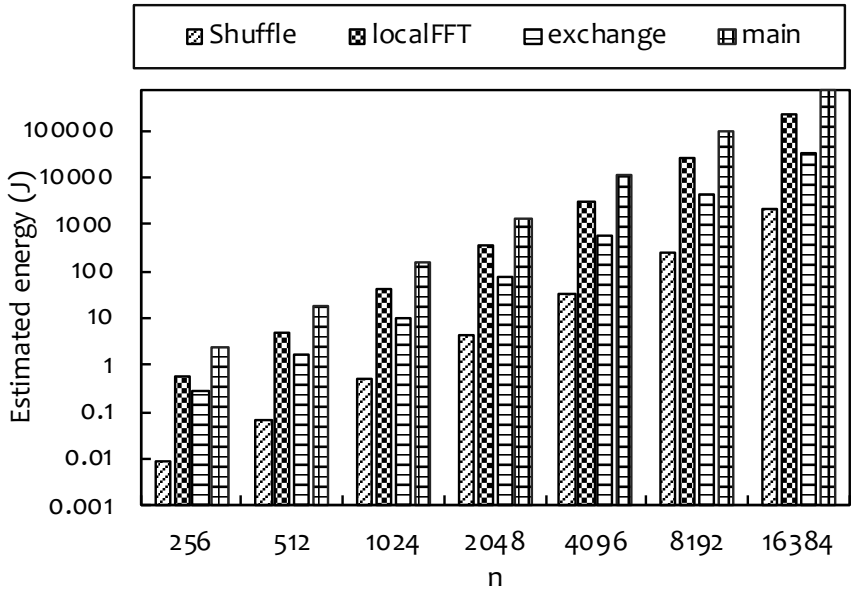


Figure 3.13: Estimated energy for 3D-FFT kernels on GPU (Tesla M2090)

results from our model using a combination of application and machine specific models for the pencil decomposition. In order to characterize the relative importance of each kernel on the energy consumed by the entire 3D-FFT application, we examine the estimated energy consumption on a kernel level on both processing elements in Keeneland. Figures 3.12 and 3.13 shows the result for the CPU and GPU, respectively. We observed following findings from Figures 3.12 and 3.13:

- Our analysis shows that communication dominates computation for small input sizes [38]. However, as the problem size increases the energy consumed by computation grows faster than the energy consumed by communication.

- Second, we observe that the overall application and kernel energy use is smaller in the GPU processing element. This result was expected, since SIMD operations are more efficiently processed in the GPU than in the CPU, as evidenced by our experiment in Section 3.6.1 and shown in Figure 3.8. In addition, the difference between the estimated

energy for the GPU and for the CPU is larger for the shuffle kernel. The reason for this is the GPU's provide a higher memory bandwidth per Joule, as evinced by our experiment, shown in Section 3.6.1.

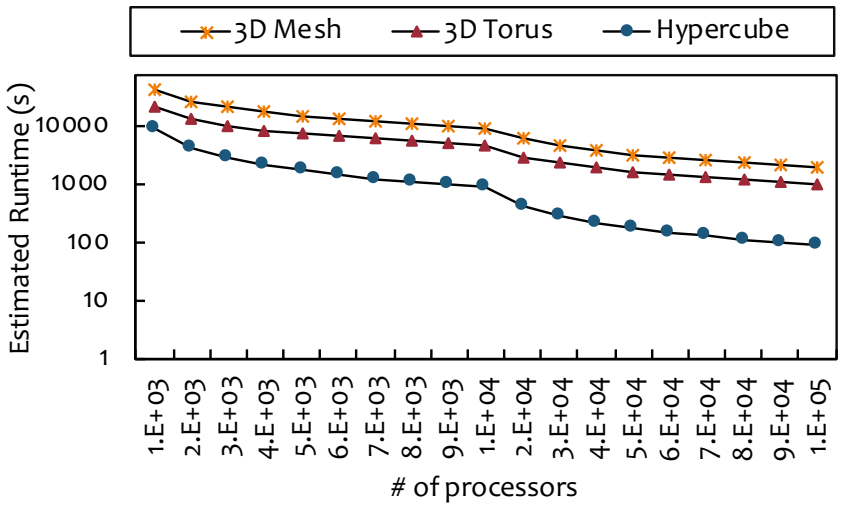


Figure 3.14: Estimated runtime of the exchange kernel on variant of the Keeneland machine using a QDR Infiniband interconnect

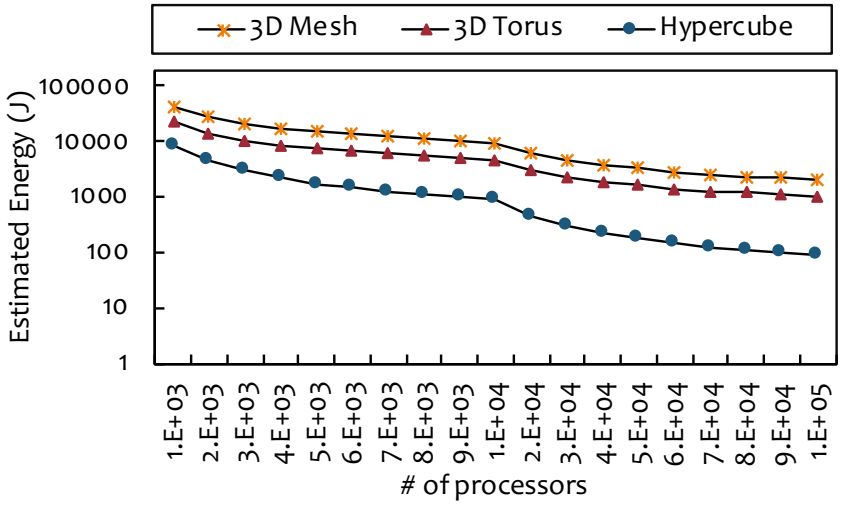


Figure 3.15: Estimated energy of the exchange kernel on variant of the Keeneland machine using a QDR Infiniband interconnect

Our final analysis of the 3D FFT model characterizes the behavior of the communication subsystem when executing the exchange kernel in a strong scaling scenario. Figures 3.14 and 3.15

estimate runtime and energy as the number of processors increases from one thousand to one hundred thousand processors in Keeneland. The figures show the results for a fixed problem size of $n = 2^{16}$ and 3 different types of interconnect topologies: hypercube, 3D torus and 3D mesh. The Aspen runtime expression for these topologies are illustrated in Table 3.5.

Table 3.5: Runtime expression for different interconnect topologies

Topology	Expression
3D torus	$T = \frac{msgsize}{2P^{2/3}.B_{link}}$
3D mesh	$T = \frac{msgsize}{P^{2/3}.B_{link}}$
hypercube	$T = \frac{msgsize}{P/2.B_{link}}$

As shown in Figure 3.14, as the number of processors increases, the estimated runtime decreases. This reduction is more accentuated in the hypercube topology, demonstrating agreement with previous work [38, 55, 147]. Figure 3.15 shows similar behavior for the predicted energy. This observation can be explained by our communication model, which does not account for the static power draw by the communication subsystem. Thus, increasing the number of nodes reduces overall energy.

3.7 Results and Analysis

Both of our models, ACEE and AEEM, enable energy estimation for heterogeneous architectures, and they provide a guideline to Aspen to enable execution on the energy efficient device as well as facilitate Aspen to optimize the application to stay under a energy budget. One of the main strengths of Aspen is to couple application abstraction with machine abstractions in order to estimate various traits of performance. By modeling application and device character-

istics and their corresponding energy budgets, we improve the accuracy, portability and speed of performance prediction and energy estimation.

To study the accuracy of our models in more detail, we explored five kernels and proxy applications (Matrix Multiply (MM), Fast Fourier Transform (FFT), Molecular Dynamics (MD), LULESH and Jacobi), each for three different input sizes. For each application, estimated energy results are presented for both CPU and GPU. Since ACEE requires access to real hardware in order to generate estimates, we used a cluster consisting of two physical CPUs, each consisting of an 8 core 64-bit Intel Sandy Bridge Xeon E5-2637 at 3.50 GHz. There are two GPUs on the system: an NVIDIA TESLA K20c with 5 GB GDDR5 memory and and an NVIDIA Quadro K600 with a 1GB GDDR3 memory. For AEEM we created an abstract machine model in Aspen that matches the specifications of this cluster. We present results for each application along with analysis and explanation.

3.7.1 FFT Results and Analysis on CPU

The results of Fast Fourier Transform for small, medium and large input sizes are shown in Figures 3.7(a), 3.7(b), 3.7(c). For each of the input sizes, ACEE closely follows the measured value, and it also follows the spikes observed in the measured values. For small and medium input sizes, AEEM (Aspens Embedded Energy Model) follows closely the measured and ACEE values, while for large input sizes, the difference between AEEM and measured, ACEE values is much greater. This happens because of the shortcoming of the analytical model of AEEM to capture the characteristics of large amount of communication involved between kernels of FFT and between FFT and hardware. On the other hand, the accuracy of ACEE follows closely for large input sizes too, since it takes into consideration performance counters corresponding to communication and memory accesses, e.g., memory bandwidth, memory data volume,

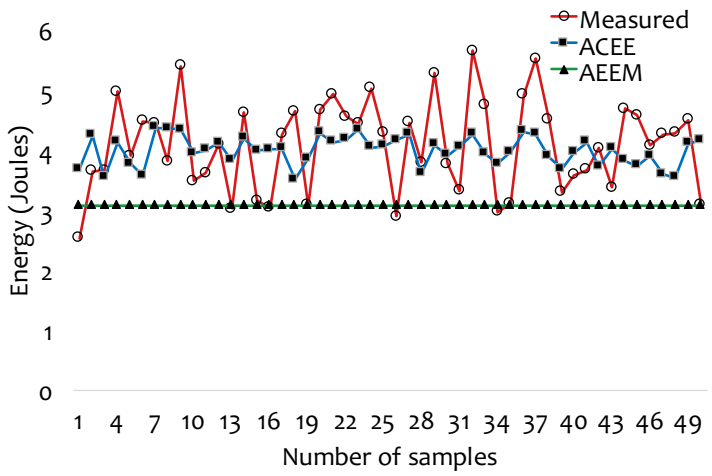
memory read latency, memory write latency, load to store ratio, etc., to capture the behavior of increased communication with larger input sizes. AEEM does not possess the ability to capture the characteristics of compile time optimization because of the inherent short-coming of analytical modeling techniques. For small and medium input sizes running on CPU, ACEE and AEEM can be used interchangeably because of minimal difference in accuracy, although AEEM provide faster results. But for larger input sizes for FFT, it is preferable to implement ACEE to capture the direct communication behavior and to enable more accuracy.

3.7.2 FFT Results and Analysis on GPU

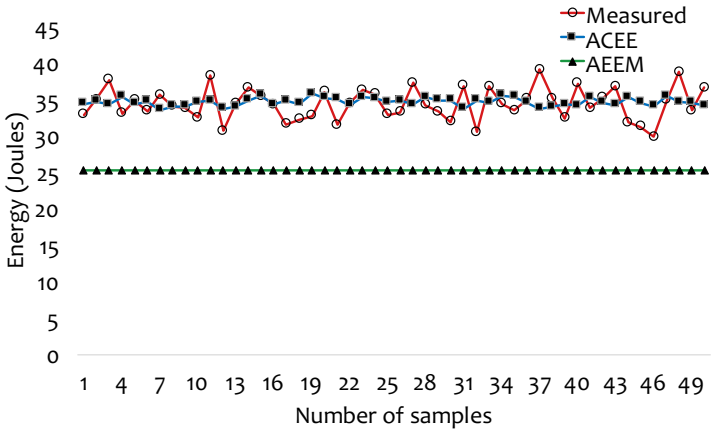
FFT small, medium and large input sizes are shown in Figures 3.16(a), 3.16(b), 3.16(c) respectively. Similar to FFT results on CPU, ACEE energy estimation closely matches with the measured values. We observe that AEEM is generally more optimistic in its performance estimates on the GPU FFT algorithm, resulting in under prediction at small and medium problem sizes, but a little over prediction for the large problem size. The under prediction in small and medium input sizes results because of inexact matching of the communication cost for smaller input sizes. The actual communication cost happens to be larger than predicted by AEEM for small and medium input sizes. For large input sizes, AEEM tends to over-predict as it does not have the ability to capture the behavior of reusing the data in GPU memory.

3.7.3 Matrix Multiply Results and Analysis on CPU

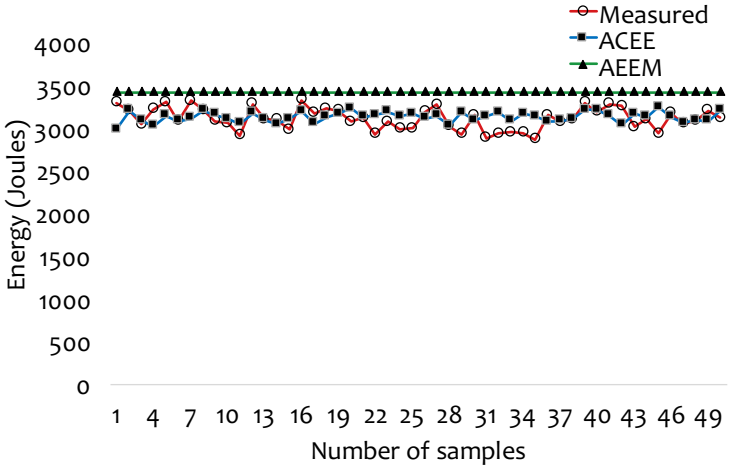
For Matrix Multiply [151] small, medium and large input sizes are shown in Figures 3.17(a), 3.17(b), 3.17(c). ACEE closely matches the measured energy values, while AEEM tends to overestimate the energy consumption pattern. We believe that running Matrix Multiply on newer architectures [84] generates some automatic compiler optimizations, which tend to reduce



(a) Small



(b) Medium



(c) Large

Figure 3.16: Comparison between measured, ACEE and AEEM for Fast Fourier Transform with varying input sizes, running on GPU

the energy consumption of the application and helps to improve the energy consumption. On the other hand, AEEM is unable to capture the optimization strategies and hence it tends to overestimate the behavior. On a general note, both ACEE and AEEM can be used to estimate energy, although the user can tradeoff between speed benefits provided by AEEM or advantages of accuracy provided by ACEE.

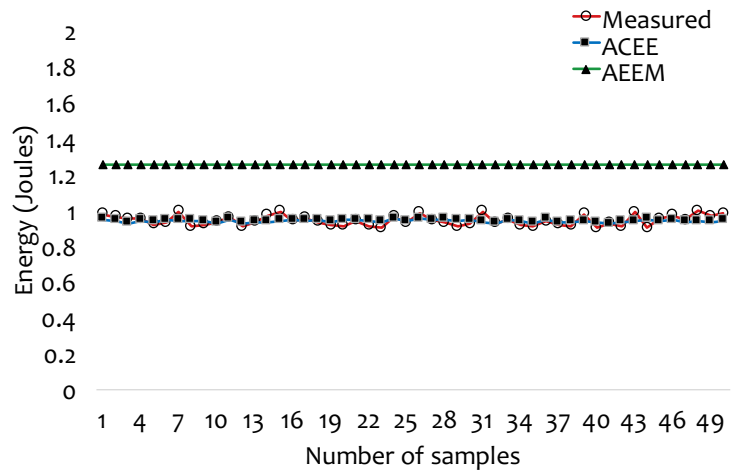
3.7.4 Matrix Multiply Results and Analysis on GPU

On the GPU, the results of ACEE and AEEM for Matrix Multiply small, medium and large input sizes are shown in Figures 3.18(a), 3.18(b), 3.18(c). ACEE and AEEM closely matches with the measured values, such that the choice of one modeling and estimation framework provides no advantage with respect to accuracy, although AEEM provides more timely results as compared to ACEE.

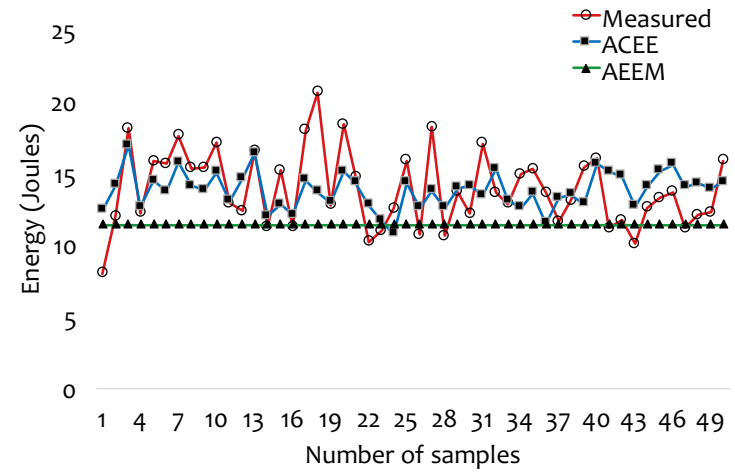
The reason ACEE and AEEM provides similar results as the measured values is because our non-blocked version of Matrix Multiply is memory intensive and the memory characteristics are captured by both ACEE and AEEM. ACEE uses all performance counters corresponding to memory bandwidth and AEEM captures the relationship between the memory bandwidth used and the peak memory bandwidth provided by the system. Moreover, the effect of compiler optimizations while running Matrix Multiply on GPU is minimal, hence ACEE and AEEM provides similar levels of accuracy.

3.7.5 Molecular Dynamics Results and Analysis on CPU

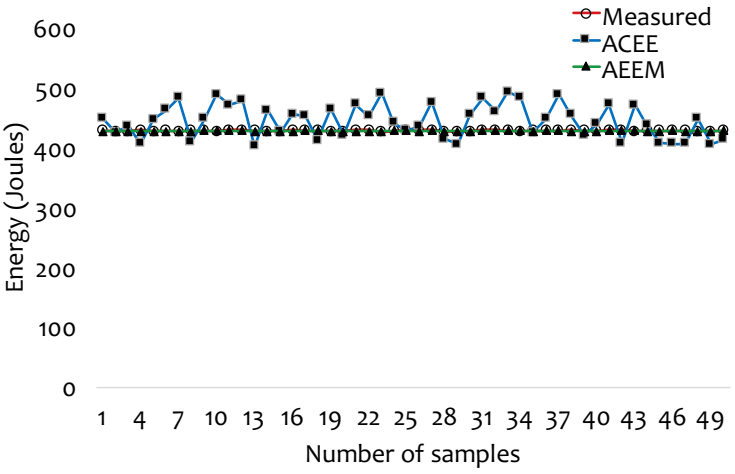
Molecular dynamics (MD) [93] results for running on CPU for small, medium and large input sizes are shown in Figures 3.19(a), 3.19(b), 3.19(c) respectively. For small, medium and large input sizes for MD running on CPU, AEEM tends to over-estimate the energy consumption as



(a) Small

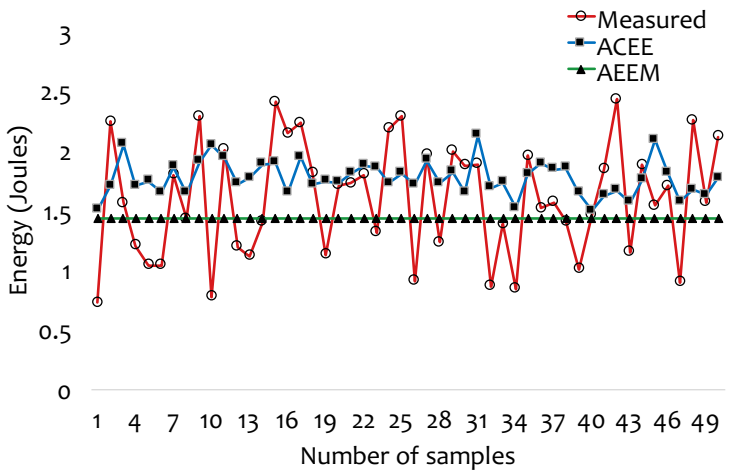


(b) Medium

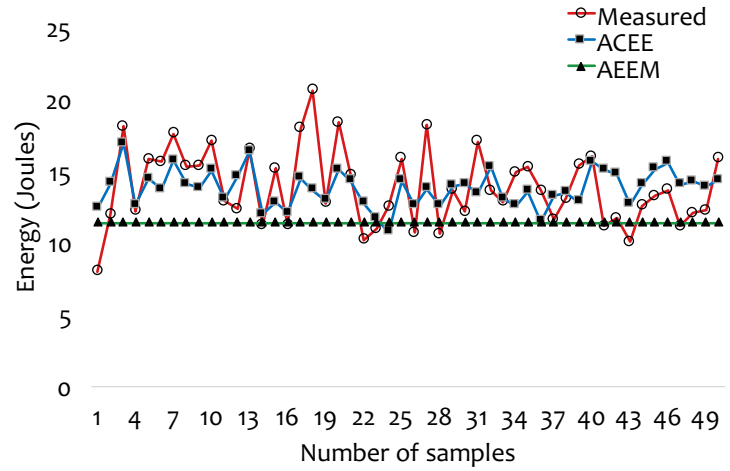


(c) Large

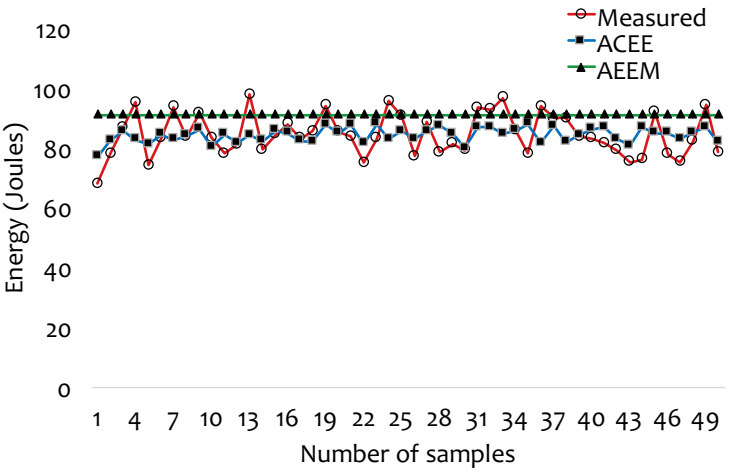
Figure 3.17: Comparison between measured, ACEE and AEEM for Matrix Multiply with varying input sizes, running on CPU



(a) Small



(b) Medium



(c) Large

Figure 3.18: Comparison between measured, ACEE and AEEM for Matrix Multiply with varying input sizes, running on GPU

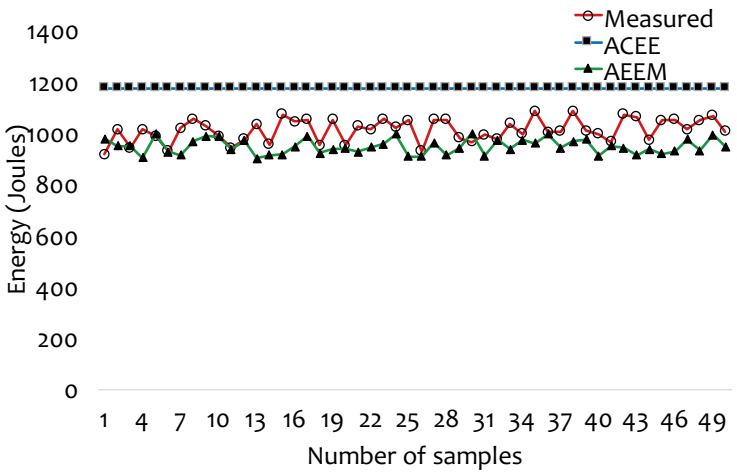
compared to ACEE. This happens because of over-estimation of computation time for the potential calculating kernel. On real hardware, compile time optimizations reduce the theoretical energy estimation of the kernel, which are not accounted in AEEM as it uses theoretical bounds to compute the energy estimation.

3.7.6 Molecular Dynamics Results and Analysis on GPU

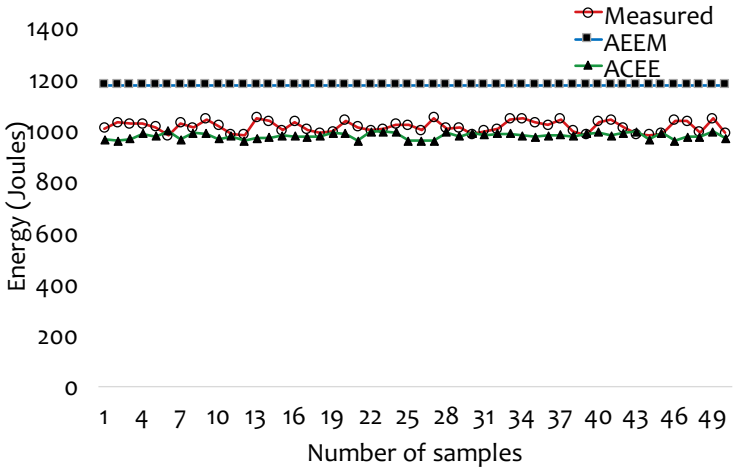
Molecular dynamics (MD) results for small, medium and large input sizes running on GPU are shown in Figures 3.20(a),3.20(b),3.20(c). For MD running on GPU, the AEEM results for small, medium and large inputs are closer in accuracy as compared to the measured results due to the small gap between theoretical and practical bounds on MD application kernel runtime. Moreover the default optimizations for MD show less effect in the case of GPU, hence similar accuracy for all input sizes. In the case of MD running on GPU, both ACEE and AEEM can be used to provide similar accuracy, though AEEM provides much faster results.

3.7.7 Jacobi Iteration Results and Analysis on CPU

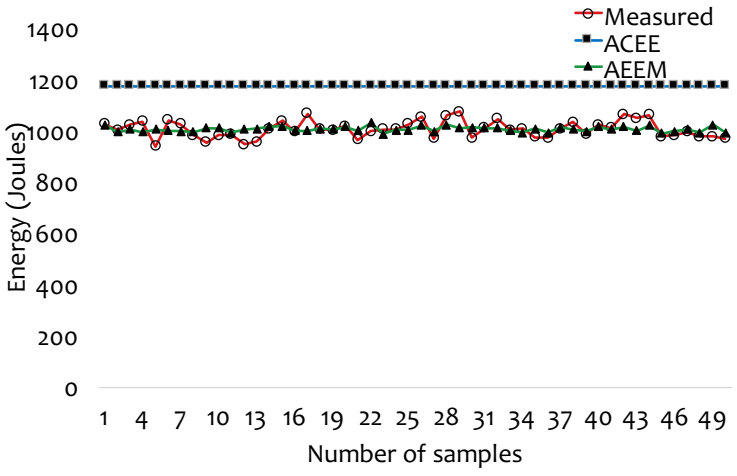
Jacobi iterative solver results for small, medium and large input sizes for CPU are shown in Figures 3.21(a), 3.21(b), 3.21(c). Jacobi iterative solver convergence rate is relatively slow as compared to other iterative solvers and it has parallelism inherent in the code. In case of small and medium input sizes, AEEM tends to over-estimate because of inability to capture auto-parallelism in the application, while for large input sizes, AEEM under-estimates.



(a) Small

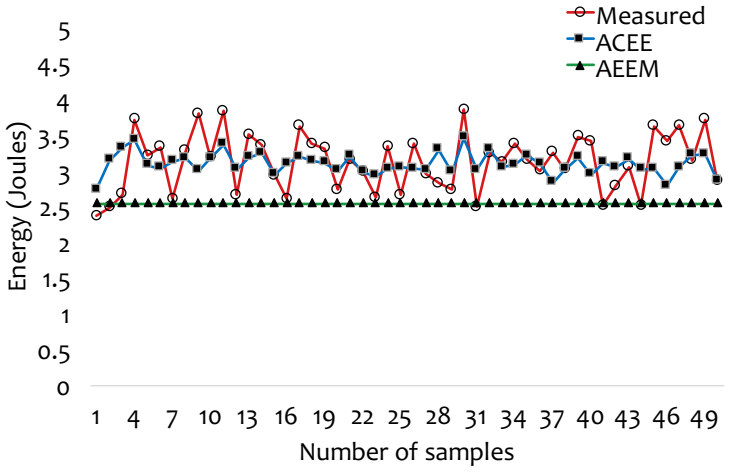


(b) Medium

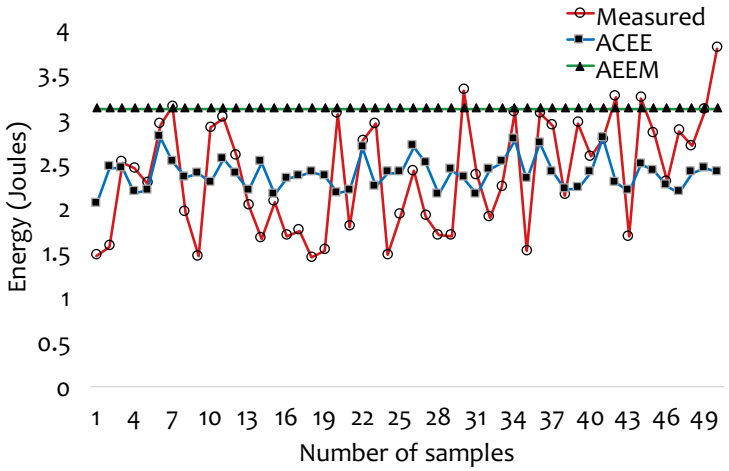


(c) Large

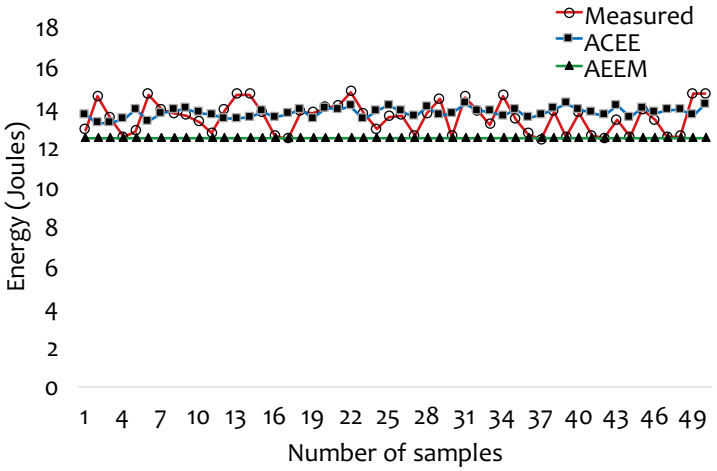
Figure 3.19: Comparison between measured, ACEE and AEEM for Molecular Dynamics with varying input sizes, running on CPU



(a) Small

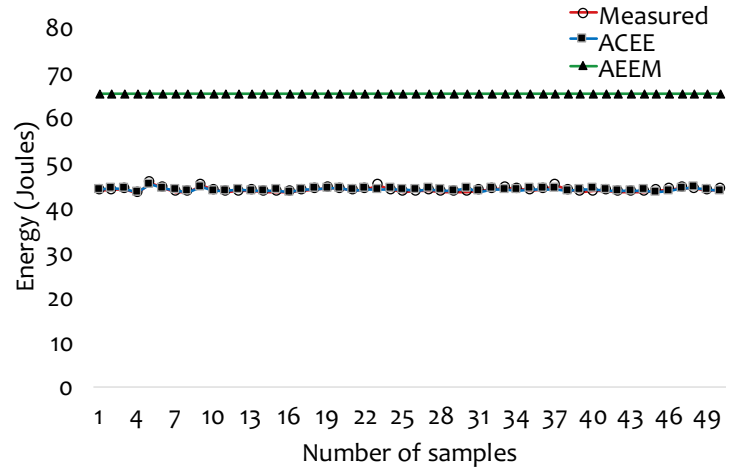


(b) Medium

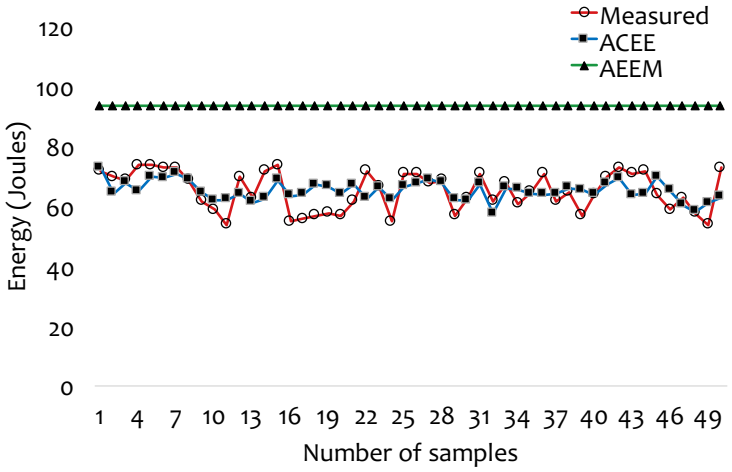


(c) Large

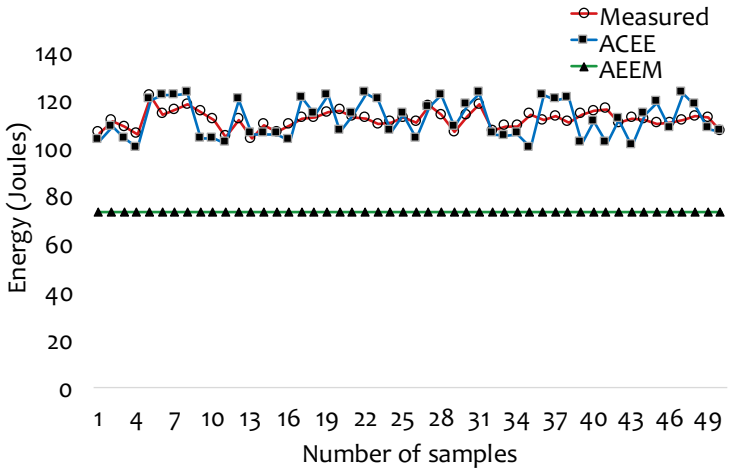
Figure 3.20: Comparison between measured, ACEE and AEEM for Molecular Dynamics with varying input sizes, running on GPU



(a) Small



(b) Medium



(c) Large

Figure 3.21: Comparison between measured, ACEE and AEEM for Jacobi with varying input sizes, running on CPU

3.7.8 Jacobi iteration Results and Analysis on GPU

Figures 3.22(a), 3.22(b), 3.22(c) represents the Jacobi results for small, medium and large input sizes. As can be seen in the figures, AEEM tends to underestimate for all input sizes. The convergence step in Jacobi iteration takes a lot of time and hence it causes an increase in energy too, which is accounted well in ACEE.

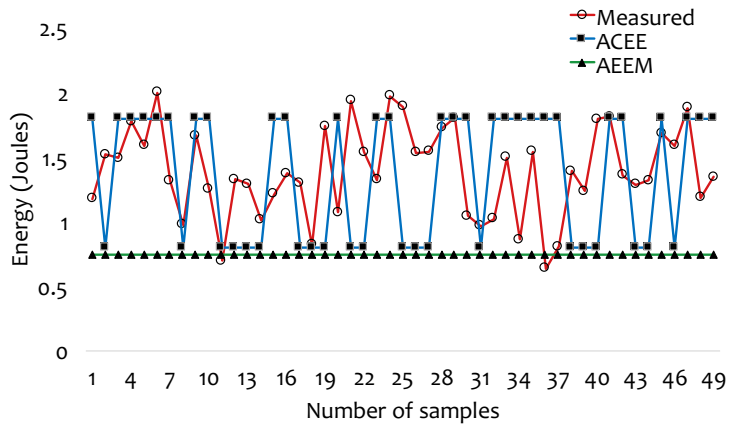
3.7.9 LULESH Results and Analysis on CPU

LULESH results for small medium and large are shown in Figures 3.23(a), 3.23(b), 3.23(c) respectively. A conventional LULESH application has approximately 42 kernels. The effects incured by individual kernels are taken into account in ACEE as the counters collected in ACEE tend to represent the behavior of entire application as well as individual kernels, and hence it closely matches the measured values. AEEM values are close in accuracy too for all input sizes, hence both ACEE and AEEM can be used to achieve comparable accuracy.

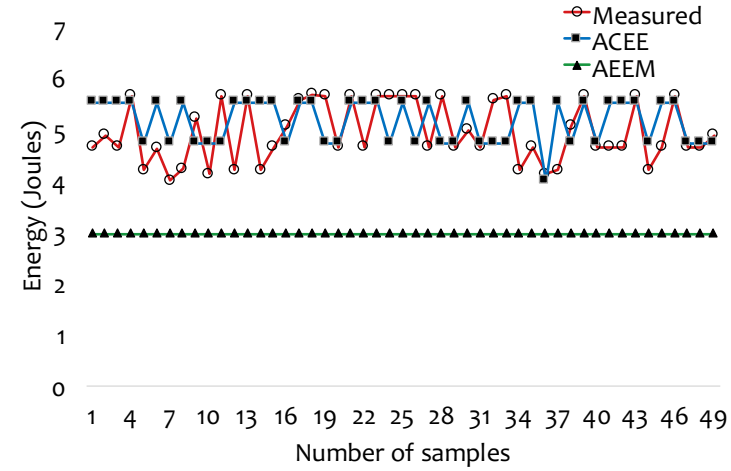
3.7.10 LULESH Results and Analysis on GPU

LULESH small, medium and large input sizes running on GPU are shown in Figures 3.24(a), 3.24(b), 3.24(c) respectively. AEEM accuracy gap is larger than any of previous applications running on GPU. This is because the compiler optimizations on each kernels produces a cumulative effect of a large amount of performance improvement in application execution, which is not accounted in AEEM, hence it is preferred to use ACEE in LULESH like applications which are very complex or have a large number of kernels in them.

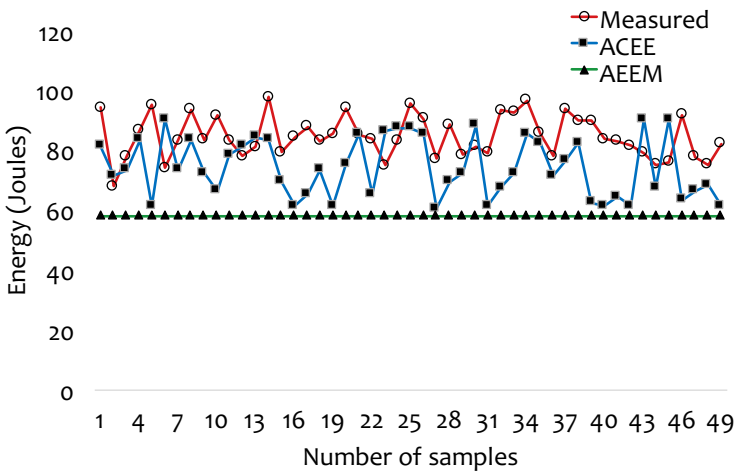
In conclusion, both ACEE and AEEM provide very good accuracy. In almost all cases, ACEE



(a) Small

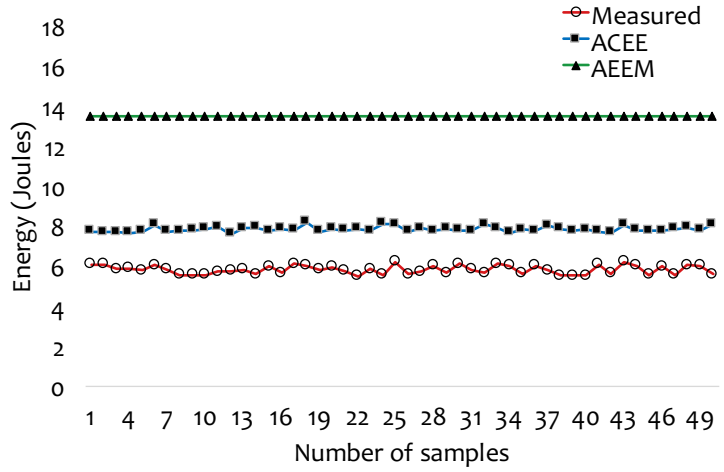


(b) Medium

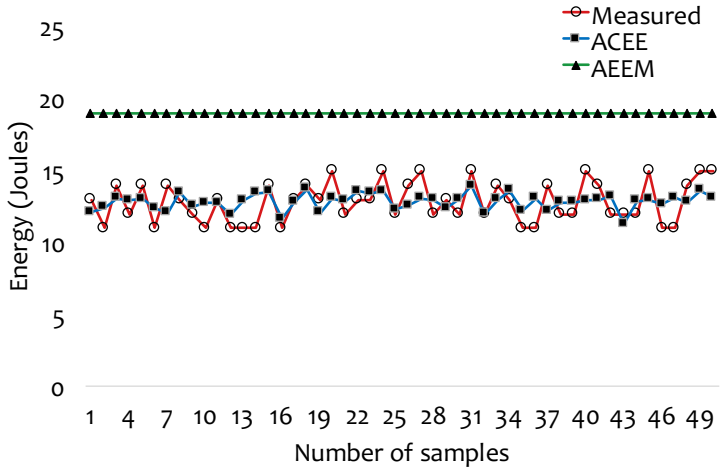


(c) Large

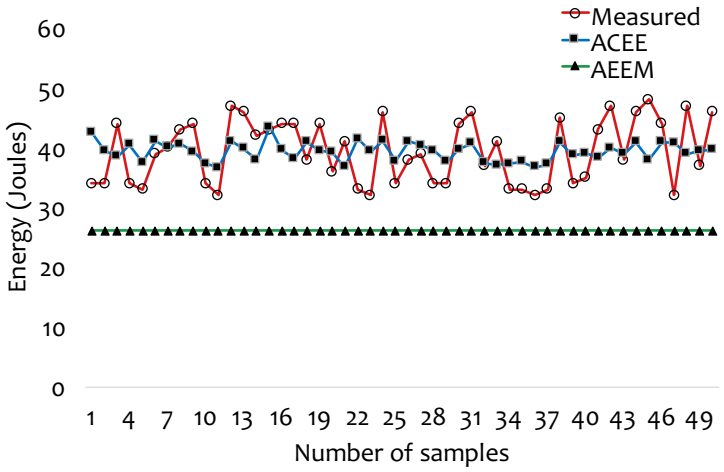
Figure 3.22: Comparison between measured, ACEE and AEEM for Jacobi with varying input sizes, running on GPU



(a) Small

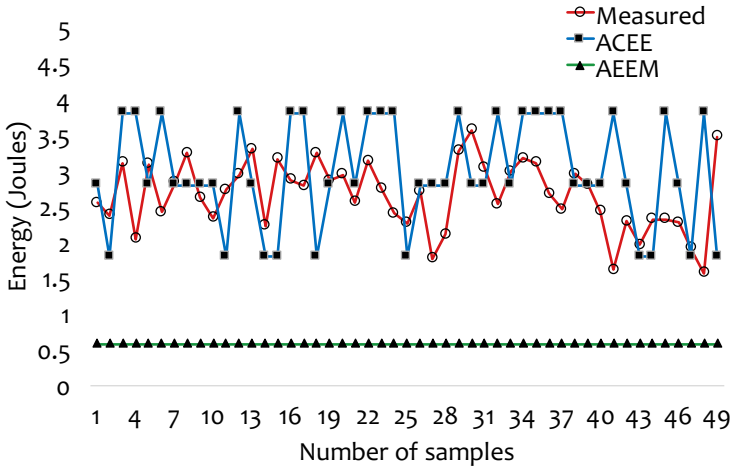


(b) Medium

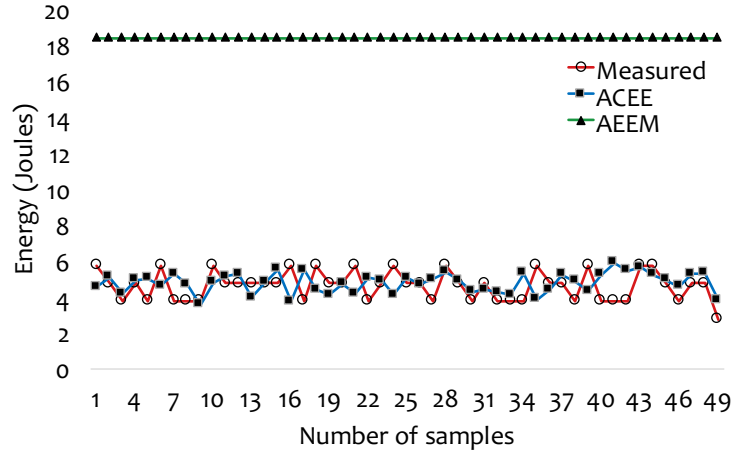


(c) Large

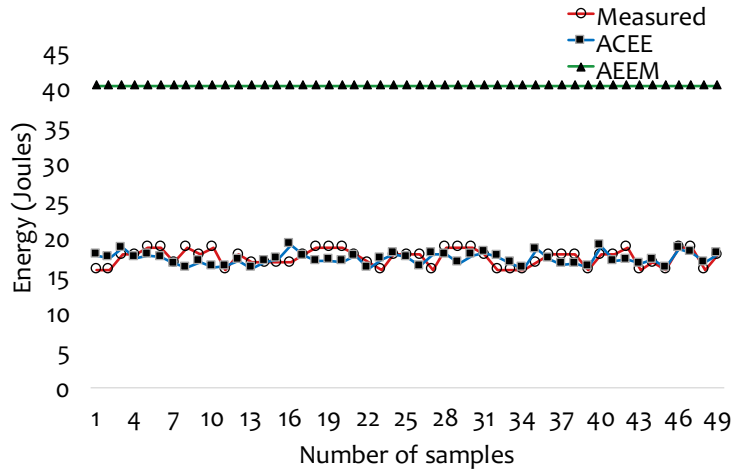
Figure 3.23: Comparison between measured, ACEE and AEEM for LULESH with varying input sizes, running on CPU



(a) Small



(b) Medium



(c) Large

Figure 3.24: Comparison between measured, ACEE and AEEM for LULESH with varying input sizes, running on GPU

accuracy is better since it uses application characteristics at runtime in order to better match the application execution time behavior, while AEEM requires less overhead.

3.8 Comparison and Contrast

Table 3.6: Benefits of the ACEE and AEEM methods.

Benefits	**ACEE**	**AEEM**
Portability	Very good	Good
Speed	Fast	Faster
Accuracy	Very good	Good
Ability to extrapolate	Yes	Not yet

Table 3.7: Drawbacks of the ACEE and AEEM methods.

Drawbacks	**ACEE**	**AEEM**
Programmers effort	Moderate	Minimal
Profiling	Heavy	Light/None
Access to hardware	Required	Minimal

Table 3.6 and 3.7 shows benefits and drawbacks of our two models, ACEE and AEEM. This table serves as a guideline to the user of which model is best with respect to a given execution scenario.

ACEE does not depend upon Aspen to provide estimation of energy, so it has less dependencies than AEEM. AEEM requires programming environment of Aspen to function.
Both ACEE and AEEM are fast, although AEEM provides faster as it uses only modeling information to generate estimations and does not require any pre-processing steps.
As shown in the previous section, the accuracy of ACEE is better as it takes into account the runtime and compiler optimizations applied by the system. ACEE results can be extrapolated for increased input size or increased number of cores, but extrapolation beyond the domain of the regression modeling comes with the usual caveat. The same does not hold for AEEM; it

can quickly generate new estimations. Moreover, AEEM also has the ability to generate models for the hardware beyond the current scenario.

Scalability is handled equally well by ACEE and AEEM as shown in next section (§3.9), where we explore the scalability of Matrix Multiply for increased input sizes. In terms of drawbacks, for ACEE, applications have to be profiled to enable collection of performance counters. This is a one-time cost for developers, but AEEM does not require any programmer effort beyond a corresponding application model in Aspen.

ACEE also requires access to real hardware in order to provide estimates. ACEE has an initial setup cost, it takes some time to generate the regression model. Such initial setup costs do not exist in AEEM. For ACEE, we expect the estimation bucket (a collection of profiled results for re-use) will amortize this cost.

3.9 Use Cases

The ability to estimate energy use for applications is helpful in many ways for the research community, and the choice of estimation techniques ultimately depends upon the user's goals. In this section, we present some common and important use cases that revolve around issues of energy overuse and can be addressed using our techniques.

3.9.1 Choosing the most energy efficient device

One common use of energy estimation models is to determine which device is the most energy efficient for a given scenario. Both ACEE and AEEM have the ability to generate analysis regarding which device is better with respect to energy, power or time efficiency. Table 3.8 presents an estimate for energy for our three applications (MM [126], FFT [38] and MD) using

ACEE and AEEM. The % represents the energy benefit of the more efficient device.

Table 3.8: Results of applying ACEE and AEEM to find energy estimation, shows the devices chosen and the percentage improvement in energy consumption as compared to slower device

Application	**ACEE**	**AEEM**
Mat-Mul	14.53%-GPU	6.91%-GPU
FFT	0.046%-GPU	0.039%-GPU
CoMD	0.19%-GPU	0.14%-GPU

3.9.2 Locate regions of code consuming more energy

In this use case, we investigate a scenario involving multiple kernels in an application where the programmer wants to explore the patterns of energy consumption for the application source code. Table 3.9 provides a breakdown of energy consumed in terms of percentage with varying input sizes for CPU and GPU for FFT using AEEM (a similar trend can be shown for ACEE as well). We see in this breakdown each of the kernels, i.e., shuffle, localFFT, main and exchange for different input sizes. For FFT, we see a small trend towards greater time in the *exchange(message communication kernel)* phase with increasing problem size, a similar energy consumption trend exists across all problem sizes for the communication kernel. This result is specific to FFT, but it elaborates how similar trends can be investigated in other applications.

Table 3.9: Percentage of energy consumed by each kernel for 3D-FFT.

FFT		**% of each kernel**			
Input size		**Shuffle**	**Local FFT**	**Main**	**Exchange**
CPU	2048	1.27%	36.02%	55.71%	6.97%
	4096	1.17%	35.99%	55.83%	6.99%
	8192	1.08%	35.97%	55.93%	7.03%
GPU	2048	15.13%	60.47%	21.22%	3.16%
	4096	14.63%	59.70%	22.37%	3.28%
	8192	14.18%	58.96%	23.44%	3.40%

3.9.3 Trigger callbacks in applications

This use case represents the evaluation of a performance model at runtime; it is particularly useful in situations when the user wants to be informed of an anomaly, such as at what problem size, the power consumption exceeds expectations. The performance model can be developed for the energy estimation or any of the performance counters listed in Tables 3.1 and 3.2. For example, Figure 3.25 shows the predicted energy usage with increasing input sizes. If a user likes, as these predictions are fast, they can repeatedly call the predictor and trigger a callback to the application if their energy budget is exceeded to increase during application execution. Figure 3.25 also shows the scalability of our model with increasing input sizes, comparing predictions from ACEE and AEEM with the measured values using (RAPL [70] and NVML [2]). The minor changes in energy usage in Figure 3.25 being tracked by our predictors (and ACEE in particular) show that our model behaves accordingly with changing application execution patterns.

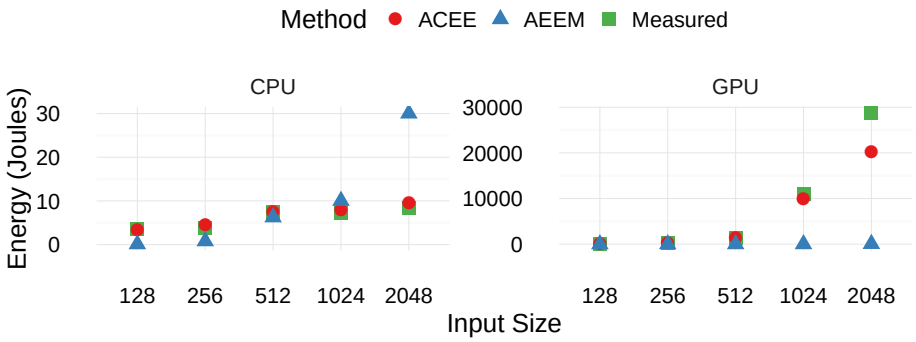


Figure 3.25: Effect of estimating energy for Matrix Multiply with increasing input sizes for CPU and GPU

3.10 Conclusion

In this chapter, we presented two modeling frameworks that leverage Aspen for energy measurement and estimation. One approach, *ACEE*, uses hardware performance counters to generate analytical and empirical modeling techniques to provide accurate energy estimation, while *AEEM* uses Aspen's existing performance modeling information in order to generate energy estimations. Both *ACEE* and *AEEM* attempt to provide reasonably accurate results without the overhead of detailed architectural simulation. Our approaches use application and device specific information, i.e., either hardware performance counters (ACEE) or abstract performance models (AEEM), in order to generate estimations for energy. Our models serve as a guideline to the user about the energy consumption patterns exhibited by the application. We shaped our frameworks to support Aspen DSL in order to guide the user about the most energy efficient device present in the platform. Moreover, our models are compatible with Aspen without imposing any significant overheads. We find that this combination provides an accurate, fast, portable and reliable energy estimation framework.

We tested our frameworks on five kernels and proxy applications – matrix multiplication, FFT and MD, Jacobi iteration, and LULESH – and provided the estimates for power, energy and time and most energy efficient devices corresponding to multiple example scenarios.

In the next two chapters, we will explain our contributions towards design space exploration techniques for performance and energy efficiency. In chapter 4, we present Prometheus, which is a composable hardware-software optimization framework. Prometheus, uses a combination of analytical and machine learning modeling techniques to capture application characteristics, and uses those characteristics to determine the sensitivity of application performance and energy improvement towards reconfigurable architectures.

Chapter 4

Modeling Framework Enabling Co-design for Exascale Systems

4.1 Introduction

Conventional optimizations for high-performance computing have been focused on software tuning (e.g., compile [10,28,94] and runtime optimizations [36,161]). However, there is growing evidence of performance gains when hardware configurations and software approaches are optimized in unison [30, 34, 48, 89, 122]. Determining coherent hardware and software optimizations is challenging, partly because of the increase in the search space (i.e., permutations of tunable parameters); brute force search of optimal hardware configurations and software tuning quickly becomes impractical [89].

We anticipate that this challenge will become much more difficult with the increase in scale for Exascale computing and the expectation that there will be an increase in reconfigurable hardware parameters for emerging architectures [111] — such as CPU and memory frequency,

number of cores.

A number of techniques have been proposed that are geared towards understanding application behavior so that the findings may be applied to achieve near-optimal performance of applications. These methods include analytical modeling [33, 115, 144], simulators [20, 82, 110], emulators [107], and co-design modeling techniques [114, 157].

In the context of Exascale computing, it is paramount that methods predicting application behavior are accurate, scalable, portable, involve minimal overhead, and are easy to use [44]. While the aforementioned methods offer promising results towards the goals defined by the authors, most approaches do not take into account all the Exascale computing goals listed above. Although the co-design modeling methods take several goals into account, the lack of automation and limited scalability leave room for improvement.

Towards this end, we propose *Prometheus*, a composable hardware-software optimization framework. As part of Prometheus, we develop a combination of analytical and machine-learning techniques to capture application characteristics. These application characteristics are subsequently used to determine the hardware-software configuration for near-optimal performance and energy improvements.

Prometheus uses the Aspen [147] domain-specific language for abstract application and machine representation and fast exploration of the optimization search space. We leverage Aspen's ability to represent application and machine behavior and develop an automated optimization-search-space exploration framework around Aspen.

We evaluate Prometheus for its efficacy using four widely used proxy applications: LULESH, CoMD, CG and CoEVP [47, 87, 122, 127]. We demonstrate that Prometheus identifies near-optimal hardware-software configurations and verify the results via brute-force search of the design space.

We present the following contributions in this chapter:

1. Analytical and empirical modeling techniques that capture application characteristics, highlighting the sensitivity of performance and energy to hardware parameters;

2. An automation framework that uses Aspen to represent abstract application and machine models, and thereby programmatically explore the tuning search space for the near-optimal performance gains and energy efficiency; and

3. Evaluation of Prometheus' efficacy on a local cluster as well as NERSC's Cori and discussion of relevant concerns using four proxy applications.

4.2 Automatic application and machine model generator in Aspen

Aspen [147], because of its inherent composable and modular nature, provides the benefit of exploring and improving existing architectures and applications – with the added feature of automaticity, i.e., automatically generating application and machine models can boost the usability of Aspen to a new level.

While this automation may seem simplistic, it helps to solve numerous issues faced by the research community, such as improving current architecture and designing exascale software and architecture with maximum performance and minimum energy consumption, fusing multiple memories together to achieve maximum bandwidth, implementing resilience methods on memory hierarchies. The hardware/software co-design techniques presented in this chapter are also expected to be portable, simple, accurate and robust, and they also must possess the ability to change application specific configurational changes in hardware/software at runtime.

The initial phase of our approach consists of automatic generation of abstract application and machine models for Aspen to introduce automation in the domain specific language.

The block diagram of automatic generation of the abstract application model is shown in Figure 4.1, where an input program analyzer takes source code, analyzes application characteristics such as FLOPS, loads, stores, iterations, complexity, etc. and generates Aspen IR (Intermediate Representation), which is adapted from Compass [97]. Then it is consumed by an Aspen IR post processor and produces an Aspen application model.

For abstract machine models, Figure 4.1, a machine specification extractor uses a combination of kernel and user level functions to wring out machine specifications such as sockets, cores, nodes, memory type, cache, cache hierarchy, PCIe latency, memory bandwidth, etc. Using the extracted machine parameters, we generate the Aspen IR, which is consumed by the Aspen machine IR post processor and yields an Aspen abstract machine model. The automated framework provides many advantages, such as no requirement for hardware while testing applications. It is portable and extendable to current and future architectures, requires no programmers effort, has low overhead, and inherits accuracy from Aspen as previously established in Aspen [147].

We use this framework of automatic application and machine model generation to include automaticity in Prometheus. Before deploying the automatic application and machine model at the core of the Prometheus framework, we test the efficacy of the automatic model generation by testing for sensitivity to changing CPU and GPU hardware configurations in Aspen. Figures 4.2 & 4.3 show the sensitivity of LULESH and CoMD's runtime with various CPU and GPU configurations in Aspen's framework environment. A *what-if analysis* of LULESH and CoMD is shown in Figure 4.4. LULESH because of its inherent complex compute and memory intensive nature, and a complicated combination of almost 42 kernels is not largely effected by a variety of hardware configuration changes. Therefore, LULESH's performance will improve

from application and kernel-specific optimizations e.g., loop fusion (Figure 4.4). CoMD, on the other hand, shows runtime improvement with improving communication, memory properties and system clock properties. Therefore, memory throttling, utilizing high-bandwidth interconnect technology (e.g., dragonFly) and increasing system clock will help to improve CoMD performance.

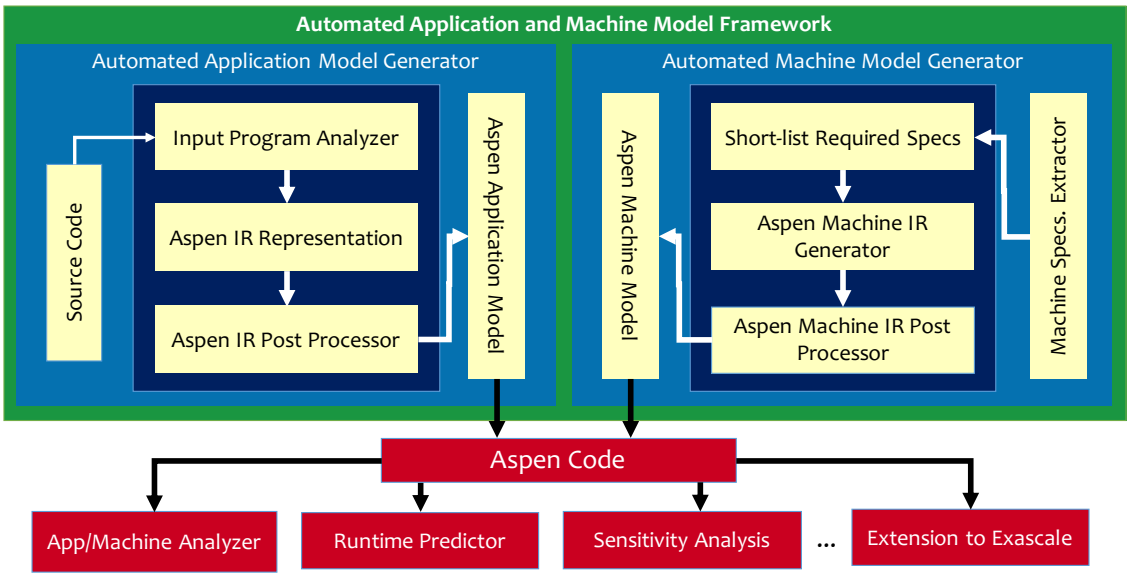


Figure 4.1: Automatic application and machine model generation in Aspen

4.3 Design and Methodology

Prometheus' high-level overview and workflow are shown in Figures 4.5. A clarification between the vanilla Aspen and Prometheus is shown in Figure 4.6. For clarity, we call the extended Aspen as *Aspen+*, which is a combination of Vanilla Aspen and all the extensions (automatic application model generator, automatic machine model generator, automatic machine model configurator) that we added in the domain specific language.

We use Aspen to represent application and machine models that allow us to explore the variation in configurations and design space.

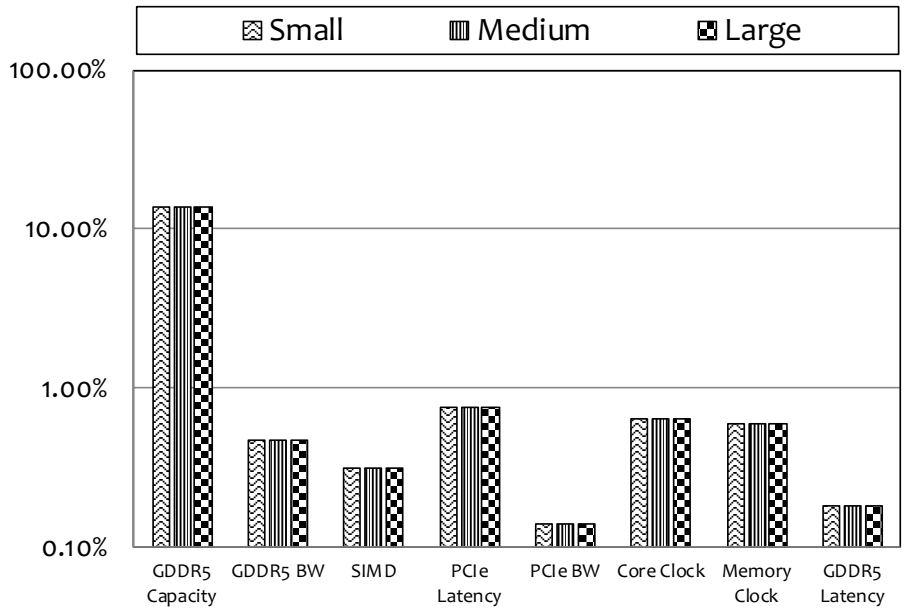


Figure 4.2: Sensitivity analysis of LULESH's runtime with varying CPU and GPU hardware configurations

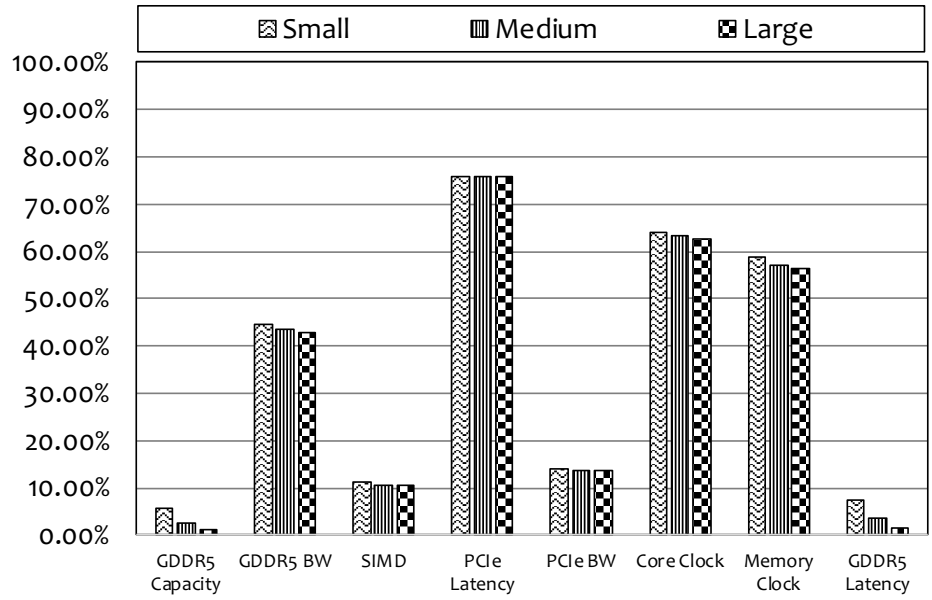


Figure 4.3: Sensitivity analysis of CoMD's runtime with varying CPU and GPU hardware configurations

Though Prometheus uses the same level of abstraction and grammatical semantics as Aspen, but the major difference lies in the level of automation and the added feature of Aspen to explore the design space of hardware. Figure 4.7 shows the representation of application and

What-If Analysis			
Application	**Dominant hardware parameter?**	**Why?**	**Improvements?**
Lulesh	Not significantly impacted by change in hardware	Tendency to not change because of different properties incurred by ~40 kernels	Application specific improvements e.g., loop fusion, array contraction etc.
CoMD	PCIe Latency, Core Clock, Memory Clock	Memory and communication intensive	DVFS, memory throttling, Application specific optimizations etc.

Figure 4.4: What-if analysis of LULESH and COMD performance improvement with varying configurations.

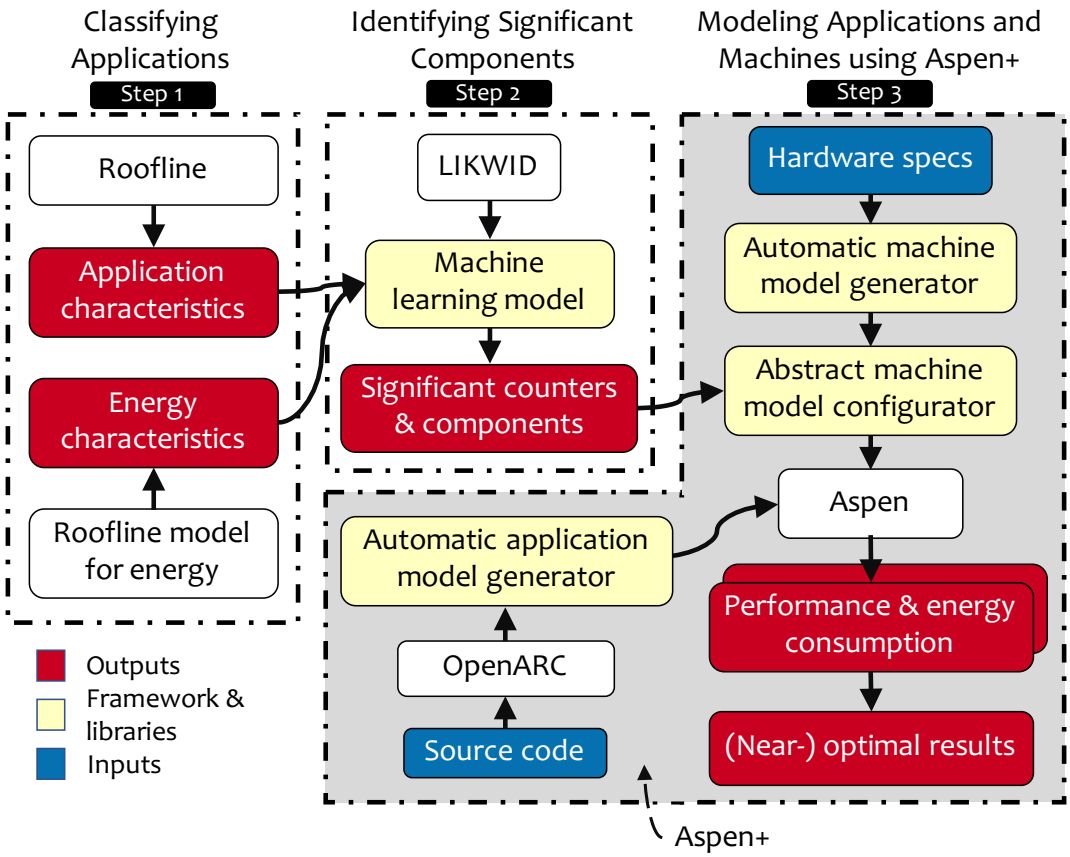


Figure 4.5: Workflow of Prometheus

hardware in various hierarchies, and the difference between Aspen and Prometheus in terms of automation level.

As the current high-performance computing architectures provide limited options to explore

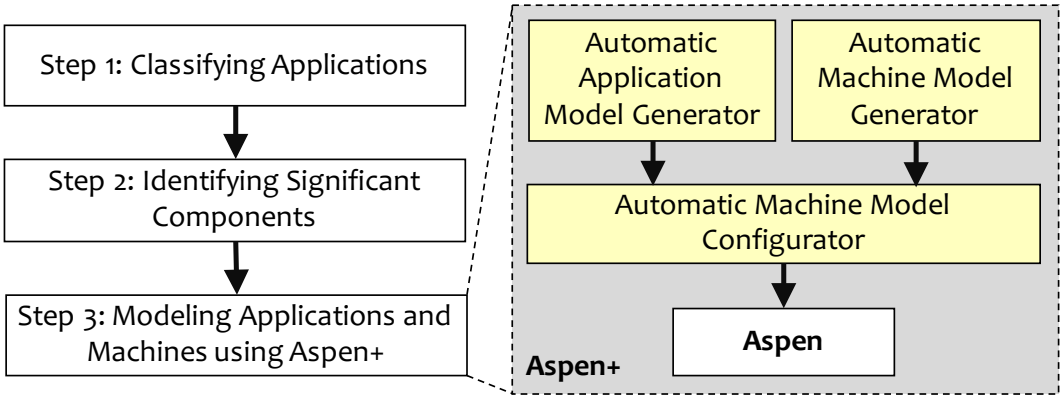


Figure 4.6: Delineation between Aspen+ and Aspen

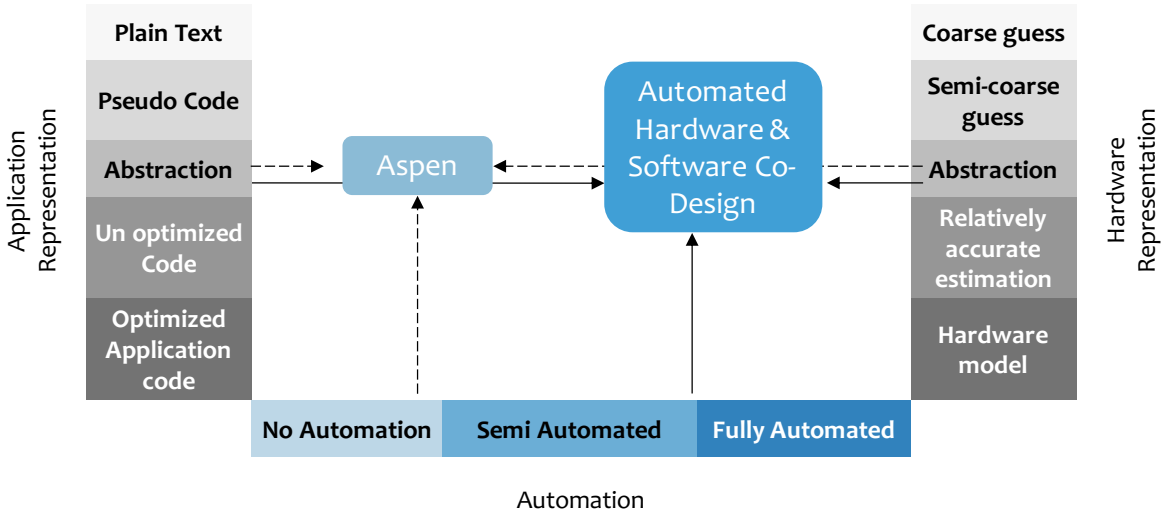


Figure 4.7: Difference of automation between Aspen and Prometheus

reconfigurations of hardware components (e.g., varying clock and memory frequency), therefore we develop a framework that provides us with the ability to experiment with a wide range of significant hardware components in an abstract manner and analyze the outcomes on performance.

As a first step of the workflow, Prometheus classifies applications (e.g., compute, memory, or I/O bound). Based on the application characteristics, Prometheus determines the hardware components that affect the application performance and energy consumption by analyzing the hardware performance counters. Among the hardware components that influence performance

and energy consumption, Prometheus selects the top contributors and analyzes the effects of varying their configurations.

We extend Aspen, in the form of *Aspen+* (see holistic view in Figure 4.6). The extensions enable:

- Automatic generation of abstract application and machine models using Aspen. (The automation is in contrast to the manual implementation of abstractions §4.3.3).
- Automatic reconfiguration of hardware components for exploration of tuning and performance evaluation (§4.3.3).
- Modular and composable implementation of Aspen by integration of performance and energy measurement and modeling methods (in the form of Apsen+ §4.3.3).

The workflow of Prometheus involves three major steps, as shown in Figure 4.5. These steps are:
A) Application classification,
B) Identifying and exploring the significance of hardware components using supervised machine learning algorithm, and
C) Automatic application and machine modeling using Aspen+.
We describe these steps and components below.

4.3.1 Application Classification (Step-1)

Application classification (both in terms of performance and energy consumption) helps to classify the applications in terms of their inherent characteristics (e.g., compute bound, memory bound). Such high level classification serves as a guideline for the *supervised machine-learning algorithm* (§4.3.2).

There are a number of techniques that can be used to determine applications' execution pattern. We use Roofline method [173] to identify *performance* characteristics of the application and to develop application signatures. We relate the performance and operational intensity of the application with the platform's peak performance and memory bandwidth. The operational intensity is a metric that relates to the number of operations performed per memory access.

We use the Roofline model of energy [31] to capture the holistic *energy* characteristics. We capture the cost of performing an operation both in terms of time and energy consumption. Here we use the relationship of energy per FLOPs and energy per byte in addition to the operational intensity exhibited by the application.

This preliminary analysis of the application serves as a guideline for determining the class of application (i.e., whether the application is compute, memory, or I/O intensive) and therefore enables us to observe the relevant set of performance counters.

4.3.2 Exploring Significance of Hardware Components (Step-2)

We develop a machine-learning model by identifying the hardware components that play a significant role in the application's execution. We identify the significant hardware components using performance counter collection and analysis.

A summary of the steps involved in exploring the significance of the hardware components are:

1. Exploring the activity of hardware components using performance counters.
2. Short-listing significant hardware counters using supervised machine learning.
3. Mapping of significant counters to corresponding hardware components using machine learning based classification techniques.

The steps involved in finding the significance of hardware components are explained below in order of execution.

Metrics and Counters

To identify the significant hardware components we consider the performance counters relevant to the class of application identified in step 4.3.1. In this work, we use LIKWID [154] to collect performance counters. LIKWID is an open source tool that enables collection of both core and un-core events of an application's run. It also enables measurement of power and energy consumption using machine-specific registers (MSRs). In Prometheus we use preset counter groups including `UOPS`, `UOPS_RETIRED`, `UNCORECLOCK`, `UOPS_EXEC`, `FLOPS_DP`, `CYCLE_ACTIVITY`, etc,. The implementation includes scripts that automatically collect and marshal the measurements from the application's execution. The sampling is configured to gather enough measurements to satisfy a 95% confidence interval [105].

Generation of Supervised Machine Learning Model

The next step is to determine the significance of each collected counter group with respect to runtime and energy consumption. We use a supervised regression based machine-learning algorithm to find significant counters. Several conditions need to be fulfilled to enable efficient machine learning based modeling approaches — e.g., satisfying confidence interval criteria, avoiding model over and under-fitting, choosing an appropriate model for the data set, explaining the outliers etc,. After fulfilling these conditions, we generate supervised machine learning models for runtime and energy consumption using equation 4.1, where the outer summation indicates repetition of measurement for statistical rigor and to gather median and quartile statistics.

$$F(X) = \begin{cases} \sum_{i=1}^{n} \sum_{j=1}^{m} \beta_{ij} X_{ij} + \alpha_{ij} & \text{where } \beta_{ij} \neq 0 \\ \sum_{i=1}^{n} \sum_{j=1}^{m} \beta_{ij} Z_{ij} + \alpha_{ij} & \text{Otherwise} \end{cases} \tag{4.1}$$

where β is the performance counter coefficient, X is performance counter type, α is the intercept, $F(X)$ is estimated energy value, n is iteration count/number of samples of performance counters, m is the number of hardware counters and Z are the performance counters whose value is never zero even when system is idle, e.g., clock frequency.

Mapping of Significant Counters to Hardware Components

After shortlisting significant counters, we map them to their corresponding hardware components using classification analysis. We use a supervised machine-learning algorithm — fuzzy c-means clustering algorithm [19] — for classification of performance counters. We classify counters based upon their characteristics/type.

Note that there may be scenarios where counters correspond to multiple hardware components, for example, `L3_to_MEM_BW` and `L3_to_MEM_Data_Volume` correspond to L3 caches as well as DRAM bandwidth, and hence our justification for use of the fuzzy c-means clustering algorithm [19]. Fuzzy c-means differs from the K-means clustering algorithm as the clusters can overlap resulting in belonging to multiple groups simultaneously(more mathematical details can be found in FCM [19]). After a mapping of significant counters to hardware components is generated, we pass this information to Aspen. Figure 4.8 shows a simplified FCM for the clustering of significant counters to their respective hardware components.

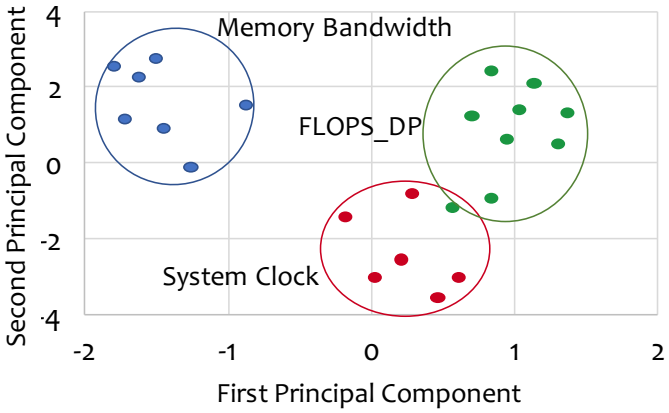


Figure 4.8: An illustration showing fuzzy c-mean clustering of performance counters and their mapping to hardware characteristics

4.3.3 Application and Machine Modeling (Step 3)

After we have identified all significant hardware components, we prune the design space by varying configurations of significant hardware components using Aspen's modeling framework.

Automatic Application and Machine Model Generation using Compass in Aspen+

In order to execute an application in Aspen's environment, any application and hardware needs to be formatted in Aspen's specific grammar by abstracting the characteristics of application and hardware. We generate application and machine model automatically, as described in the previous section.

Varying Hardware Configurations in Aspen+: Abstract Machine Model Configurator

Once we have automatically generated the abstract machine model, we can make hardware configurational changes using the Abstract Machine Model Configurator (AMMC). Using AMMC, we change the configuration of significant hardware components identified as an outcome of the classification tree in the previous step. AMMC changes the configuration of the hardware,

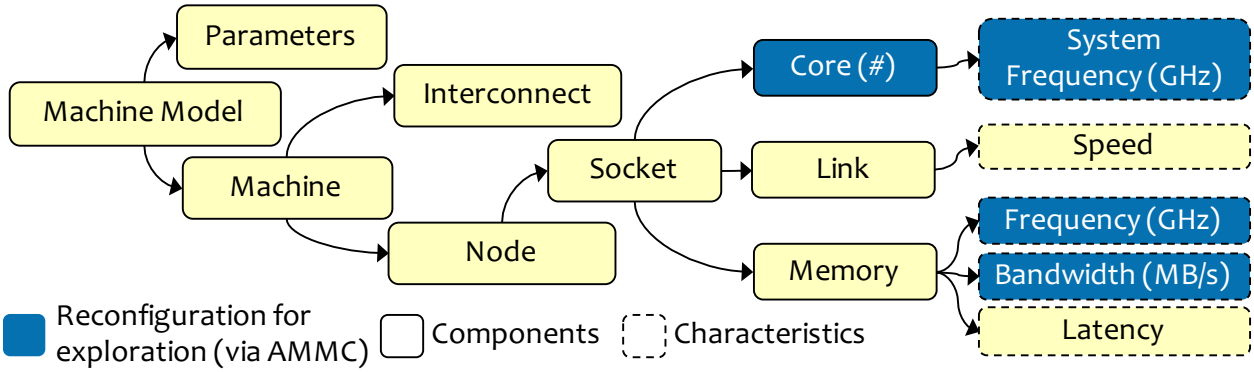


Figure 4.9: Highlight of the components and their characteristics tuned via the Automatic Machine Model Configurator (AMMC).

as shown in Figure 4.9 using the manufacturer specification as a baseline. Figure 4.9 shows the hierarchy of the machine and the blue rectangles shows the components that were tweaked automatically by the automatic machine model configurator for a given problem. AMMC changes the configuration between minimum and maximum range using either of the following two ways:

- Using *system specification*, for example, to find the reconfigurable range of clock frequency, we can use Linux functions. One of the outcomes is shown in Listing 4.1. Our experimental system has the capability to change system frequency between 1200 MHz to 3800 MHz.

- Using *statistical methods* to include the manufacturer's provided values and manual variations to explore a range. Doing so ensures that the minimum and maximum values are within the manufacturer specifications.

Listing 4.1: Tuning CPU frequency within a range.

```
Analyzing CPU 0:
  driver: intel_pstate
  ...
```

hardware limits: 1.20 GHz — 3.80 GHz

...

Using the significant hardware components identified as part of the above process we change configuration of each hardware component in isolation as well as in combinations of two hardware components.

Calculating Runtime and Energy Consumption for Application Execution via Aspen

As we consider runtime and energy consumption as performance metrics, we use the original runtime modeling framework proposed by Aspen. We measure the energy consumed by the application using the methodology proposed in Chapter(§3).

Runtime Calculation in Aspen The runtime calculating code in Aspen takes an application model and a machine model as input. Aspen generates a symbolic expression for how long the application will take to run for a given machine, and then substitutes all of the hardware and application parameters and subsequently evaluates the expression to calculate the runtime using a throughput-based analytical model. More details can be found in Aspen by Spafford et al. [147].

Energy Consumption Calculation in Aspen The energy consumption calculation and evaluation is based on Umar et al. and Gamel et al. [56, 156], which proposes activity factors to determine the energy consumption in the system. Gamell et al., [56] divide total energy required by a kernel E_i^{system} into energy consumed in communicating messages E_i^{comm} and total systems energy E_i^{system}– where E_i^{system} is composed of energy consumed by processor and memory.

Combining the energy consumed throughout the system, i.e., the processor, memory and communication components in Aspen, provides us the overall energy consumed by the application execution.

Determining the Best Hardware Configurations for Runtime and Energy Consumption

After following the Prometheus workflow, we test the application with the configurations of all significant hardware components identified by machine-learning model in Aspen framework. All significant hardware configurations are changed in isolation and in combination of two hardware components, until we cover all combinations. An overall summary mentioning the hardware component that results in highest sensitivity towards runtime and energy is provided at the completion of Prometheus framework.

4.4 Results and Analysis

One strength of the Prometheus framework is its ability to determine the near-optimal software-hardware configuration using automatic modeling techniques. Modeling the behavior of the application on varying hardware configurations and using this information to optimize the performance design space makes it a framework of choice to explore the design space for current and future architectures.

To study, develop and evaluate Prometheus, we developed an abstract machine model consisting of Intel's Haswell processor in Aspen. Each node in the abstraction contains an 8-core 64-bit Intel's Haswell processors with 32-GB of system memory. Nodes are interconnected by an InfiniBand network. The abstract machine model implements a multi-core processor with SIMD capability, out-of-order execution and latency hiding capabilities. Aspen+ has the abil-

ity to reconfigure hardware components including number of nodes. The results for varying system clock are validated on a local cluster with the same hardware configurations as the abstract machine model in Aspen+, while the scalability results are validated on the NERSC Cori system [9] and presented and analyzed in (§4.5).

4.4.1 Proxy Applications

We analyzed four proxy applications in this chapter. These applications are summarized below:

LULESH is a proxy application that explores the performance of parallel static unstructured mesh applications [87]. It solves the hydrodynamics equation by separating the spatial problem space, using the concept of hexahedral mesh generations. Using this concept, it separates the problem design space into sets of elements and uses unstructured data structures in order to fulfill the complex geometric construct required by the application.

CoMD is a simulation code [122] that calculates the potential among elements. A few significant factors dictating the performance of CoMD are setting up a distance, calculating the potential within that distance, and maintaining a specific number of atoms within a cell under observation. The data are represented in three dimensions. A number of optimization strategies have been proposed to improve performance, including the division of the problem area into multiple small areas and making a working group of threads focused on each group.

CoEVP is a simulation framework that models the mis-proportion of a titanium cylinder when it is fired at a wall [47]. The code contains many fine-grained models that implement scaling techniques. It generates the mis-proportion of the cylinder in both fine and coarse grained manners to present a Lulesh finite element and viscoplasticity. It implements a type of empirical modeling technique for every new simulation, the profiled results are consulted before starting the new experiments.

NPB-CG is an application that computes smallest eigen values of a sparse matrix, where the matrix is always unstructured [127]. It verifies the computations and communication performed on an unstructured grid with a random matrix. CG is a widely used application to test unstructured grids, to calculate their computation and communication pattern, and to verify the calculated values.

4.4.2 Analysis and Discussion

In this section, we study four proxy applications and present the results of varying hardware configurations in terms of sensitivity analysis and scalability testing on single and multi-node settings. *Sensitivity* is calculated using variations in the output parameter corresponding to a change in input parameter. We use normalized sensitivity in our experiments, where we present sensitivity results with respect to runtime and energy improvement. For each application, we will answer the following research questions:

- We investigate the sensitivity analysis of all the significant hardware components (shortlisted by machine learning algorithm in Prometheus) for runtime and energy for four proxy applications. We provide an explanation for sensitivity analysis with respect to application characteristics. We analyze the effect of varying various architectural configurations e.g., system clock, memory clock, number of cores, memory bandwidth etc., – where memory bandwidth is calculated using a combination of memory clock, data rate, memory bus width and number of interfaces.

- We present scalability (strong and weak) results for the hardware configuration that provide the most sensitivity towards runtime and energy improvement.

- We validate the results with current high-performance computing systems for the hardware components that can be reconfigured on today's architectures – i.e., system clock

and number of cores (memory frequency can be changed in BIOS in some recent HPC clusters, but our experimental system lacks this feature).

- We validate the scalability results with the NERSC Cori system.
- We map the Prometheus investigation along the lines of the proposed exascale architectures and the individual hardware components abstractions [11, 44, 157].

Case Study-1: LULESH

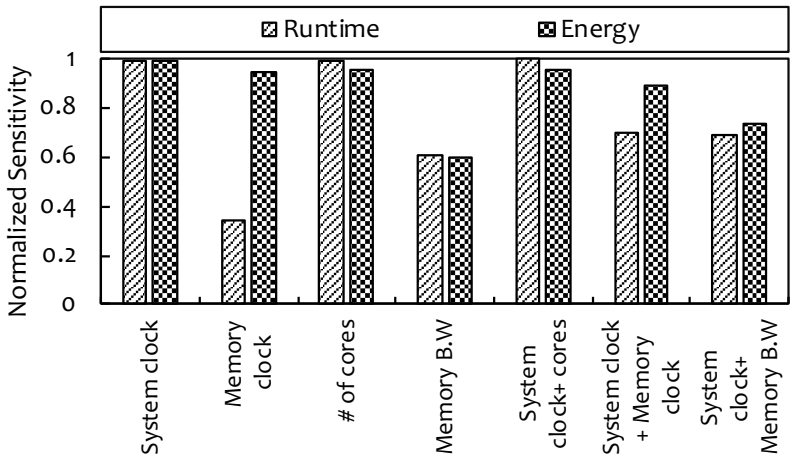
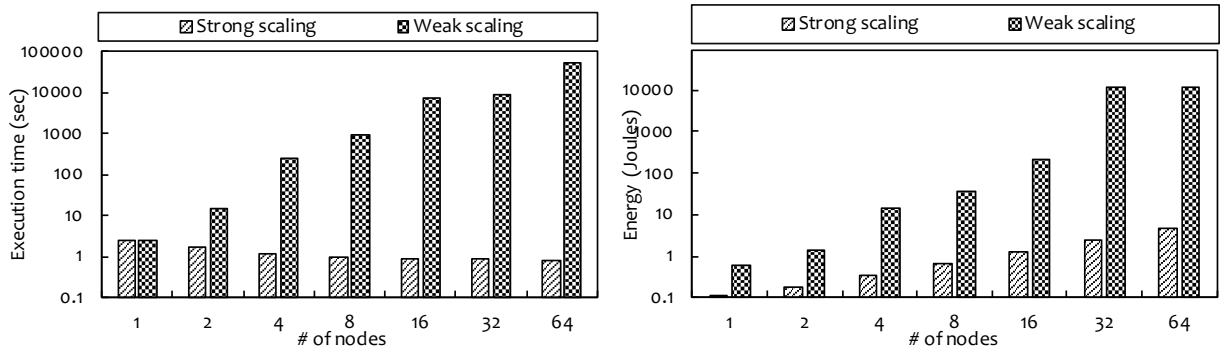


Figure 4.10: Sensitivity analysis of LULESH

LULESH is an application with both compute and memory intensive properties; however, its evaluation on our experimental system shows its trend towards the memory intensive group. We followed the Prometheus workflow by first applying Roofline model [173] and Roofline model of energy [31] to characterize the application, followed by collection of hardware performance counters. We used the machine learning algorithm on the training set of performance counters to identify the significant counters. The significant counters are mapped to their corresponding hardware components using the abstract machine model configurator. We changed the configurations of the significant hardware components in Aspen's abstract machine model.



(a) Runtime (b) Energy

Figure 4.11: Scalability analysis for LULESH

Figure 4.10 shows the effect of changing significant hardware components on runtime and energy for LULESH in isolation as well in combinations of two components.

Sensitivity towards runtime: Figure 4.10 shows that runtime is most sensitive to system clock and number of cores individually and in combination. Our profiling results show that the *Hourglass* kernel is most sensitive to system clock frequency and the *Stress calculations* kernel is most sensitive to the number of cores, and these two are a few of the most influential kernels in the overall application execution. Hence, the overall application is most sensitive to the combination of system clock frequency and number of cores.

Sensitivity towards energy improvement Figure 4.10 shows that energy consumption is sensitive to multiple hardware components – i.e., system clock, memory clock and number of cores in isolation. Energy consumption is also sensitive to the combination of system clock and number of cores as well as the combined effect of system and memory clock. Similar to runtime, *hourglass calculation, element connectivity* and *stress calculations* are the most energy-consuming kernels. Each of them have different computation and memory access characteristics, and that is why there are a number of hardware configurations that can be used to improve

the energy efficiency.

Scaling effects for runtime and energy improvement Figure 4.11(a), 4.11(b) shows the strong and weak scaling effects with respect to changing system clock and the # of cores in combination (identified as the hardware component that provides the improved performance) for LULESH runtime and energy. The execution time reduces with increasing number of nodes, which means there is a lot of opportunity to improve application performance using parallelism for both runtime and energy. However, for weak scaling, the execution time is increasing, therefore the application will not benefit from weak scaling.

Since LULESH performs all the calculations locally, it shows a lesser trend towards weak scaling. We are using MPI version of LULESH; we believe that a hybrid MPI-OpenMP version will have degraded weak scaling effects as there will be more data movement involved to reduce the race conditions for communication critical kernels e.g., hourglass and stress computations.

Case Study 2: CoMD

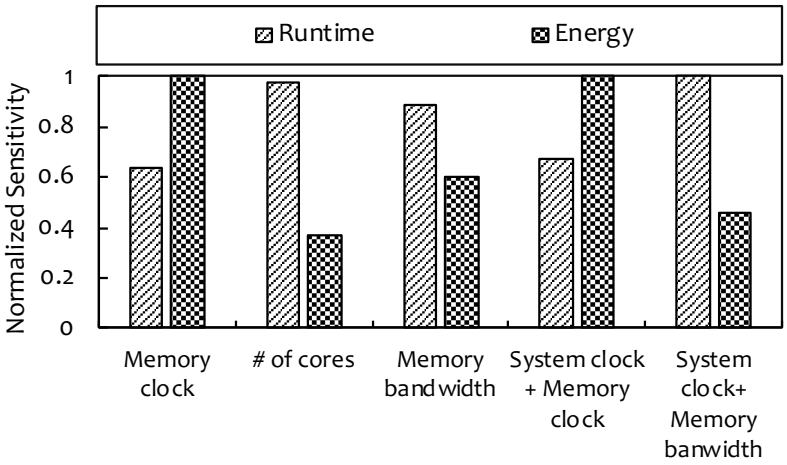
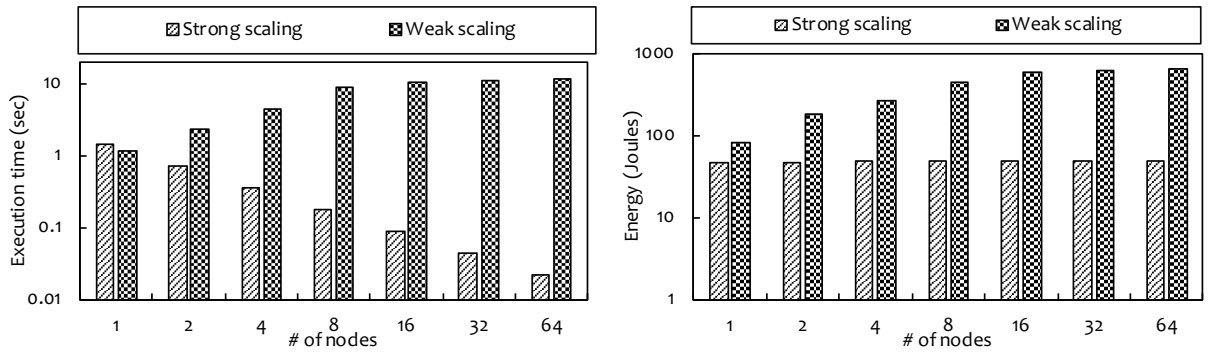


Figure 4.12: Sensitivity analysis of CoMD



(a) Runtime (b) Energy

Figure 4.13: CoMD strong and weak scaling

Sensitivity towards runtime Figure 4.12 shows the sensitivity analysis for runtime. Increased number of cores in isolation and system clock and memory bandwidth in combination show the major effects on runtime. There are two major runtime dominant components in CoMD, the *potential calculation* within a node and the *communication* between nodes. For potential calculation within a node – potential, velocity and force are organized as 3-dimensional data which requires much memory access. Hence increasing the memory bandwidth with increased clock frequency proves to be the best hardware configuration as it enables more data accesses at a faster rate and a consequential effect of faster intra-node potential velocity and force calculation due to faster clock speed.

Sensitivity towards energy improvement Sensitivity of hardware components for energy improvement is shown in figure 4.12. Energy consumption is most sensitive to memory clock and combined effect of system clock and memory clock. The intra-node computations (velocity and position calculations) are a few of the most time and energy consuming components in CoMD. There are some optimizations already provided by the ExMatEX CoMD code e.g., using smaller coefficients to calculate potential within a node to improve memory accesses, and these optimizations increase the arithmetic intensity which consequently improves the energy

efficiency. The optimal hardware configuration proposed by *Prometheus* is increasing memory clock frequency. Increasing memory clock frequency will help increase the number of memory operations per cycle, which will improve the overall execution time.

Scaling effects for runtime and energy improvement Scaling effect with best hardware configuration – i.e., the effect of increases in system clock and memory bandwidth together – are shown in figure 4.13(a) & 4.13(b). Since computation within the node is a major part in application execution, the execution time improves with increasing number of nodes.

We do not observe any positive effects of weak scaling on CoMD runtime and energy improvement. Since the number of messages transmitted during each iteration increases with increasing processes, therefore communication overhead increases resulting in increased runtime and energy improvement. Although there is a window to achieve good performance from weak scaling, this may incur some memory latency issues depending upon the interconnect in the system.

Case Study 3: CoEVP

Sensitivity towards runtime improvement Figure 4.14 shows the sensitivity of runtime to various hardware configurations. Although CoEVP exhibits both compute and memory intensive characteristics, the significant hardware components showing the most effect are memory bandwidth in isolation and system clock and memory bandwidth in combination. Since CoEVP makes extensive use of mathematical functions to complete computation, therefore system clock is a hardware component that helps in calculating the mathematical functions quicker and helps to improves the runtime of the application. Moreover, it also exhibits the properties of the LULESH proxy application, e.g., *Hourglass and stress calculation* kernels, hence memory bandwidth is also a significant hardware component as mentioned in LULESHs sensitivity

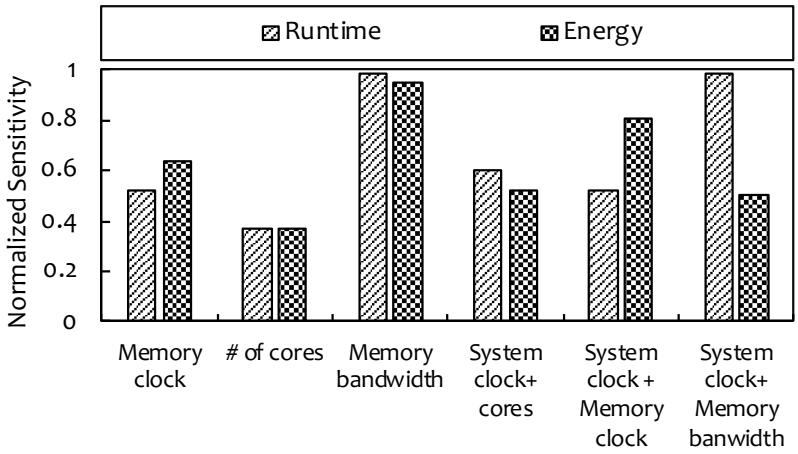
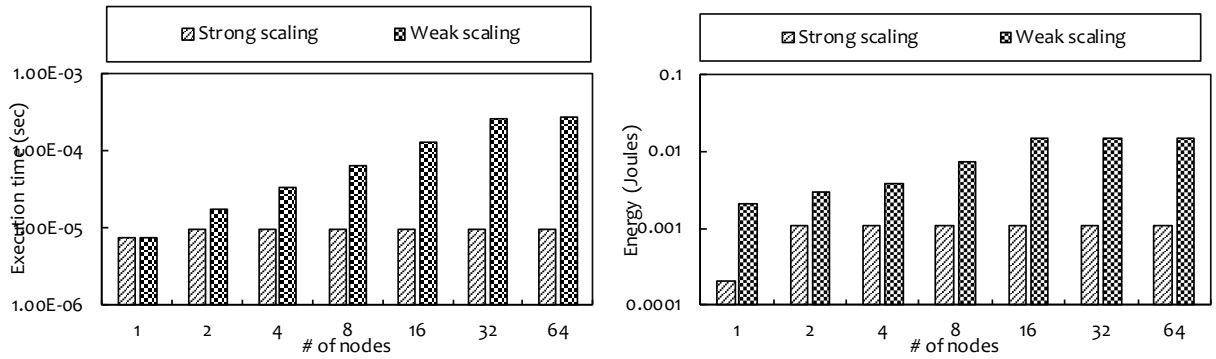


Figure 4.14: Sensitivity analysis of CoEVP



(a) Runtime (b) Energy

Figure 4.15: CoEVP strong and weak scaling

analysis.

Sensitivity towards energy improvement Normalized energy consumption sensitivity is shown in Figure-4.14, the most sensitive hardware components are the same as that for the runtime. Since increasing the system clock reduces the time taken by most time consuming kernels i.e., *Hourglass and stress calculation kernel*, the energy consumption also improves.

Scaling effects for runtime and energy improvement Figures 4.15(a) & 4.15(b) show the effect of strong and weak scaling for runtime and energy for CoEVP. Similar to CoMD, there is no effect of improvement in the case of weak scaling for runtime and energy improvement. Since with increasing processes, the communication overhead is also increased at the same rate, we do not observe any improvement. For strong scaling, the application scales well with increasing processes both for runtime and energy improvement.

Case Study 4: CG

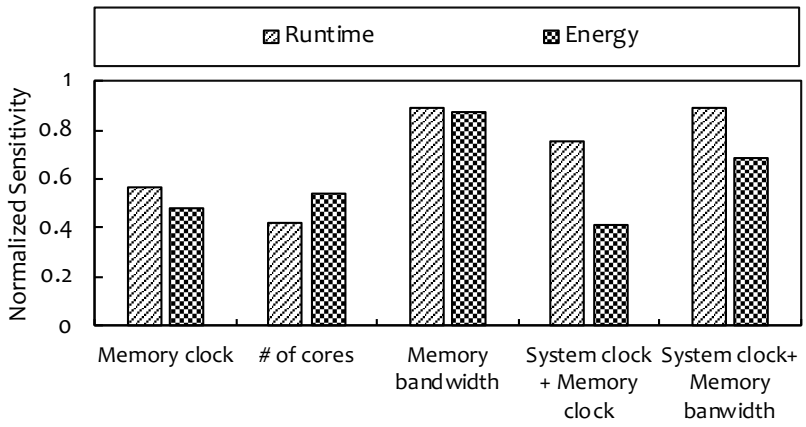
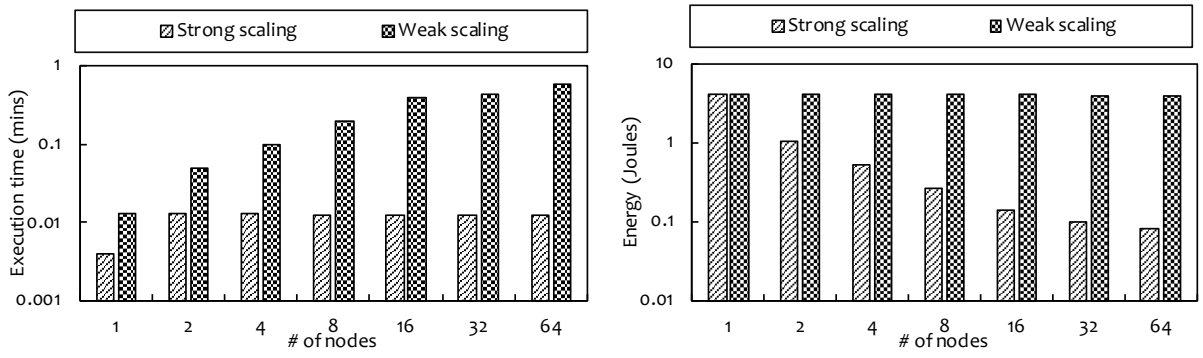


Figure 4.16: Sensitivity analysis of CG

Sensitivity towards runtime improvement Sensitivity with respect to runtime is shown in Figure 4.16, which represents the effect of varying configurations of various hardware components. As can be seen in figure 4.16, memory bandwidth in isolation and in combination with system clock significantly affects runtime. CG exhibits the properties of sparse pattern for every computation performed on each row, which results in increased memory access. This slow down effect in runtime will be more prominent in a low memory bandwidth systems, hence improving the memory bandwidth helps to increase the speedup. Increased system clock im-



(a) Runtime (b) Energy

Figure 4.17: CG strong and weak scaling

proved memory bandwidth operations too, and therefore helps to access more elements in each clock cycle and improves performance.

Sensitivity towards energy improvement Sensitivity of energy consumption to changing configurations of various hardware components is shown in Figure 4.16. As can be seen in Figure 4.16, memory bandwidth affects the energy consumption pattern by a large amount because of increased memory accesses on each computation. Similar to runtime, improving memory bandwidth will help reduce the time required to access data from memory and therefore improves the energy efficiency.

Scaling effects for runtime and energy improvement Strong and weak scaling for runtime and energy are demonstrated in Figure 4.17(a) & 4.17(b) respectively. Strong scaling shows no improvement for runtime, while weak scaling shows an increase in runtime. For energy, strong scaling shows energy improvement while weak scaling does not show any effect.

4.5 Validation results

In order to validate the results produced by Prometheus, we performed three set of experiments. We validate varied system clock and throttled memory bandwidth results on a local cluster composed of two physical CPUs, each containing an 8 core 64-bit Intel Haswell processor (Xeon E5-2637) at 3.50 GHz. There are two GPUs on the system: an NVIDIA TESLA K20c with 5-GB GDDR5 memory and and an NVIDIA Quadro K600 with a 1-GB GDDR3 memory. In this chapter, we use Intel's Xeon CPU only to study the effect of system clock and memory bandwidth frequencies.

For the validation of scalability results, we performed strong and week scaling experiments on Lawrence Berkeley Lab's (NERSC) Cori system. Cori was installed in two phases. The first phase consists of 1632 dual socket compute nodes that comprise of two 2.3 GHz 16-core Intel's Haswell processors and 128 GB of DRAM per node. The latest phase consists of over 96000 compute nodes composed of Intel's Knight Landing (KNL). We use the phase-1 Intel's Haswell processor in our experiments. We show the following validation results in this section:

1. Validation of applications sensitivity results for runtime by varying system clock frequency
2. Validation of applications sensitivity results for runtime by throttling memory bandwidth
3. Analysis of scalability results for runtime for CoMD proxy application

4.5.1 Varied clock frequency validation results

We enabled DVFS by varying system clock frequency at runtime using Linux user and kernel level commands. We varied the system clock frequency between minimum and maximum ranges provided by the system, our system supports varying clock frequency between 1200

MHz to 3500 MHz. Figure 4.18 shows the cumulative effect of varying clock frequency on four proxy applications under study. As can be seen from Figure 4.18 measured results closely matches the predicted results provided by Prometheus. The measured results always provide less value than predicted results, since Prometheus ignores the runtime overhead.

The analytical modeling does not take into account the runtime effects of the system, and therefore show a higher range for all the four candidate applications.

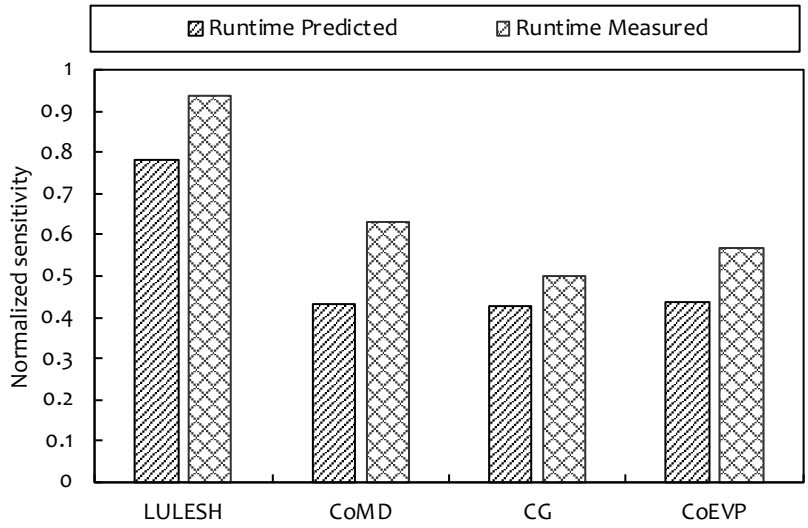


Figure 4.18: Evaluation of measured and predicted runtime with varying clock frequency

4.5.2 Memory bandwidth throttling validation results

We also validated the results for varying memory bandwidth for the candidate proxy applications. Our idea of memory throttling is inspired by Gremlin emulator [107], a tool that emulates resource restriction for the architecture of next generation HPC systems. It provides four major classes of resource restriction.

1. *Power Gremlin* mimics the behavior of reduced power of a CPU.

2. *Memory Gremlin* limits memory resources e.g., memory bandwidth, etc.,

3. *Resilience Gremlin* instills faults in to the application under observation.

4. *Noise Gremlin* adds noise to systems and operating system performance.

Inspired by memory Gremlin, we tested our system for constrained memory bandwidth using Stream benchmark. We run STREAM benchmark [112] simultaneously while executing the proxy application under observation (LULESH, CoMD, CG, CoEVP) on a memory bandwidth constrained system. We also ran varying instances of STREAM benchmark to throttle the memory bandwidth in a piecewise manner. Figure 4.19 shows the comparison between measured and predicted sensitivities towards memory bandwidth for the four proxy applications. As seen in the validation results (Figure 4.18) for varied system clock frequency, measured results are less than predicted results, but closely follow the trend. The predicted values proposed by memory bandwidth show higher numbers because of the inability to capture the system noise and other system variances that comes into effect while observing measured values.

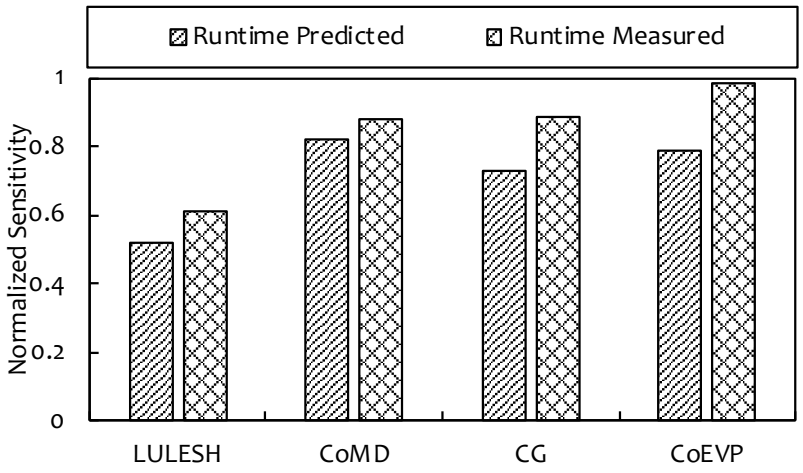


Figure 4.19: Evaluation of measured and predicted runtime with varying memory bandwidth

4.5.3 Validation of scalability results for runtime on Cori

We tested scaling results on NERSC's Cori system. We used phase-1 of Cori consisting of Intel's Haswell processors to test the scalability of the proxy application (Figure 4.20). We tested CoMD for the strong and weak scaling analysis. CoMD's execution time decreases with increasing number of processes, which validates the increasing sensitivity for strong scaling shown in figure 4.13(a). CoMD's runtime does not reduce with increasing number of processes and increasing input size simultaneously – weak scaling. This effect is shown by Prometheus in figure 4.13(a), where weak scaling shows no improvement in the runtime for CoMD.

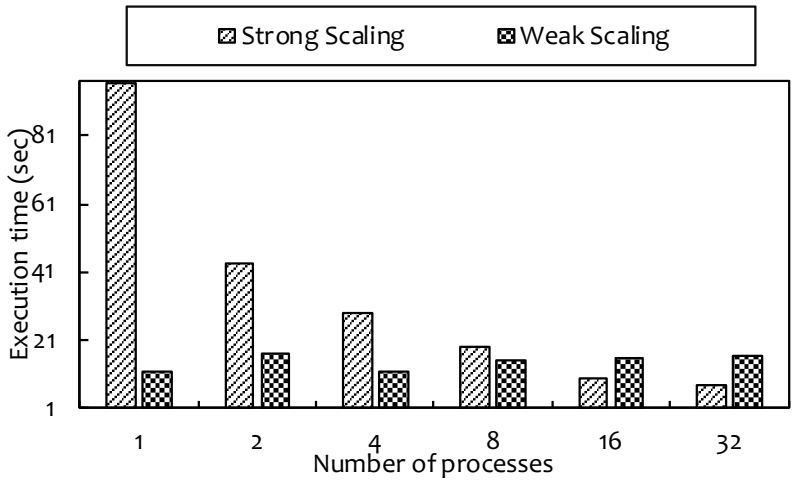


Figure 4.20: Evaluation of scalability results for CoMD

4.5.4 Mapping towards exascale architectures

Lulesh & CoEVP

Figure 4.10 shows that the hardware components that improve performance of LULESH are system clock, number of cores for runtime, and system clock, memory clock and number of cores in isolation and in combination for energy. CoEVP is an alternative to ALE3D and uses

LULESH in the coarse-scale model, and it is sensitive to memory bandwidth, system clock and their increased speed and capacity in combination.

System clock is expected to remain the same for exascale architectures, therefore researchers might revert towards employing increased number of cores in the experiments to improve LULESH and CoEVP class of applications on a reconfigurable architecture cluster. But making a decision at runtime about how many cores should be used, is dictated by application type and the number of nodes in the hardware. Therefore to improve runtime and energy using number of cores, choosing appropriate number of nodes and managing data movement (in case of communication intensive applications) must be handled in conjunction. Moreover, increasing cores will affect the utilization of available memory bandwidth too, as the intra-node communication will be a dominating factor affecting performance improvements.

CoMD & CG

The hardware components that affect the application performance to the most extent are number of cores, memory clock and memory bandwidth. CG is sensitive to increased memory bandwidth and system clock. Memory bandwidth is a very important component in the design of exascale architectures, and therefore architects are exploring high bandwidth memories e.g., HBM. Programming languages and models can gain advantage from data locality of complex memory architectures expected in exascale systems. Some notable directive based programming models [98, 178] mention the available but yet to be explored methods of programming to improve memory bandwidth performance. The other possible approach to gain benefit from memory bandwidth is to shift power from other hardware components towards memory bandwidth.

4.6 Use Cases

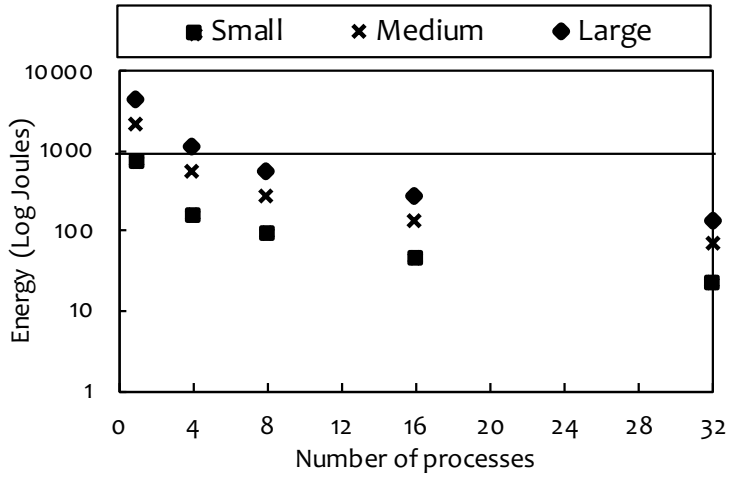


Figure 4.21: Energy consumption pattern for CoMD

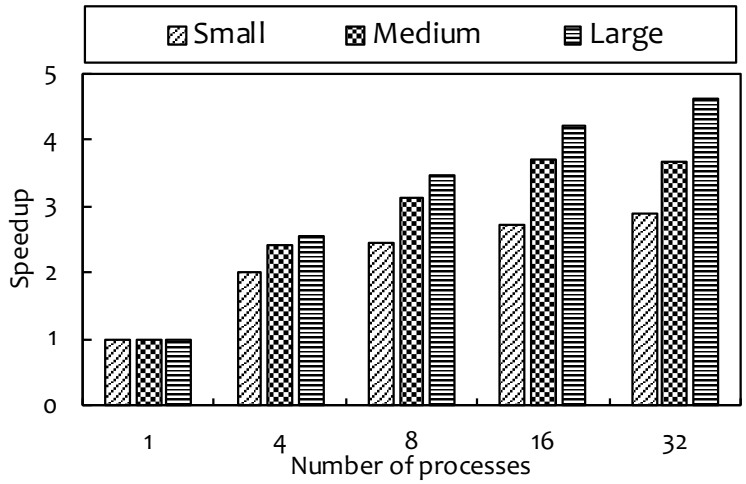


Figure 4.22: Strong scaling of runtime of CoMD for different input sizes

Prometheus can help to achieve optimal performance by understanding the impact of hardware on application execution and system utilization. We present a few use cases where Prometheus can be deployed.

4.6.1 Adjusting node configuration per power/energy budget

Maximizing performance and power/energy improvement is a system design goal for evolving architectures. Therefore, it is important to set and stay under an energy budget. Moreover, following the energy budget can also help researchers to design configurations for their reconfigurable architectures (e.g., optimal # of nodes, clock frequency, memory characteristics).

In Figure 4.21, we present results of maintaining the energy consumption of CoMD with respect to strong scaling for three different input sizes. The hypothetical energy budget is 1000 Joules, as indicated by the line in Figure 4.21. The execution for small input size stays under budget, but making an informed decision about number of nodes for medium and large inputs can help us stay in the energy limiting budget.

4.6.2 Proposing evolving architectures

Our framework can also help to design new exascale architectures. We believe that future architectures will have more reconfigurable options.

Assume a postulated cluster, customized for two classes of applications (LULESH and CG). As mentioned in §4.4, Lulesh shows improved performance towards increase clock frequency and number of nodes, while CG is sensitive to higher memory bandwidth. Nodes customized for LULESH will exhibit properties of higher clock frequencies and encompasses more nodes. Similarly the other group of nodes will possess more memory bandwidth and less memory latency to suit the application characteristics of CG.

4.6.3 Choosing appropriate number of nodes

Benefits of scaling in terms of improved performance and energy consumption can be achieved by choosing the optimal number of nodes. The traditional approach to choose optimal nodes is to repeat the experiments on the cluster and learn the trend from empirical values. A framework like Prometheus which abstracts the behavior of hardware and has the ability to study co-design issues; will be more time and resource efficient than empirical training.

Figure 4.22 shows that for small and medium input sizes, 16 is the optimal number of nodes; beyond that will not provide much benefit. However, for large data sizes, the speedup benefit continues to grow until 32 nodes.

4.7 Summary

Hardware resource optimization holds the same importance and priority as application optimization as we move towards the exascale era. It is crucial to understand the effect of hardware components on an application execution as it plays a key role in the overall performance improvement of application and system in conjunction. We present a methodology called *Prometheus* that uses application signature as a guideline to identify hardware bottlenecks, measure the sensitivity of significant hardware components, and configure selected hardware components to show the effect on the application performance and energy consumption. *Prometheus* encapsulates a combination of analytical and machine learning techniques, and uses Aspen DSL as a modular and composable tool to extract the benefits of the various modeling techniques e.g., Roofline, supervised machine learning etc. Aspen DSL also helps us in changing and testing configurations of hardware with applications and showing their effect on application performance and energy footprint. We validate our results on 4 proxy

applications, LULESH, CoMD, CoEVP and CG.

In the next chapter (chapter 5), we present our second contribution towards design space exploration techniques. We implement the homogeneous exascale proxy architecture. We validate it for theoretical peak performance and present a communication and computation model for co-design for molecular dynamics code.

Chapter 5

Performance and Communication Modeling for Exascale Architectures

Current high-performance computing systems are at the peak of their computational performance and therefore the research community is driving efforts towards deployment of exascale systems. Application and hardware design has to go hand-in-hand for the evolution of any new system. The complexity of studying and analyzing applications and hardware of exascale is much more challenging as the hardware does not exist today. DOE has proposed a set of four theoretical abstract architectures with recommended system configurations. The development of an application suite has progressed to provide a suite of 13 proxy applications that are representative for exascale systems. Any research involving the exploration of exascale systems has to first devise a technique to transform the theoretical abstract architecture to proxy architecture and validate it to see whether it provides the same peak theoretical performance limits as expected by DOE. Later, this proxy architecture can be used to explore the patterns of execution of proxy applications and to study various performance, scalability and communication aspects of the system. Moreover, the predictive modeling for communication

and computation at the scale of exaFlops requires complex processing on application type, its input size, hardware configuration, and systems conditions.

We deal with two issues in this chapter, the realization of exascale architecture in terms of proxy machine model deployment and testing it for communicational and computational constraints. There has been some research done to propose and test various prototype exascale hardware, that represents the scalability of proposed exascale machines [165, 167], however, the resources required to generate prototype hardware are limited. Simulators, have a long prevalence to model various kinds and scales of architecture [20, 82]. ExaSAT [157] is a scalable simulator that is developed specifically for varying and testing the scale of exascale architectures. Emulators have also been used to test and reconfigure various architectural features [107]. Machine learning and statistical modeling techniques have been used to study the effect of application performance on current Giga and Peta scale performing systems, and those results are used to propose and extrapolate towards exascale architectures. Though all of these techniques have worked for the specific research area, these are not applicable to a wider problem area. None of the modeling and performance prediction techniques reflect the abstract architectures proposed by DOE [8]. Even if we were to ignore the expected scale of future architectures, communication parameters and execution time prediction is itself a challenging problem, which takes into account the number of nodes, processes per node, data distribution, data placement and so forth. A number of communication prediction models have been proposed to optimize applications in terms of reducing message size [51], reducing communication overhead [42], hybrid MPI/OpenMP modeling [65], and automatic mapping to heterogeneous resources [102].

Our contributions in this domain are two-fold: We have implemented a homogeneous abstract machine model and present it as parameterized proxy architectures. We use Aspen DSL – a fast, accurate, portable and composable domain specific language to develop the proxy homo-

geneous architecture design. We test the theoretical peak performance of the proxy homogeneous architecture in Aspen versus that is predicted by the exascale computing project. We test the Co-design of Molecular Dynamics (CoMD) Proxy application on the proxy architecture in Aspen. As the main experimental consideration on the extent of exascale architecture is the span of computational nodes, it is expected that communication design and mapping will be a critical modeling issue.

We develop and test a communication mapping framework in Aspen that uses communication and computation modeling of the application in order to map the application on the hardware. Moreover, it shows near-optimal communication mapping parameters e.g., optimal number of ranks, memory type to use in a multiple type of memory hierarchy, etc.

Our contributions in this direction are:

1. Implementation of a homogeneous, exascale abstract machine model [8] using Aspen and validation of the model for theoretical peak performance

2. Validation of the homogeneous exascale architecture using the CoMD proxy application

3. Design and implementation of a communication mapping framework in Aspen that maps the profiled application requirements with hardware requirements and provides a summary of the near-optimal communication mapping choices at runtime. We develop the framework and validate the results using the supercomputing facilities of Cori at NERSC.

5.1 Design

In this section, first we explain the theoretical constraints and requirements of the homogeneous exascale abstract architecture and how we implemented and validated them in Aspen.

Later, we propose our analytical communication and computation model that maps the proxy application to the candidate proxy architecture in Aspen.

5.1.1 Abstract homogeneous exascale architecture

In the top ten exascale challenges [7], the implementation of an abstract machine model and an abstract application model are defined as two important aspects of exploration of co-design required by exascale systems. Ang et al, [8] define four different abstract machine models, (homogeneous, non-integrated heterogeneous, integrated heterogeneous and tightly integrated heterogeneous). In each of the abstract models, nodes can comprise of multiple units of multi-cores, along with the accelerator (in case of heterogeneous architectures). The individual processors of the exascale architecture are currently unknown, and therefore the current processor technology is expected to be used as the basic building blocks of the system. All the abstract models define an on-chip and off-chip memory and an on-chip network required to process the communication between multiple processing units.

Figure 5.1 shows the block diagram of a homogeneous CPU based architecture(source: [8]). In this chapter, we implement Intel's Haswell processor as the basic processor technology in the architecture since it is proposed that existing compute units will remain the basic building blocks of the exascale system. A conceptual view of Intel's Haswell processor is shown in Figure 5.2.

Table 5.1 lists the configurations of the abstract homogeneous architecture implemented in this chapter, as proposed by Ang et al, [8]. Each processor is a 64-128 bit, dual-quad threaded, 8-16 issue and out of order execution micro-processor. We use three levels of cache hierarchies for each microprocessor, where L1 cache is in the range of $8KB-128KB$, L2 cache ranges between $256KB-2MB$, and L3 cache contains $64MB-128MB$. As mentioned in Figure 5.1,

two types of memories are mentioned in the system, where the in-package memory is expected to be of HBM (high-bandwidth) memory [18], and is marked having a capacity of 64-192 GB. The off-chip memory is proposed to be using DRAM technology and provides a much larger capacity i.e., 512GB-2TB. An off-chip NVRAM (2TB-16TB) is also expected to be included to provide asymmetric access latency. A summary of this information is provided in Table 5.1 [8]. Figure 5.3 shows the high-level representation of individual nodes in an overall view of system.

The interconnect technology to be used for exascale system is of high-importance. The latency incurred by the interconnect may cause a serious performance issue. A number of high bandwidth and low latency interconnects are considered for exascale architecture by Ang et al. [8]. In this chapter, we implement and explore the Dragon Fly interconnect for homogeneous exascale architectures that provides the benefit of reduced optical interconnect and enables high-bisection bandwidth.

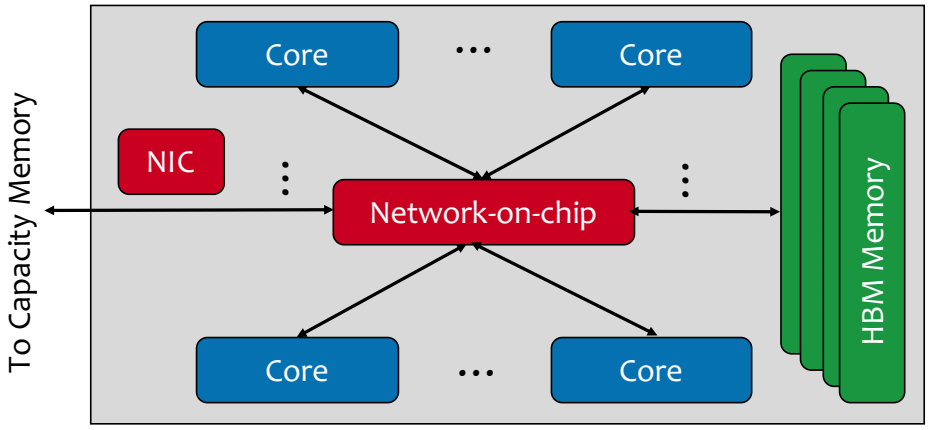


Figure 5.1: Homogeneous Abstract Machine Model [8]

5.1.2 Validation criteria for exascale proxy architecture

We implemented the proxy machine model in Aspen that represents the theoretical homogeneous abstract machine model presented by Ang et al. [8]. Previously we used the proxy

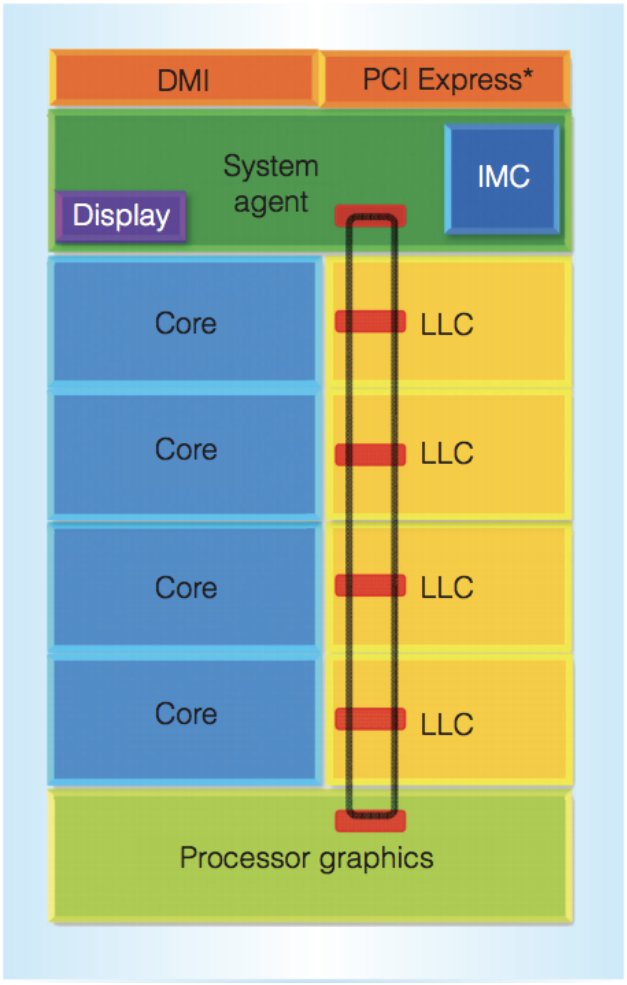


Figure 5.2: Conceptual diagram of Intel's Haswell processor (Source [73])

architecture for testing applications. The proxy architecture needs to be validated to show that whether the peak theoretical performance values are representative of what was expected by the exascale computing project by Ang et al. [7]. We have developed a set of micro-kernel benchmarks that saturates the machine for the required performance metric under observation in Aspen. The list of a set of micro kernel benchmarks and their brief explanation is as follows.

- *Theoretical maximum single-precision Flops*

  Provides the maximum single-precision Flops produced by the overall system composed of $\sim 125K$ nodes.

Table 5.1: Theoretical abstract machine model configurations [8]

Parameter	**Range**
Number of nodes	33K-125K
Number of cores (per chip)	64-256
Processor cores/node	256
NUMA region/node	64
Gflops/per proc core	64
Processor SIMD vectors (units X width)	8X16
Off-chip memory B.W Chip-memory	60-100 GB/s
On-Chip memory B.W Chip-memory	600-1200 GB/s
Network-on-chip B.W Per Core	8-64 GB/s
In-package memory capacity	32-64 GB
Off-chip memory capacity	∼512 GB-2TB

- *Theoretical maximum double-precision Flops*

  Provides the maximum double-precision Flops produced by overall system composed of $\sim 125K$ nodes.

- *Theoretical maximum single-precision Flops per Joules*

  The energy consumed by the system when working on theoretical maximum single-precision floating-point operations.

- *Theoretical maximum double-precision Flops per Joules*

  The energy consumed by the system when working on theoretical maximum double-precision floating-point operations.

- *Theoretical maximum Memory bandwidth*

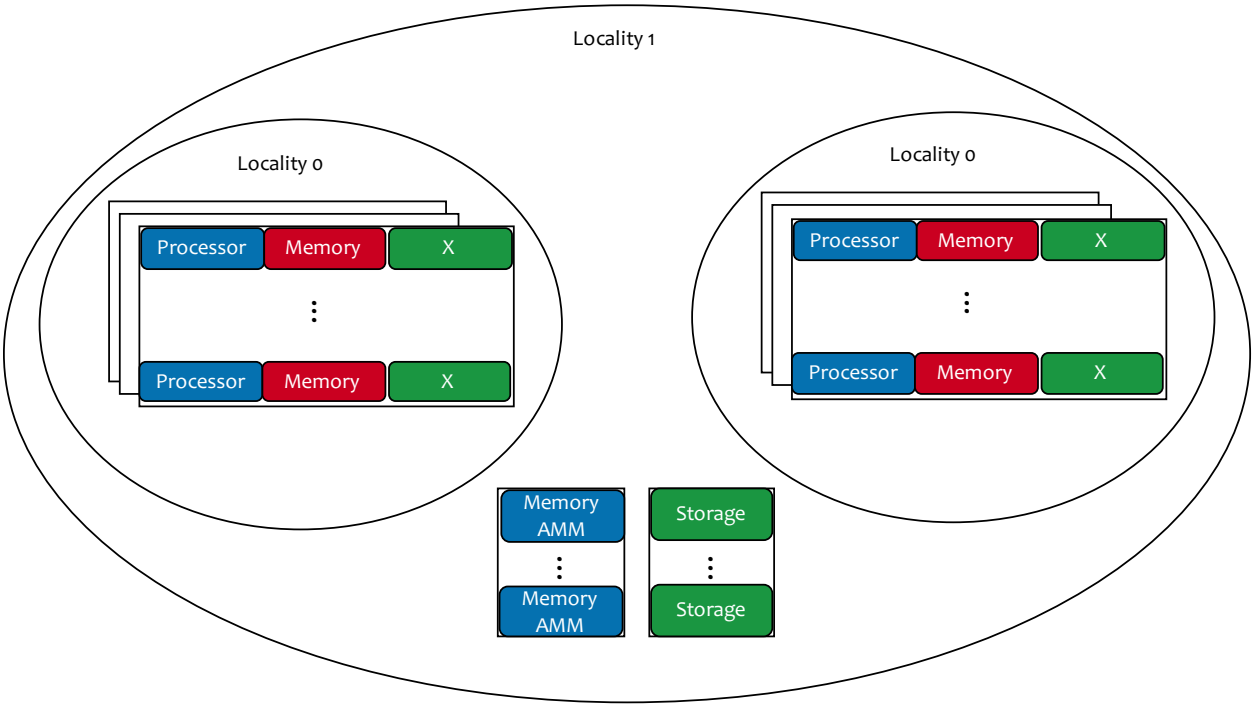


Figure 5.3: System abstract machine model

The achievable maximum memory bandwidth provided by the system.

- *Theoretical maximum Memory bandwidth per Joules*

  An estimate of the energy consumed while providing maximum memory bandwidth.

- *Roofline analysis*

  A basic Roofline model showing computational and memory bounds of the system.

We tested our proxy homogeneous machine model in Aspen with the micro-kernel benchmarks for verification before deploying the system for the execution of the proxy application. The details of the verification results are shown in Section 5.2.

5.1.3 Proxy application for exascale architecture

We tested CoMD on the homogeneous proxy architecture in Aspen DSL environment to explore the execution pattern as well as communication and computation requirements of the application. First we will explain computation and communication models and later we will explain the mapping framework that builds on top of these analytical models.

CoMD application characteristics

CoMD is the co-design of molecular dynamics application code. It is a simulation code that focuses on the potential calculation among the elements. It performs two computation intensive operations i.e., discovery of pairs of atoms within a distance and calculation of force between those atoms. Data in CoMD is represented in three dimensions (Nx, Ny, Nz), where the atoms are represented in three dimensions. A few of the optimization strategies are defined to improve the performance of CoMD, including the division of the problem area into multiple small areas , where the number of areas can be decided such as it is equal to the number of MPI tasks. Individual atoms are grouped based on their physical location, and this requires computations related to force calculation, velocity calculation and position calculation within its circle and around the circle of the atom.

5.1.4 Abstract application model of CoMD in Aspen

The main emphasis in CoMD is the spatial decomposition of atoms [131] into small boxes. The CoMD code provided by ExMatEX and implemented in Aspen in abstract form requires data to be decomposed into dimensions equal to the number of processors, which helps to avoid load imbalance issues. In Aspen abstract application model, we distinguish the input elements

as atoms within the domain of the processor and the neighboring atoms, in a similar way that ExMatEX implements CoMD. Each processor calculates the atom's velocity, position and distance with other neighboring elements. All the information related to the atoms are stored in the form of a structure of arrays, consisting of atom's bond, interaction with neighboring atoms, angle, etc. The abstract application model follows a number of steps discussed in detail in the next section.

5.1.5 Communication & computation modeling and mapping framework in Aspen

Figure 5.4 shows the block diagram for all the steps required to calculate the computation and communication constraints of an application and to execute the mapping of abstract application model on the proxy homogeneous exascale architecture on Aspen. The main purpose of this framework is to map communication and computation requirements of the application with the machine model. The mapping framework consists of the following steps:

1. Abstract application and machine model generation in Aspen. This step is skipped if the model already exists in Aspen.

2. Extract computational and communication parameters from the Aspen abstract application model.

3. Compute computation and communication models that predict the patterns of execution on single and multiple nodes.

4. Generate a mapping framework that maps the abstract application with machine model on the exascale architecture and proposes the optimal number of ranks and other communication optimizations that can improve application execution.

A description of each step is given below:

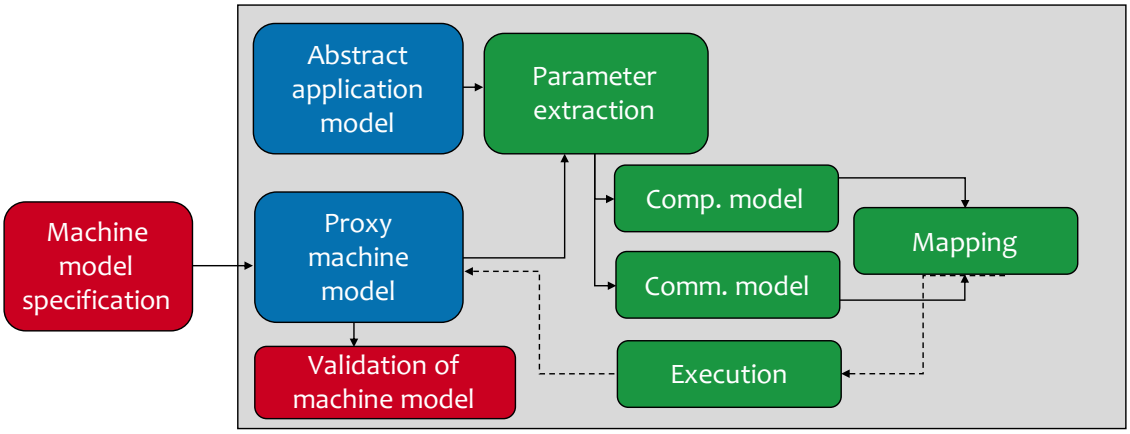


Figure 5.4: Mapping framework in Aspen

Step#1: Abstract application and machine model

We showed previously the steps required for the implementation of an abstract application and machine model in Aspen. The application and machine model need to be represented as an abstraction [147] before being tested for the mapping framework. An example of steps implemented in the abstract application model for CoMD is shown in Algorithm-2.

Input: Atoms per cell, # of processors
Result: position, velocity, clustering
while *all processors* **do**
    Calculate position;
    Communicate with neighbors;
    Calculate bonded & un-bonded forces;
    Calculate velocity;
    Calculate position;
    Communicate if necessary;
end

Algorithm 2: CoMD abstract application model in Aspen

Step#2:Parameter extraction for computation and communication model

In this step, we extract the application's parameters that are required to implement computational and communicational modeling. We extract Flops of the application, amount of total data exchanged during application execution, memory requirements, input size and the complexity of the application to use in the computation model. We also extract some of the machine characteristics i.e., number of processors used and the bandwidth of the memory – that will play a key role in the communication of the application on multiple processing elements.

Step#3:Computation and communication modeling

This section describes the computation and communication model based on the extracted applications and machine characteristics. A common method for accurate performance modeling is to monitor performance counters. However, since the hardware for exascale does not exist, we implement a system level performance model. Our model takes into account computation and communication aspects of the application. Our model uses applications complexity, number of memory accesses, and time taken by the computation to populate the model. We consider computation and communication as separate entities to find the total time taken by the application to execute. T_{total} defines the time taken by the application to execute on proxy homogeneous architecture. T_{comp} & T_{comm} are the time taken during computation and communication phases. We calculate the time taken by the application as the *sum* of computation and communication.

$$T_{total} = T_{comp} + T_{comm} \tag{5.1}$$

Computation model We partition the time taken by computation into time to execute the instructions(Flops) and time spent in accessing the memory. Inspired by Czechowski et al. [38], we calculate the flops and memory costs in terms of T_{flops} and T_{mem}. We use the max function between flops and memory, in a similar way that Aspen calculates the time taken by the application i.e., maximum of the time spent using processor or memory.

$$T_{comp} = max(T_{flops} + T_{mem}) \tag{5.2}$$

Each floating point operation in CoMD operates in three dimensions, so the input data is represented in three-dimensions i.e., (N_x, N_y and N_z). This three-dimensional data (N_x,N_y,N_z) is distributed and operated by each processing element (P), which represents the first fraction of the equation 5.3 on the amount of work performed by each processing element(P).

The second fraction of the equation 5.3 defines the flops executed by each processor in terms of an exponential function of the total input size i.e., $AtomsPerCell$. We divide the total input by the maximum theoretical flops provided by each processor ($P_{peakFlops}$).

Since we are targeting a homogeneous CPU based exascale architecture, we assume that the peak theoretical Flops provided by each processor is the same. In the case of heterogeneous architectures (e.g., GPUs, FPGAs), multiple flops cost estimations must be used, corresponding to the peak theoretical flops provided by each processing unit.

$$T_{flops} = \frac{N_x * N_y * N_z}{P} * \frac{log(AtomsPerCell)}{P_{peakFlops}} \tag{5.3}$$

In case of cost of the memory operations, we partition total input size i.e., ($AtomsPerCell$) into three dimensions (N_x,N_y,N_z) distributed to each processor (P), which represents the first fraction of equation 5.4.

The second fraction in equation 5.4 represents memory operations performed by each processor on total input size i.e.,$(AtomsPerCell)$. This factor is dependent upon the peak theoretical bisection bandwidth of the memory β_{mem}.

$$T_{mem} = \frac{N_x * N_y * N_z}{P} * \frac{log(AtomsPerCell)}{\beta_{mem}} \tag{5.4}$$

Communication model We implement an analytical model for communication between various domains of the proxy exascale architecture. We calculate the time taken during communication (T_{comm}) as the time taken for communication between the nodes ($T_{intranode}$) and within the node ($T_{internode}$).

$$T_{comm} = T_{intranode} + T_{internode} \tag{5.5}$$

In case of communication within the node, the amount of data transferred is in terms of total atoms i.e.,(N). CoMD does not encounter any load imbalance as the amount of data, $(AtomsPerCell)$ is always the same that is transferred to each processing element. We take the product of the data that is communicated $(2^{1/N} * log(AtomsPerCell, 10))$ with the time (t) taken to transfer it in order to calculate the communication within the node ($T_{intranode}$). We divide the data communicated by the theoretical bandwidth, where β_{mem} is the bisection bandwidth of the memory. Since all the nodes have the same theoretical peak bandwidth, the same β_{mem} is used to compute the time taken during the intra node communication.

$$T_{intranode} = \frac{2^{1/N} * log(AtomsPerCell, 10)}{\beta_{mem}} * t \tag{5.6}$$

The time taken for communication between the nodes is represented as $T_{internode}$. It is the combination of network latency in micro/nano seconds and time taken during intra node com-

munication (t). The latency of the interconnect is represented as ℓ, and its value is provided by Aspen as the theoretical limit of the interconnect.

$$T_{internode} = \ell + \frac{2^{1/N} * log(AtomsPerCell, 10)}{\beta_{memory}} * t \tag{5.7}$$

Step#4:Mapping framework

After generating communication and computation models for CoMD, we test the application with varying number of processors in order to find the number of processors that provides the optimal execution in terms of communication and computation. While executing the application in Aspen, we also monitor the dominant communication calls, and provide the recommendation regarding optimizing the dominant communication calls, considering the scale of the future architectures.

5.2 Results and Analysis

The main strength of the analytical modeling and mapping framework is to model the communication and computation patterns of the application on the proxy architecture. It also helps to identify the near optimal number of ranks that can be used in an exascale homogeneous architecture consisting of millions of compute processors. To show the effectiveness of the modeling and mapping framework, we present two types of results in this section. We start with presenting the theoretical limits of the exascale homogeneous architecture that matches with the values proposed by the exascale computing project. After validating the theoretical configurations of the architecture, we use it to test our modeling framework on the CoMD proxy application for strong and weak scaling results. We also study the effect of dominant

MPI communication calls and their behavior on a scale of millions of nodes.

5.2.1 Evaluation of DOE's exascale homogeneous architecture

We first test the theoretical peak performance of the proxy exascale architecture in order to see whether it matches with those proposed by the Exascale Computing Project [117]. Table 5.2 shows the values obtained by micro-benchmarks provided by Aspen and the extensions that we implemented in the domain specific language. DOE proposes the theoretical Flops of the exascale machine to be ∼1-ExaFlops. Our proxy architecture shows the single-precision and double precision flops to be 1.47 & 0.736 ExaFlops respectively. While implementing real hardware, a number of factors will decide the true configuration of the system, and therefore system configurations (i.e., number of cores, number of sockets) might have different configurations when implementing at exascale. We also calculate the energy consumed to produce one single and double precision ExaFlops. The maximum bandwidth provided by memory is $\sim 592GB$ as proposed by the Exascale Computing Project. As can be seen from the Table 5.2, the theoretical peak performance matches the one proposed by the Exascale Computing Project, therefore we can test it for the execution of the application.

One might argue that the future exascale architectures will likely be heterogeneous containing multiple types of accelerators while our analysis in this chapter is focused on homogeneous architectures. Though our study focuses on homogeneous exascale architecture we wanted to convey two observations. First, regardless of the scale, homogeneous CPU based systems will play a key role in the exascale system and their evaluation using Aspen shows they provide the capabilities proposed by the Exascale Computing Project. Secondly, we wanted to underscore Aspen's potential in terms of implementing and testing future architectures.

Figure 5.5 shows the Roofline analysis of the proxy homogeneous architecture implemented in

Table 5.2: Theoretical peak validations of exascale proxy architecture provided by Aspen

Parameters	Aspen
SP Flops	∼1.47 ExaFlops
DP Flops	∼0.736 ExaFlops
SP/Joules	∼10833.3 GFlops/Joule
DP/Joules	∼5416 GFlops/Joule
Memory Bandwidth	∼592 GB
Memory Bandwidth/Joules	∼1.1636 GB/Joule

Aspen. The Roofline model defines the upper limit on kernel performance P_k with the equation

$$P_k = min(P_f, \beta A_i) \tag{5.8}$$

where P_f is the maximum floating point operation provided by the system, β is the maximum theoretical bandwidth provided by the system, and A_i is the theoretical Flops to Byte ratio of the system. Identifying the Flops to Byte ratio and achievable GFlops of an application and comparing it with the Roofline plot helps to find limits of the system and whether any application will be compute or memory bound. Our analysis of CoMD using the Roofline model shows that CoMD behaves as a compute intensive application on the theoretical homogeneous exascale architecture.

We also studied the effect of increasing the number of messages with respect to the runtime. Figure 5.6 shows the effect of increasing message sizes on the overall latency of the system. This information is particularly useful as it will play a key role in deciding the bottlenecks of the machine and predicting the communication time of the application. Our analysis shows that considering the scale of the system, we need more efficient interconnects that can furthur reduce the latency incurred by the system.

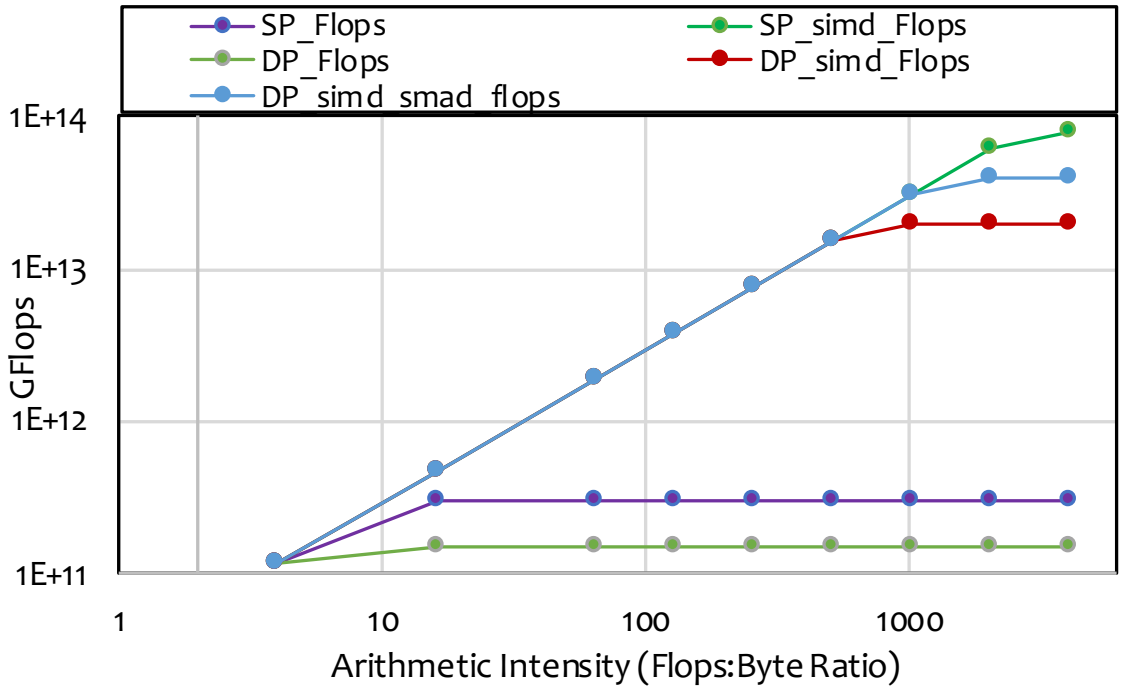


Figure 5.5: Roofline analysis of proxy homogeneous exascale architecture

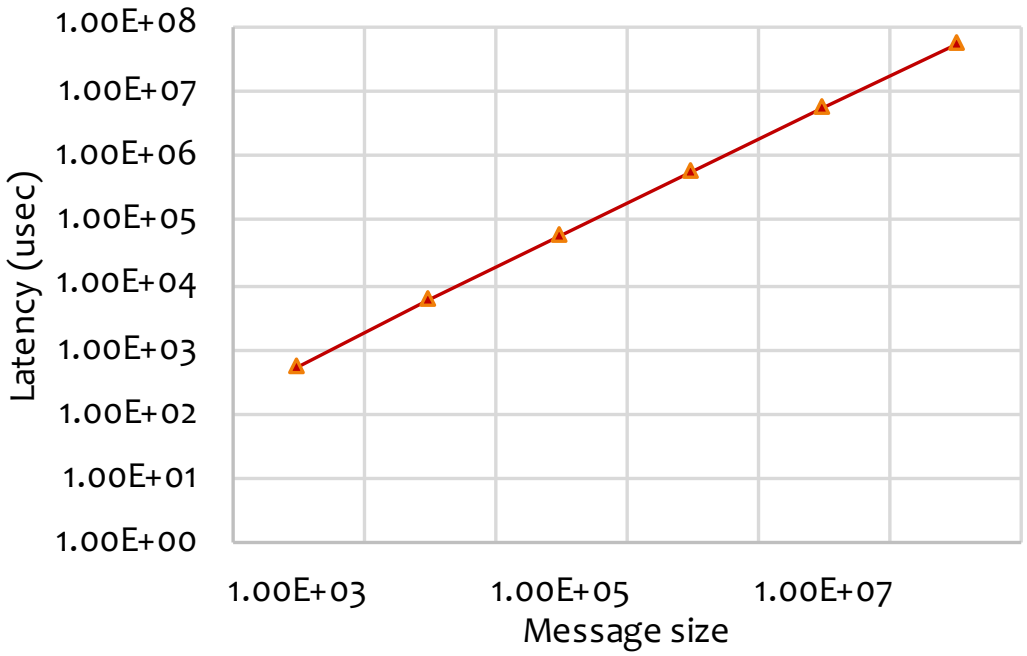


Figure 5.6: Effect on latency with increasing message size

5.2.2 Communication and Computation modeling of CoMD

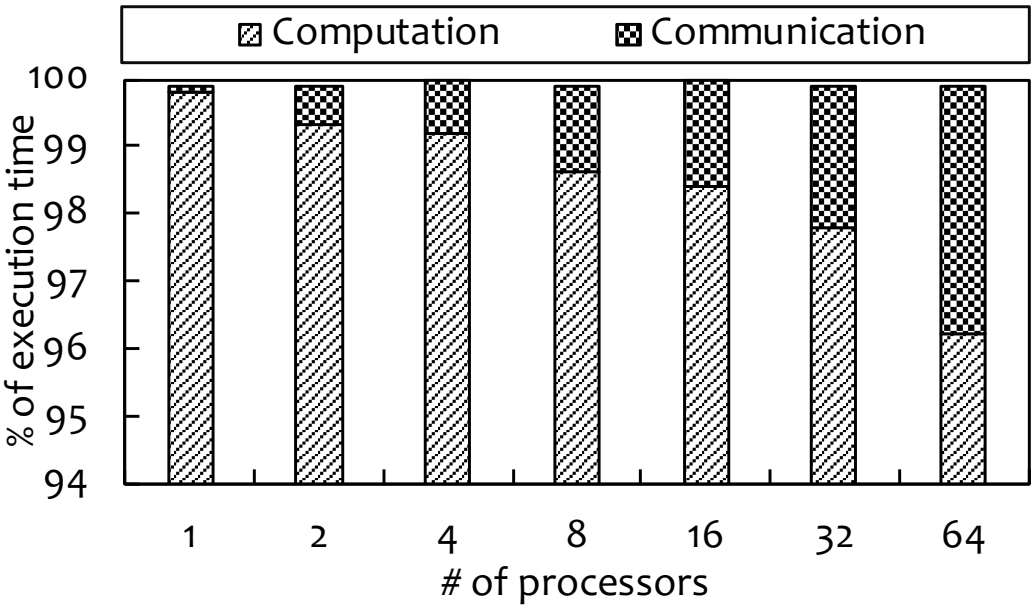


Figure 5.7: Profiling of strong scaling of CoMD using Intel's Trace Analyzer and Collector

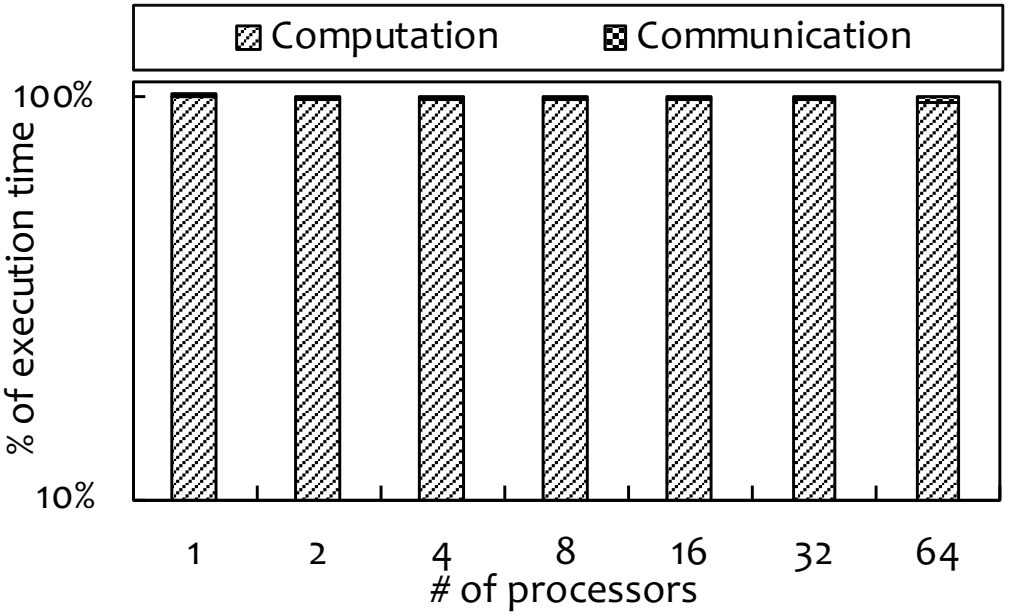


Figure 5.8: Profiling of weak scaling of CoMD using Intel's Trace Analyzer and Collector

We executed CoMD on the proxy homogeneous exascale architecture to study the communication and computation effect on the overall application execution. We calculated the Flops per Byte ratio and theoretical Flops of CoMD; we compared this information with the Roofline of the system (Figure 5.5). Our analysis shows that CoMD is a compute intensive application for the processes within the node, while communication overhead starts to emerge for executions between a number of nodes. In order to verify this insight, we profiled CoMD on NERSC's Cori using Intel's Trace Analyzer and Collector for strong and weak scaling. Figure 5.7 and Figure 5.8 show the distribution between computation and communication on strong and weak scaling of CoMD using Intel's Trace Analyzer and Collector. The computation plays a major part in the overall application execution time for strong scaling, while the communication effects are not very dominant in the case of weak scaling.

We also profiled the CoMD communication patterns in Aspen in order to find out the dominant communication routines using Intel's Trace Analyzer and Collector. $MPI\_Barrier$, MPI_Bcast and $MPI\_Allreduce$ are the communication routines that take a significant amount of time during the communication phase of CoMD. While MPI_Bcast is used to communicate the position and velocity of the atom at runtime, and it is supported by $MPI\_Barrier$ in order to ensure a reliable communication. The assurance of reliability is not an issue on a small cluster, but considering the scale of future architectures, some optimization strategies need to be applied at runtime before issuing MPI_Bcast. Similar situations exist for $MPI\_Allreduce$ where the calculations from individual processes are gathered to produce a converged result. Since $MPI\_Allreduce$ is targeting millions of processing elements for future architectures, some optimization to break down the $MPI\_Allreduce$ function into phases can help to avoid the overhead at exascale. One possible option is to introduce a virtual hierarchy in the cluster to parallelize the barrier function, it will also aid in recovering from failure to synchronize at larger scale. A failure to synchronize at larger scale will be more detrimental as compared to

a small subset of the system, and recovery will be faster.

Figures 5.9 and 5.10 show the strong and weak scaling for $8-Million$ processes. In the case of strong scaling, execution time continues to decrease with increasing number of processes until the point where we experience intra-node communication overhead which causes an increase in execution time. In terms of weak scaling, execution time continues to increase with increasing processes as the amount of work along with communication costs increase with increasing processes.

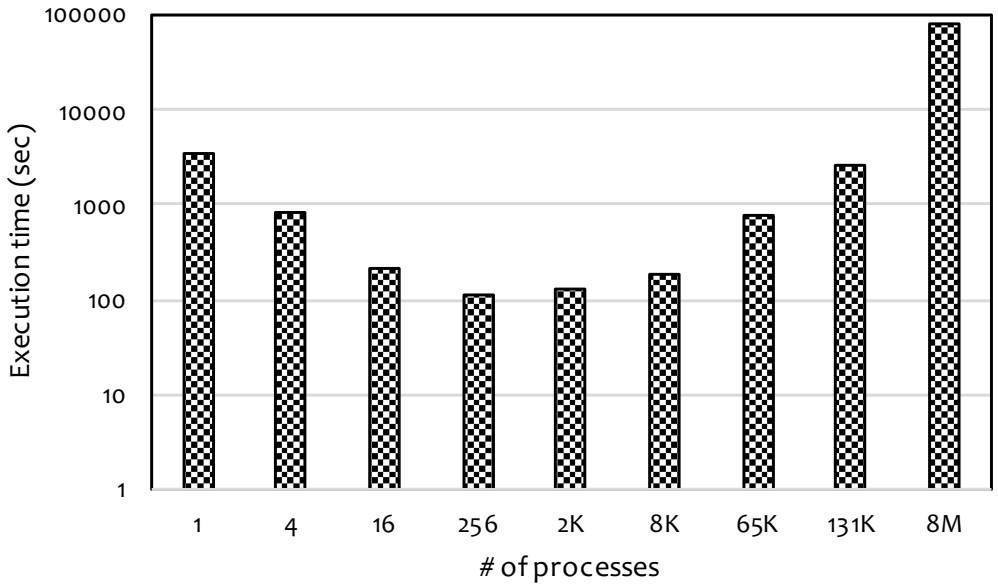


Figure 5.9: Strong Scaling on 8 Million processes

Figure 5.11 shows the comparison between measured and predicted values of CoMD for the strong scaling case for an input size of 64000 atoms. Measured values were obtained from NERSC's Cori leadership computing facility. The two most time consuming components of CoMD are *position calculation* kernel and the *communication* between processes. As before, we under predict as the model does not take into account the system noise in the background. As CoMD's execution time improves with increasing number of processes, increased processor

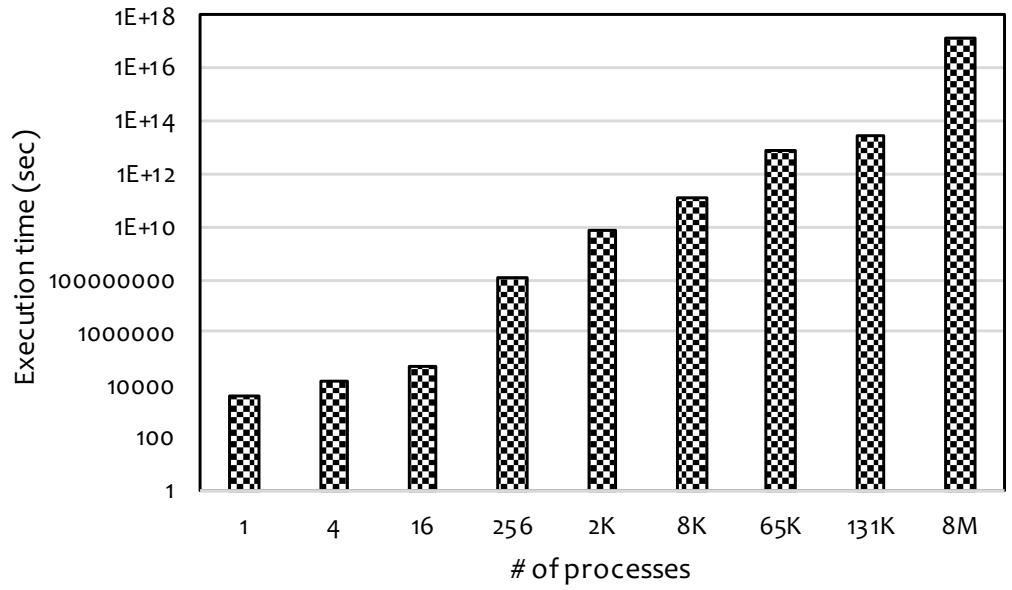


Figure 5.10: Weak Scaling on 8 Million processes

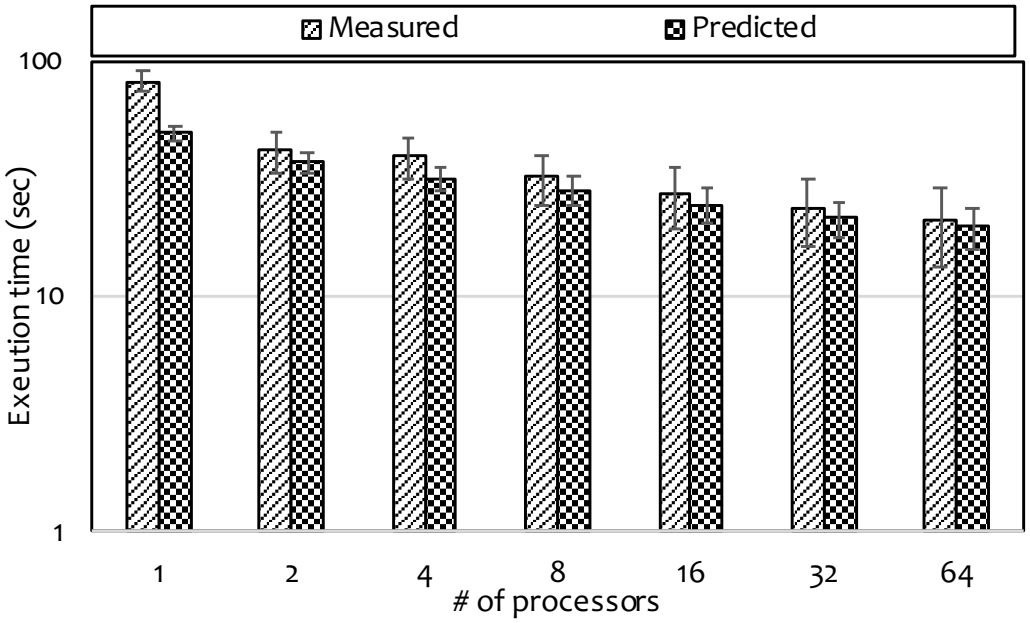


Figure 5.11: Comparison between measured and predicted values for strong scaling of CoMD

allocation is one technique to improve the application execution. This holds until the point where communication overhead is greater than the improvement achieved with increasing processes. In the case of exascale homogeneous architecture, it is important to consider the

number of processes with respect to input size, as allocating more processes than required will result in over-subscription, and provides no benefit.

Our communication and computation model tests the application with increasing number of processes to see the optimal number of ranks for a given execution environment. Figure 5.12 shows the comparison between the values measured at NERSC Cori and the predicted values on the proxy exascale architecture implemented in Aspen. Since the number of messages transmitted during each iteration increases with increasing processes in weak scaling, communication overhead increases resulting in increased runtime and we do not observe any performance improvement in weak scaling. The effect of weak scaling in measured values tracks closely with the predicted values in Aspen.

Programming languages and models can also be used to take advantage of the data locality in complex memory architectures expected in exascale systems. Some notable directive based programming models(e.g, Lee et al. [98]) mention the available but yet to be explored ventures of programming to improve memory bandwidth performance and to improve performance of CoMD like nearest-neighbor applications for weak scaling.

5.3 Conclusions and Future Work

The exascale architecture implementation is still in its infancy, but a group of abstract machine models are proposed to represent the prospective architectures of the future. In this chapter, we implement, validate and use one of the homogeneous architectures proposed by the Exascale Computing Project in Aspen DSL. We evaluate the proxy architecture to test for theoretical peaks. We implement a communication and computation mapping model in Aspen to test it for exascale proxy application and architecture. We also study the strong and weak scaling effects of the application characteristics of CoMD on communication and computation at the

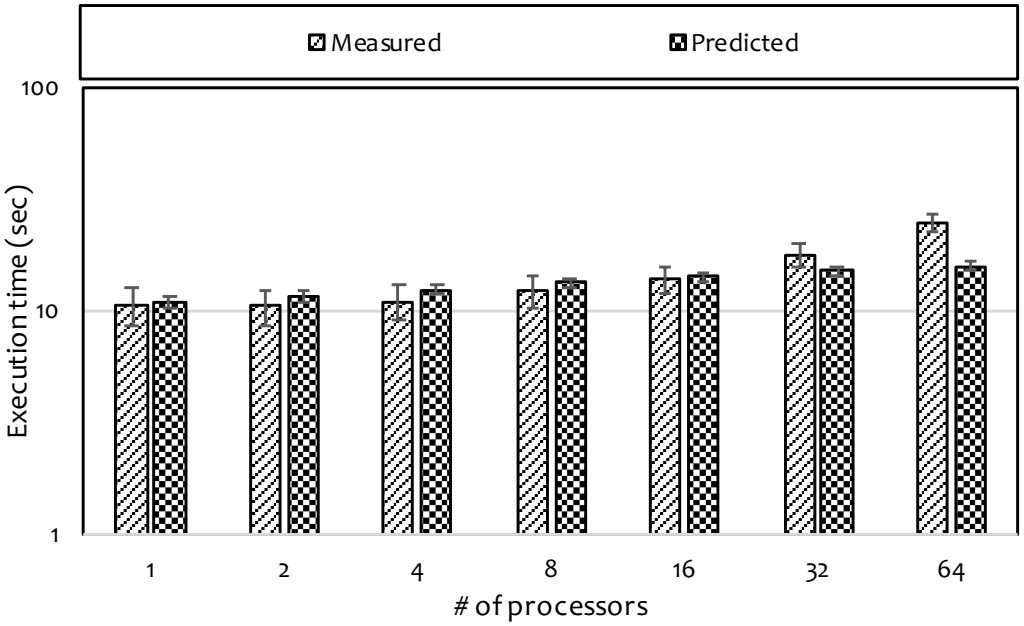


Figure 5.12: Comparison between measured and predicted values for weak scaling of CoMD

scale of future architectures.

In the next chapter, we present a summary of this thesis and present a few directions in which we plan to expand this work in the future.

Chapter 6

Summary and Future Work

6.1 Summary of Dissertation

This thesis presents theory, techniques and details for modeling performance and energy as the primary design constraints for current and future computing systems. Since increasing performance by traditional means has limits (i.e., increasing number of cores, increasing system clock frequency), domain scientists and architects have to redesign the optimization paradigm by proposing scalable and accurate models to predict the performance and energy of the applications. Moreover, with the expected scale of the system, the conventional trial and error modeling techniques are no longer viable, hence motivating the need for automated modeling techniques. In this thesis, we demonstrate the use of co-design modeling to evaluate prevailing exascale designs.

We propose two techniques: one as an extension of Aspen for power and energy and a workflow framework that supports automation of design space exploration. We provide the following contributions in terms of scalable and accurate performance/energy modeling and design-

space exploration of future architectures:

- A system-level performance and energy estimation and modeling framework for Aspen that uses activity factors of the system's components to calculate the time and energy consumed by the application,
- A performance and energy modeling framework that provide the guidelines for device selection in Aspen for a heterogeneous architecture that presents evidence of performance and energy estimations using performance counters and statistical modeling techniques,
- A framework that enables automatic generation of abstract application and machine models using Aspen,
- A design space exploration framework that analyzes the sensitivity and scalability analysis of candidate proxy applications on reconfigurable architectures, at scale,
- Implementation of a homogeneous proxy exascale architecture in Aspen to test theoretical peaks of the system as well as the communication and computation analysis of an exascale proxy application at a scale,

6.2 Future work

There are a number of research directions we are considering.

6.2.1 Analyze the Impact of Memory Types

As mentioned by Ang et al. [8], the scale and complexity of exascale architectures require multiple hierarchies of memory. For example, High Bandwidth Memory (HBM) on-package

memory contains stacked 2.5D or 3D layers of DRAM and connects them using Through Silicon Vias (TSV) to achieve high bandwidth. Though HBM provides high bandwidth and improved energy efficiency, it lacks the high capacity required by the complexity and scale of exascale systems. Non-Volatile Memory (NVM), on the other hand provides higher capacities compared with HBM, and may be employed as an off-chip memory combined with HBM as the on-package memory. Aspen is well suited to evaluate a number of memory types and to quantify their advantages over each other at scale.

6.2.2 Implementation of Proxy Exascale Architectures using Aspen

A selected list of four abstract machine models are proposed by Ang et al. [8]. The list includes a homogeneous, multi-core CPU and discrete accelerator model, integrated CPU and accelerator model, and heterogeneous multi-core models. In this thesis, we implemented the homogeneous model in Aspen and discussed the theoretical limits of the architecture. One future direction of this work is to implement all the proposed abstract models in Aspen and classify them by class of application based on their theoretical limits.

6.2.3 Evaluation of Interconnect Technologies for Exascale Computing

Different interconnect technologies are under investigation for deployment in exascale architectures. As mentioned by Ang et al. [8], a system with thin nodes will be more affected by the interconnect as compared to an exascale system with fat nodes. An analytical-model based prediction of different interconnect technologies (e.g., fatTree, dragonFly etc,) and its validation with simulated results will help to make an informed decision by architects during exascale system design.

6.2.4 Topology-Aware Job Placement Schemes for Exascale

Since future exascale architectures are estimated to comprise millions of nodes, parallel job placement and mapping schemes for different network topologies are a critical issue. We believe that because of the scale of future architectures, most job placements will be based on topology. Efficient graph mapping and partitioning schemes that efficiently distribute the job based on rank preference, data availability, deployed memory hierarchy show promise. Moreover, an efficient scalable graph partitioning mechanism would help to mitigate slow interconnect issues by intelligent job placements at runtime. More efficient mapping techniques that better exploit data locality at the scale of million of nodes are worthy of future study.

Bibliography

[1] "Keeneland Projects," http://keeneland.gatech.edu/, 2016.

[2] "NVIDIA Management Library (NVML)," 2017. [Online]. Available: https://developer.nvidia.com/nvidia-management-library-nvml

[3] L. Adhianto, S. Banerjee, M. Fagan, M. Krentel, G. Marin, J. Mellor-Crummey, and N. R. Tallent, "Hpctoolkit: Tools for performance analysis of optimized parallel programs," *Concurrency and Computation: Practice and Experience*, vol. 22, no. 6, pp. 685–701, 2010.

[4] K. Ahmed, M. Obaida, J. Liu, S. Eidenbenz, N. Santhi, and G. Chapuis, "An integrated interconnection network model for large-scale performance prediction," in *Proceedings of the 2016 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. ACM, 2016, pp. 177–187.

[5] A. Alexandrov, M. F. Ionescu, K. E. Schauser, and C. Scheiman, "Loggp: incorporating long messages into the logp modelâĂŤone step closer towards a realistic model for parallel computation," in *Proceedings of the seventh annual ACM symposium on Parallel algorithms and architectures*. ACM, 1995, pp. 95–105.

[6] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the April 18-20, 1967, spring joint computer conference*. ACM, 1967, pp. 483–485.

[7] J. Ang, K. Bergman, S. Borkar, W. Carlson, L. Carrington, G. Chiu, R. Colwell, W. Dally, J. Dongarra, A. Geist *et al.*, "Top ten exascale research challenges," *DOE Office of Science, Advanced Scientific Computing Advisory Committee, Subcommittee for the Top Ten Exascale Research Challenges*, 2014.

[8] J. A. Ang, R. F. Barrett, R. E. Benner, D. Burke, C. Chan, J. Cook, D. Donofrio, S. D. Hammond, K. S. Hemmert, S. Kelly *et al.*, "Abstract machine models and proxy architectures for exascale computing," in *Hardware-Software Co-Design for High Performance Computing (Co-HPC), 2014*. IEEE, 2014, pp. 25–32.

[9] K. Antypas, N. Wright, N. P. Cardo, A. Andrews, and M. Cordery, "Cori: a Cray XC pre-exascale system for NERSC," *Cray User Group*, 2014.

[10] A. H. Ashouri, G. Mariani, G. Palermo, E. Park, J. Cavazos, and C. Silvano, "Cobayn: Compiler autotuning framework using bayesian networks," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 13, no. 2, p. 21, 2016.

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

[12] R. F. Barrett, X. S. Hu, S. S. Dosanjh, S. Parker, M. A. Heroux, and J. Shalf, "Toward codesign in high performance computing systems," in *Proceedings of the International Conference on Computer-Aided Design*. ACM, 2012, pp. 443–449.

[13] R. F. Barrett, M. A. Heroux, P. Lin, C. T. Vaughan, and A. B. Williams, "Copy of Mini-applications: Vehicles for Co-Design," Sandia National Laboratories (SNL-NM), NM, USA, Tech. Rep., 2011.

[14] A. Baumker and W. Dittrich, "Fully dynamic search trees for an extension of the bsp model," in *Proceedings of the eighth annual ACM symposium on Parallel algorithms and architectures*. ACM, 1996, pp. 233–242.

[15] N. Beckmann and D. Sanchez, "Modeling cache performance beyond lru," in *High Performance Computer Architecture (HPCA), 2016 IEEE International Symposium on*. IEEE, 2016, pp. 225–236.

[16] F. Bellosa, "The benefits of event: driven energy accounting in power-sensitive systems," in *Proceedings of the 9th workshop on ACM SIGOPS European workshop: beyond the PC: new challenges for the operating system*. ACM, 2000, pp. 37–42.

[17] R. Bergamaschi, G. Han, A. Buyuktosunoglu, H. Patel, I. Nair, G. Dittmann, G. Janssen, N. Dhanwada, Z. Hu, P. Bose *et al.*, "Exploring power management in multi-core systems," in *Proceedings of the 2008 Asia and South Pacific Design Automation Conference*. IEEE Computer Society Press, 2008, pp. 708–713.

[18] C. L. Bertin and E. L. Hedberg, "High performance, high bandwidth memory bus architecture utilizing sdrams," Feb. 9 1999, uS Patent 5,870,350.

[19] J. C. Bezdek, R. Ehrlich, and W. Full, "Fcm: The fuzzy c-means clustering algorithm," *Computers & Geosciences*, vol. 10, no. 2-3, pp. 191–203, 1984.

[20] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti *et al.*, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.

[21] N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi, and S. K. Reinhardt, "The m5 simulator: Modeling networked systems," *IEEE Micro*, vol. 26, no. 4, pp. 52–60, 2006.

[22] E. A. Brewer, C. N. Dellarocas, A. Colbrook, and W. E. Weihl, "Proteus: A high-performance parallel-architecture simulator," 1991.

[23] D. Brooks, P. Bose, V. Srinivasan, M. K. Gschwind, P. G. Emma, and M. G. Rosenfield, "New methodology for early-stage, microarchitecture-level power-performance analysis of microprocessors," *IBM Journal of Research and Development*, vol. 47, no. 5.6, pp. 653–670, 2003.

[24] D. Brooks, V. Tiwari, and M. Martonosi, *Wattch: A framework for architectural-level power analysis and optimizations*. ACM, 2000, vol. 28, no. 2.

[25] D. Burger and T. M. Austin, "The simplescalar tool set, version 2.0," *ACM SIGARCH computer architecture news*, vol. 25, no. 3, pp. 13–25, 1997.

[26] K. W. Cameron, R. Ge, and X. Feng, "High-performance, power-aware distributed computing for scientific applications," *Computer*, vol. 38, no. 11, pp. 40–47, 2005.

[27] C. D. Carothers, J. S. Meredith, M. P. Blanco, J. S. Vetter, M. Mubarak, J. LaPre, and S. Moore, "Durango: Scalable Synthetic Workload Generation for Extreme-Scale Application Performance Modeling and Simulation," in *ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, 2017.

[28] J. Cavazos, G. Fursin, F. Agakov, E. Bonilla, M. F. O'Boyle, and O. Temam, "Rapidly selecting good compiler optimizations using performance counters," in *Proceedings of the International Symposium on Code Generation and Optimization*. IEEE Computer Society, 2007, pp. 185–197.

[29] A. Chan, P. Balaji, W. Gropp, and R. Thakur, "Communication analysis of parallel 3d fft for flat cartesian meshes on large blue gene systems," in *International Conference on High-Performance Computing*. Springer, 2008, pp. 350–364.

[30] C. Chan, D. Unat, M. Lijewski, W. Zhang, J. Bell, and J. Shalf, "Software design space exploration for exascale combustion co-design," in *International Supercomputing Conference*. Springer, 2013, pp. 196–212.

[31] J. W. Choi, D. Bedard, R. Fowler, and R. Vuduc, "A roofline model of energy," in *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*. IEEE, 2013, pp. 661–672.

[32] P. Conway and B. Hughes, "The amd opteron northbridge architecture," *IEEE Micro*, vol. 27, no. 2, 2007.

[33] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex fourier series," *Mathematics of computation*, vol. 19, no. 90, pp. 297–301, 1965.

[34] J. Cope, N. Liu, S. Lang, P. Carns, C. Carothers, and R. Ross, "Codes: Enabling co-design of multilayer exascale storage architectures," in *Proceedings of the Workshop on Emerging Supercomputing Technologies*, vol. 2011, 2011.

[35] D. E. Culler, R. M. Karp, D. Patterson, A. Sahay, E. E. Santos, K. E. Schauser, R. Subramonian, and T. von Eicken, "Logp: A practical model of parallel computation," *Communications of the ACM*, vol. 39, no. 11, pp. 78–85, 1996.

[36] M. Curtis-Maury, A. Shah, F. Blagojevic, D. S. Nikolopoulos, B. R. de 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*. ACM, 2008, pp. 250–259.

[37] M. Curtis-Maury, K. Singh, S. A. McKee, F. Blagojevic, D. S. Nikolopoulos, B. R. De Supinski, and M. Schulz, "Identifying energy-efficient concurrency levels using machine learning," in *Cluster Computing, 2007 IEEE International Conference on*. IEEE, 2007, pp. 488–495.

[38] K. Czechowski, C. Battaglino, C. McClanahan, K. Iyer, P.-K. Yeung, and R. Vuduc, "On the communication complexity of 3d ffts and its implications for exascale," in *Proceedings of the 26th ACM international conference on Supercomputing*. ACM, 2012, pp. 205–214.

[39] P. De la Torre and C. P. Kruskal, "Submachine locality in the bulk synchronous setting," in *European Conference on Parallel Processing*. Springer, 1996, pp. 352–358.

[40] E. Deelman, C. Carothers, A. Mandal, B. Tierney, J. S. Vetter, I. Baldin, C. Castillo, G. Juve, D. Król, V. Lynch *et al.*, "Panorama: an approach to performance modeling and diagnosis of extreme-scale workflows," *The International Journal of High Performance Computing Applications*, vol. 31, no. 1, pp. 4–18, 2017.

[41] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good *et al.*, "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," *Scientific Programming*, vol. 13, no. 3, pp. 219–237, 2005.

[42] J. Demmel, M. Hoemmen, M. Mohiyuddin, and K. Yelick, "Avoiding communication in sparse matrix computations," in *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*. IEEE, 2008, pp. 1–12.

[43] Q. Deng, D. Meisner, L. Ramos, T. F. Wenisch, and R. Bianchini, "Memscale: active low-power modes for main memory," in *ACM SIGPLAN Notices*, vol. 46, no. 3. ACM, 2011, pp. 225–238.

[44] J. Dongarra, P. Beckman, T. Moore, P. Aerts, G. Aloisio, J.-C. Andre, D. Barkai, J.-Y. Berthou, T. Boku, B. Braunschweig *et al.*, "The international exascale software project roadmap," *International Journal of High Performance Computing Applications*, vol. 25, no. 1, pp. 3–60, 2011.

[45] J. Dongarra, H. Ltaief, P. Luszczek, and V. M. Weaver, "Energy footprint of advanced dense numerical linear algebra using tile algorithms on multicore architectures," in *Cloud and Green Computing (CGC), 2012 Second International Conference on*. IEEE, 2012, pp. 274–281.

[46] J. Dongarra, H. Ltaief, P. Luszczek, and V. M. Weaver, "Energy footprint of advanced dense numerical linear algebra using tile algorithms on multicore architectures," in *Cloud and Green Computing (CGC), 2012 Second International Conference on*. IEEE, 2012, pp. 274–281.

[47] M. Dorr, N. Barton, J. Keasler, and F. Li, "CoEVP: A Co-Design Embedded ViscoPlasticity Scale-bridging Proxy App for ExMatEx," *LLNL, TR LLNL-SM-655180*, 2014.

[48] S. S. Dosanjh, R. F. Barrett, D. Doerfler, S. D. Hammond, K. S. Hemmert, M. A. Heroux, P. T. Lin, K. T. Pedretti, A. F. Rodrigues, T. Trucano *et al.*, "Exascale design space exploration and co-design," *Future Generation Computer Systems*, vol. 30, pp. 46–58, 2014.

[49] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *ACM SIGARCH Computer Architecture News*, vol. 35, no. 2. ACM, 2007, pp. 13–23.

[50] A. Faraj, S. Kumar, B. Smith, A. Mamidala, and J. Gunnels, "Mpi collective communications on the blue gene/p supercomputer: Algorithms and optimizations," in *High Performance Interconnects, 2009. HOTI 2009. 17th IEEE Symposium on*. IEEE, 2009, pp. 63–72.

[51] R. Filgueira, D. E. Singh, J. Carretero, A. Calderón, and F. García, "Adaptive-compi: Enhancing mpi-based applicationsâĂŹ performance and scalability by using adaptive compression," *The International Journal of High Performance Computing Applications*, vol. 25, no. 1, pp. 93–114, 2011.

[52] V. W. Freeh, N. Kappiah, D. K. Lowenthal, and T. K. Bletsch, "Just-in-time dynamic voltage scaling: Exploiting inter-node slack to save energy in mpi programs," *Journal of Parallel and Distributed Computing*, vol. 68, no. 9, pp. 1175–1185, 2008.

[53] 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," in *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*. IEEE, 2005, pp. 10–pp.

[54] H. Gahvari and W. Gropp, "An introductory exascale feasibility study for ffts and multigrid," in *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*. IEEE, 2010, pp. 1–9.

[55] H. Gahvari and W. Gropp, "An introductory exascale feasibility study for ffts and multigrid," in *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*. IEEE, 2010, pp. 1–9.

[56] M. Gamell, I. Rodero, M. Parashar, J. C. Bennett, H. Kolla, J. Chen, P.-T. Bremer, A. G. Landge, A. Gyulassy, P. McCormick *et al.*, "Exploring power behaviors and trade-offs of in-situ data analytics," in *High Performance Computing, Networking, Storage and Analysis (SC), 2013 International Conference for*. IEEE, 2013, pp. 1–12.

[57] M. Gamell, I. Rodero, M. Parashar, and S. Poole, "Exploring energy and performance behaviors of data-intensive scientific workflows on systems with deep memory hierarchies," in *High Performance Computing (HiPC), 2013 20th International Conference on*. IEEE, 2013, pp. 226–235.

[58] R. Ge and K. W. Cameron, "Power-aware speedup," in *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*. IEEE, 2007, pp. 1–10.

[59] R. Ge, X. Feng, and K. W. Cameron, "Improvement of power-performance efficiency for high-end computing," in *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*. IEEE, 2005, pp. 8–pp.

[60] R. Ge, X. Feng, and K. W. Cameron, "Performance-constrained distributed dvs scheduling for scientific applications on power-aware clusters," in *Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference*. IEEE, 2005, pp. 34–34.

[61] R. Ge, X. Feng, and K. W. Cameron, "Modeling and evaluating energy-performance efficiency of parallel processing on multicore based power aware systems," in *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*. IEEE, 2009, pp. 1–8.

[62] R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, and K. W. Cameron, "Powerpack: Energy profiling and analysis of high-performance systems and applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 5, pp. 658–671, 2010.

[63] R. Ge, P. Zou, and X. Feng, "Application-aware power coordination on power bounded numa multicore systems," in *Parallel Processing (ICPP), 2017 46th International Conference on*. IEEE, 2017, pp. 591–600.

[64] N. Goswami, B. Cao, and T. Li, "Power-performance co-optimization of throughput core architecture using resistive memory," in *High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on*. IEEE, 2013, pp. 342–353.

[65] G. Goumas, A. Sotiropoulos, and N. Koziris, "Minimizing completion time for loop tiling with computation and communication overlapping," in *Parallel and Distributed Processing Symposium., Proceedings 15th International*. IEEE, 2001, pp. 10–pp.

[66] S. L. Graham, P. B. Kessler, and M. K. Mckusick, "Gprof: A call graph execution profiler," in *ACM Sigplan Notices*, vol. 17, no. 6. ACM, 1982, pp. 120–126.

[67] S. Gunther, A. Deval, T. Burton, and R. Kumar, "Energy-efficient computing: Power management system on the nehalem family of processors." *Intel Technology Journal*, vol. 14, no. 3, 2010.

[68] S. Gurumurthi, A. Sivasubramaniam, M. J. Irwin, N. Vijaykrishnan, and M. Kandemir, "Using complete machine simulation for software power estimation: The softwatt approach," in *High-Performance Computer Architecture, 2002. Proceedings. Eighth International Symposium on*. IEEE, 2002, pp. 141–150.

[69] J. L. Gustafson, "Reevaluating amdahl's law," *Communications of the ACM*, vol. 31, no. 5, pp. 532–533, 1988.

[70] M. Hähnel, B. Döbel, M. Völp, and H. Härtig, "Measuring energy consumption for short code paths using rapl," *ACM SIGMETRICS Performance Evaluation Review*, vol. 40, no. 3, pp. 13–17, 2012.

[71] M. Halper, "Supercomputing's Super Energy Needs, and What to Do About Them," *Communications of the ACM*, 2015.

[72] T. Hamano, T. Endo, and S. Matsuoka, "Power-aware dynamic task scheduling for heterogeneous accelerated clusters," in *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*. IEEE, 2009, pp. 1–8.

[73] P. Hammarlund, A. J. Martinez, A. A. Bajwa, D. L. Hill, E. Hallnor, H. Jiang, M. Dixon, M. Derr, M. Hunsaker, R. Kumar *et al.*, "Haswell: The fourth-generation intel core processor," *IEEE Micro*, vol. 34, no. 2, pp. 6–20, 2014.

[74] T. Hoefler, W. Gropp, W. Kramer, and M. Snir, "Performance modeling for systematic performance tuning," in *State of the Practice Reports*. ACM, 2011, p. 6.

[75] T. Hoefler, T. Mehlan, F. Mietke, and W. Rehm, "Logfp-a model for small messages in infiniband," in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*. IEEE, 2006, pp. 6–pp.

[76] S. Hong and H. Kim, "An analytical model for a gpu architecture with memory-level and thread-level parallelism awareness," in *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3. ACM, 2009, pp. 152–163.

[77] S. Hong and H. Kim, "An integrated gpu power and performance model," in *ACM SIGARCH Computer Architecture News*, vol. 38, no. 3. ACM, 2010, pp. 280–289.

[78] S. Hong and H. Kim, "An integrated gpu power and performance model," in *ACM SIGARCH Computer Architecture News*, vol. 38, no. 3. ACM, 2010, pp. 280–289.

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

[80] C. Isci and M. Martonosi, "Runtime power monitoring in high-end processors: Methodology and empirical data," in *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 2003, p. 93.

[81] T. Z. Islam, J. J. Thiagarajan, A. Bhatele, M. Schulz, and T. Gamblin, "A machine learning framework for performance coverage analysis of proxy applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Press, 2016, p. 46.

[82] C. L. Janssen, H. Adalsteinsson, and J. P. Kenny, "Using simulation to design extremescale applications and architectures: programming model exploration," *ACM SIGMETRICS Performance Evaluation Review*, vol. 38, no. 4, pp. 4–8, 2011.

[83] S. A. Jarvis, D. P. Spooner, H. N. L. C. Keung, J. Cao, S. Saini, and G. R. Nudd, "Performance prediction and its use in parallel and distributed computing systems," *Future Generation Computer Systems*, vol. 22, no. 7, pp. 745–754, 2006.

[84] R. E. Jensen, I. Karlin, and A. C. Elster, "Auto-tuning a matrix routine for high performance," *Norsk informatikkonferanse*, vol. 2011, 2011.

[85] N. Jiang, G. Michelogiannakis, D. Becker, B. Towles, and W. J. Dally, "Booksim 2.0 userâĂŹs guide," *Standford University*, 2010.

[86] R. Joseph and M. Martonosi, "Run-time power estimation in high performance microprocessors," in *Proceedings of the 2001 international symposium on Low power electronics and design*. ACM, 2001, pp. 135–140.

[87] I. Karlin, A. Bhatele, J. Keasler, B. L. Chamberlain, J. Cohen, Z. Devito, R. Haque, D. Laney, E. Luke, F. Wang *et al.*, "Exploring traditional and emerging parallel programming models using a proxy application," in *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*. IEEE, 2013, pp. 919–932.

[88] K. Kasichayanula, D. Terpstra, P. Luszczek, S. Tomov, S. Moore, and G. D. Peterson, "Power aware computing on gpus," in *Application Accelerators in High Performance Computing (SAAHPC), 2012 Symposium on*. IEEE, 2012, pp. 64–73.

[89] D. Kerbyson, A. Vishnu, K. Barker, and A. Hoisie, "Codesign challenges for exascale systems: Performance, power, and reliability," *Computer*, vol. 44, no. 11, pp. 37–43, 2011.

[90] G. Kestor, R. Gioiosa, D. J. Kerbyson, and A. Hoisie, "Enabling accurate power profiling of hpc applications on exascale systems," in *Proceedings of the 3rd International Workshop on Runtime and Operating Systems for Supercomputers*. ACM, 2013, p. 4.

[91] K. H. Kim, R. Buyya, and J. Kim, "Power aware scheduling of bag-of-tasks applications with deadline constraints on dvs-enabled clusters." in *CCGrid*, vol. 7, 2007, pp. 541–548.

[92] R. Koller, A. Verma, and A. Neogi, "Wattapp: an application aware power meter for shared data centers," in *Proceedings of the 7th international conference on Autonomic computing*. ACM, 2010, pp. 31–40.

[93] C. D. Krieger, M. M. Strout, J. Roelofs, and A. Bajwa, "Executing optimized irregular applications using task graphs within existing parallel models," in *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*. IEEE, 2012, pp. 261–268.

[94] S. Kulkarni and J. Cavazos, "Mitigating the compiler optimization phase-ordering problem using machine learning," *ACM SIGPLAN Notices*, vol. 47, no. 10, pp. 147–162, 2012.

[95] S. Kumar, Y. Sabharwal, R. Garg, and P. Heidelberger, "Optimization of all-to-all communication on the blue gene/l supercomputer," in *Parallel Processing, 2008. ICPP'08. 37th International Conference on*. IEEE, 2008, pp. 320–329.

[96] J. H. Lee, J. Meng, and H. Kim, "Sesh framework: A space exploration framework for gpu application and hardware codesign," in *International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems*. Springer, 2013, pp. 182–202.

[97] S. Lee, J. S. Meredith, and J. S. Vetter, "Compass: A framework for automated performance modeling and prediction," in *Proceedings of the 29th ACM on International Conference on Supercomputing*. ACM, 2015, pp. 405–414.

[98] S. Lee and J. S. Vetter, "Openarc: Open accelerator research compiler for directive-based, efficient heterogeneous computing," in *Proceedings of the 23rd international symposium on High-performance parallel and distributed computing*. ACM, 2014, pp. 115–120.

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

[100] M. Y. Lim, V. W. Freeh, and D. K. Lowenthal, "Adaptive, transparent frequency and voltage scaling of communication phases in mpi programs," in *SC 2006 conference, proceedings of the ACM/IEEE*. IEEE, 2006, pp. 14–14.

[101] C. Liu, J. Li, W. Huang, J. Rubio, E. Speight, and X. Lin, "Power-efficient time-sensitive mapping in heterogeneous systems," in *Proceedings of the 21st international conference on Parallel architectures and compilation techniques*. ACM, 2012, pp. 23–32.

[102] C.-K. Luk, S. Hong, and H. Kim, "Qilin: exploiting parallelism on heterogeneous multiprocessors with adaptive mapping," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2009, pp. 45–55.

[103] C. Luo and R. Suda, "A performance and energy consumption analytical model for gpu," in *Dependable, autonomic and secure computing (dasc), 2011 ieee ninth international conference on*. IEEE, 2011, pp. 658–665.

[104] 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*, 2011.

[105] C. J. Maas and J. J. Hox, "Sufficient sample sizes for multilevel modeling." *Methodology: European Journal of Research Methods for the Behavioral and Social Sciences*, vol. 1, no. 3, p. 86, 2005.

[106] 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," *IEEE Micro*, vol. 23, no. 6, pp. 62–68, 2003.

[107] M. Maiterth, M. Schulz, D. Kranzlmüller, and B. Rountree, "Power balancing in an emulated exascale environment," in *Parallel and Distributed Processing Symposium Workshops, 2016 IEEE International*. IEEE, 2016, pp. 1142–1149.

[108] A. D. Malony, S. Biersdorff, S. Shende, H. Jagode, S. Tomov, G. Juckeland, R. Dietrich, D. Poole, and C. Lamb, "Parallel performance measurement of heterogeneous parallel systems with gpus," in *Parallel Processing (ICPP), 2011 International Conference on*. IEEE, 2011, pp. 176–185.

[109] A. Mandal, P. Ruth, I. Baldin, D. Król, G. Juve, R. Mayani, R. F. Da Silva, E. Deelman, J. Meredith, J. Vetter *et al.*, "Toward an end-to-end framework for modeling, monitoring and anomaly detection for scientific workflows," in *Parallel and Distributed Processing Symposium Workshops, 2016 IEEE International*. IEEE, 2016, pp. 1370–1379.

[110] M. M. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood, "Multifacet's general execution-driven multiprocessor simulator (gems) toolset," *ACM SIGARCH Computer Architecture News*, vol. 33, no. 4, pp. 92–99, 2005.

[111] I. Mavroidis, I. Papaefstathiou, L. Lavagno, D. S. Nikolopoulos, D. Koch, J. Goodacre, I. Sourdis, V. Papaefstathiou, M. Coppola, and M. Palomino, "Ecoscale: Reconfigurable

computing and runtime system for future exascale systems," in *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*. EDA Consortium, 2016, pp. 696–701.

[112] J. D. McCalpin, "Stream benchmark," *Link: www. cs. virginia. edu/stream/ref. html# what*, vol. 22, 1995.

[113] R. McKenna, S. Herbein, A. Moody, T. Gamblin, and M. Taufer, "Machine learning predictions of runtime and io traffic on high-end clusters," in *Cluster Computing (CLUSTER), 2016 IEEE International Conference on*. IEEE, 2016, pp. 255–258.

[114] J. Meng, X. Wu, V. Morozov, V. Vishwanath, K. Kumaran, and V. Taylor, "Skope: A framework for modeling and exploring workload behavior," in *Proceedings of the 11th ACM Conference on Computing Frontiers*. ACM, 2014, p. 6.

[115] A. Merkel and F. Bellosa, "Memory-aware scheduling for energy efficiency on multicore processors." *HotPower*, vol. 8, pp. 123–130, 2008.

[116] A. Merkel and F. Bellosa, "Memory-aware scheduling for energy efficiency on multicore processors." *HotPower*, vol. 8, pp. 123–130, 2008.

[117] P. Messina, "The exascale computing project," *Computing in Science & Engineering*, vol. 19, no. 3, pp. 63–67, 2017.

[118] M. M. Michael and A. K. Nanda, "Design and performance of directory caches for scalable shared memory multiprocessors," in *High-Performance Computer Architecture, 1999. Proceedings. Fifth International Symposium On*. IEEE, 1999, pp. 142–151.

[119] A. K. Mishra, S. Srikantaiah, M. Kandemir, and C. R. Das, "Cpm in cmps: Coordinated power management in chip-multiprocessors," in *High Performance Computing, Networking, Storage and Analysis (SC), 2010 International Conference for*. IEEE, 2010, pp. 1–12.

[120] S. Mittal and J. S. Vetter, "A survey of methods for analyzing and improving gpu energy efficiency," *ACM Computing Surveys (CSUR)*, vol. 47, no. 2, p. 19, 2015.

[121] J. Mohd-Yusof, S. Swaminarayan, and T. Germann, "Co-design for molecular dynamics: An exascale proxy application," 2013.

[122] J. Mohd Yusof, S. Swaminarayan, and T. Germann, "Co-design for molecular dynamics: An exascale proxy application," 2013.

[123] S. S. Mukherjee, S. K. Reinhardt, B. Falsafi, M. Litzkow, M. D. Hill, D. A. Wood, S. Huss-Lederman, and J. R. Larus, "Wisconsin wind tunnel ii: a fast, portable parallel architecture simulator," *IEEE Concurrency*, vol. 8, no. 4, pp. 12–20, 2000.

[124] 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*. IEEE, 2010, pp. 115–122.

[125] W. E. Nagel, A. Arnold, M. Weber, H.-C. Hoppe, and K. Solchenbach, "Vampir: Visualization and analysis of mpi resources," 1996.

[126] R. Nath, S. Tomov, and J. Dongarra, "An improved magma gemm for fermi graphics processing units," *The International Journal of High Performance Computing Applications*, vol. 24, no. 4, pp. 511–515, 2010.

[127] J. L. Nazareth, "Conjugate gradient method," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 1, no. 3, pp. 348–353, 2009.

[128] M. Noeth, P. Ratn, F. Mueller, M. Schulz, and B. R. de Supinski, "Scalatrace: Scalable compression and replay of communication traces for high-performance computing," *Journal of Parallel and Distributed Computing*, vol. 69, no. 8, pp. 696–710, 2009.

[129] K. J. Nowka, G. D. Carpenter, E. W. MacDonald, H. C. Ngo, B. C. Brock, K. I. Ishii, T. Y. Nguyen, and J. L. Burns, "A 32-bit powerpc system-on-a-chip with support for dynamic voltage scaling and dynamic frequency scaling," *IEEE Journal of Solid-State Circuits*, vol. 37, no. 11, pp. 1441–1447, 2002.

[130] T. Patki, D. K. Lowenthal, B. Rountree, M. Schulz, and B. R. De Supinski, "Exploring hardware overprovisioning in power-constrained, high performance computing," in *Proceedings of the 27th international ACM conference on International conference on supercomputing*. ACM, 2013, pp. 173–182.

[131] S. Plimpton, R. Pollock, and M. Stevens, "Particle-mesh ewald and rrespa for parallel molecular dynamics simulations." in *PPSC*. Citeseer, 1997.

[132] A. F. Rodrigues, K. S. Hemmert, B. W. Barrett, C. Kersey, R. Oldfield, M. Weston, R. Risen, J. Cook, P. Rosenfeld, E. CooperBalls *et al.*, "The structural simulation toolkit," *ACM SIGMETRICS Performance Evaluation Review*, vol. 38, no. 4, pp. 37–42, 2011.

[133] G. Rodriguez, R. M. Badia, and J. Labarta, "Generation of simple analytical models for message passing applications," in *European Conference on Parallel Processing*. Springer, 2004, pp. 183–188.

[134] E. Rotem, A. Naveh, M. Moffie, and A. Mendelson, "Analysis of thermal monitor features of the intel pentium m processor," in *TACS Workshop at ISCA-31*, 2004.

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

[136] B. Rountree, D. H. Ahn, B. R. De Supinski, D. K. Lowenthal, and M. Schulz, "Beyond dvfs: A first look at performance under a hardware-enforced power bound," in *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International*. IEEE, 2012, pp. 947–953.

[137] 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 *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*. ACM, 2007, p. 49.

[138] 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*. IEEE, 2011, pp. 1–8.

[139] B. Rountree, D. K. Lownenthal, B. R. De Supinski, M. Schulz, V. W. Freeh, and T. Bletsch, "Adagio: making dvs practical for complex hpc applications," in *Proceedings of the 23rd international conference on Supercomputing*. ACM, 2009, pp. 460–469.

[140] K. Rupp, "42 Years of Microprocessor Trend Data," https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/, 2018.

[141] S. Ryoo, C. I. Rodrigues, S. S. Baghsorkhi, S. S. Stone, D. B. Kirk, and W.-m. W. Hwu, "Optimization principles and application performance evaluation of a multithreaded gpu using cuda," in *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming*. ACM, 2008, pp. 73–82.

[142] S. Shao, A. K. Jones, and R. Melhem, "A compiler-based communication analysis approach for multiprocessor systems," in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*. IEEE, 2006, pp. 10–pp.

[143] S. S. Shende and A. D. Malony, "The tau parallel performance system," *The International Journal of High Performance Computing Applications*, vol. 20, no. 2, pp. 287–311, 2006.

[144] S. Song, 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*. IEEE, 2011, pp. 262–271.

[145] S. Song, C. Su, B. Rountree, and K. W. Cameron, "A simplified and accurate model of power-performance efficiency on emergent gpu architectures," in *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*. IEEE, 2013, pp. 673–686.

[146] K. Spafford and J. S. Vetter, "Automated design space exploration with aspen," *Scientific Programming*, vol. 2015, p. 7, 2015.

[147] K. L. Spafford and J. S. Vetter, "Aspen: a domain specific language for performance modeling," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society Press, 2012, p. 84.

[148] V. Spiliopoulos, S. Kaxiras, and G. Keramidas, "Green governors: A framework for continuously adaptive dvfs," in *Green Computing Conference and Workshops (IGCC), 2011 International*. IEEE, 2011, pp. 1–8.

[149] D. Sundaram-Stukel and M. K. Vernon, "Predictive analysis of a wavefront application using loggp," *ACM SIGPLAN Notices*, vol. 34, no. 8, pp. 141–150, 1999.

[150] N. R. Tallent and A. Hoisie, "Palm: easing the burden of analytical performance modeling," in *Proceedings of the 28th ACM international conference on Supercomputing*. ACM, 2014, pp. 221–230.

[151] L. Tan, L. Chen, Z. Chen, Z. Zong, R. Ge, and D. Li, "Hp-daemon: High performance distributed adaptive energy-efficient matrix-multiplication," *Procedia Computer Science*, vol. 29, pp. 599–613, 2014.

[152] C. Thompson, "CPUSpeed," http://carlthompson.net/Software/CPUSpeed, 2008.

[153] V. Tiwari, D. Singh, S. Rajgopal, G. Mehta, R. Patel, and F. Baez, "Reducing power in high-performance microprocessors," in *Proceedings of the 35th annual Design Automation Conference*. ACM, 1998, pp. 732–737.

[154] J. Treibig, G. Hager, and G. Wellein, "Likwid: A lightweight performance-oriented tool suite for x86 multicore environments," in *Parallel Processing Workshops (ICPPW), 2010 39th International Conference on*. IEEE, 2010, pp. 207–216.

[155] Y. Ukidave, A. K. Ziabari, P. Mistry, G. Schirner, and D. Kaeli, "Analyzing power efficiency of optimization techniques and algorithm design methods for applications on heterogeneous platforms," *The International Journal of High Performance Computing Applications*, vol. 28, no. 3, pp. 319–334, 2014.

[156] M. Umar, J. S. Meredith, J. S. Vetter, and K. W. Cameron, "A study of power-performance modeling using a domain-specific language," in *Computer Architecture and High Performance Computing (SBAC-PAD), 2016 28th International Symposium on*. IEEE, 2016, pp. 84–92.

[157] D. Unat, C. Chan, W. Zhang, S. Williams, J. Bachan, J. Bell, and J. Shalf, "Exasat: An exascale co-design tool for performance modeling," *The International Journal of High Performance Computing Applications*, vol. 29, no. 2, pp. 209–232, 2015.

[158] D. Unat, T. Nguyen, W. Zhang, M. N. Farooqi, B. Bastem, G. Michelogiannakis, A. Almgren, and J. Shalf, "Tida: High-level programming abstractions for data locality management," in *International Conference on High Performance Computing*. Springer, 2016, pp. 116–135.

[159] L. G. Valiant, "A bridging model for parallel computation," *Communications of the ACM*, vol. 33, no. 8, pp. 103–111, 1990.

[160] A. Varga and R. Hornig, "An overview of the omnet++ simulation environment," in *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008, p. 60.

[161] S. Venkataraman, Z. Yang, M. J. Franklin, B. Recht, and I. Stoica, "Ernest: Efficient performance prediction for large-scale advanced analytics." in *NSDI*, 2016, pp. 363–378.

[162] J. S. Vetter and J. S. Meredith, "Synthetic program analysis with aspen," in *Proceedings of the 3rd International Conference on Exascale Applications and Software*. University of Edinburgh, 2015, pp. 1–6.

[163] J. S. Vetter and P. H. Worley, "Asserting performance expectations," in *Proceedings of the 2002 ACM/IEEE conference on Supercomputing*. IEEE Computer Society Press, 2002, pp. 1–13.

[164] J. S. Vetter and A. Yoo, "An empirical performance evaluation of scalable scientific applications," in *Supercomputing, ACM/IEEE 2002 Conference*. IEEE, 2002, pp. 16–16.

[165] T. Vijayaraghavany, Y. Eckert, G. H. Loh, M. J. Schulte, M. Ignatowski, B. M. Beckmann, W. C. Brantley, J. L. Greathouse, W. Huang, A. Karunanithi *et al.*, "Design and analysis of an apu for exascale computing," in *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on*. IEEE, 2017, pp. 85–96.

[166] N. Vijaykrishnan, M. Kandemir, M. J. Irwin, H. S. Kim, and W. Ye, "Energy-driven integrated hardware-software optimizations using simplepower," *ACM SIGARCH Computer Architecture News*, vol. 28, no. 2, pp. 95–106, 2000.

[167] O. Villa, D. R. Johnson, M. Oconnor, E. Bolotin, D. Nellans, J. Luitjens, N. Sakharnykh, P. Wang, P. Micikevicius, A. Scudiero *et al.*, "Scaling the power wall: a path to exascale," in *High Performance Computing, Networking, Storage and Analysis, SC14: International Conference for*. IEEE, 2014, pp. 830–841.

[168] R. W. Vuduc and J. W. Demmel, *Automatic performance tuning of sparse matrix kernels*. University of California, Berkeley, 2003, vol. 1.

[169] H.-S. Wang, X. Zhu, L.-S. Peh, and S. Malik, "Orion: a power-performance simulator for interconnection networks," in *Microarchitecture, 2002.(MICRO-35). Proceedings. 35th Annual IEEE/ACM International Symposium on*. IEEE, 2002, pp. 294–305.

[170] M. Ware, K. Rajamani, M. Floyd, B. Brock, J. C. Rubio, F. Rawson, and J. B. Carter, "Architecting for power management: The ibm® power7âĎć approach," in *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*. IEEE, 2010, pp. 1–11.

[171] V. M. Weaver, M. Johnson, K. Kasichayanula, J. Ralph, P. Luszczek, D. Terpstra, and S. Moore, "Measuring energy and power with papi," in *Parallel Processing Workshops (ICPPW), 2012 41st International Conference on*. IEEE, 2012, pp. 262–268.

[172] S. Wienke, J. Miller, M. Schulz, and M. S. Müller, "Development effort estimation in hpc," in *High Performance Computing, Networking, Storage and Analysis, SC16: International Conference for*. IEEE, 2016, pp. 107–118.

[173] S. Williams, A. Waterman, and D. Patterson, "Roofline: an insightful visual performance model for multicore architectures," *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, 2009.

[174] F. Wolf, C. Bischof, T. Hoefler, B. Mohr, G. Wittum, A. Calotoiu, C. Iwainsky, A. Strube, and A. Vogel, "Catwalk: a quick development path for performance models," in *European Conference on Parallel Processing*. Springer, 2014, pp. 589–600.

[175] F. Wolf, B. J. Wylie, E. Abrahám, D. Becker, W. Frings, K. Fürlinger, M. Geimer, M.-A. Hermanns, B. Mohr, S. Moore *et al.*, "Usage of the scalasca toolset for scalable performance analysis of large-scale parallel applications," in *Tools for High Performance Computing*. Springer, 2008, pp. 157–167.

[176] Q. Wu, P. Juang, M. Martonosi, L.-S. Peh, and D. W. Clark, "Formal control techniques for power-performance management," *IEEE micro*, vol. 25, no. 5, pp. 52–62, 2005.

[177] F. Xie, M. Martonosi, and S. Malik, "Intraprogram dynamic voltage scaling: Bounding opportunities with analytic modeling," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 1, no. 3, pp. 323–367, 2004.

[178] R. Xu, S. Chandrasekaran, and B. Chapman, "Exploring programming multi-gpus using openmp and openacc-based hybrid model," in *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International*. IEEE, 2013, pp. 1169–1176.

[179] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, "The design and use of simplepower: a cycle-accurate energy estimation tool," in *Proceedings of the 37th Annual Design Automation Conference*. ACM, 2000, pp. 340–345.

[180] L. Yu, D. Li, S. Mittal, and J. S. Vetter, "Quantitatively modeling application resilience with the data vulnerability factor," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Press, 2014, pp. 695–706.

[181] Y. Zhang, L. Peng, B. Li, J.-K. Peir, and J. Chen, "Architecture comparisons between nvidia and ati gpus: Computation parallelism and data communications," in *Workload Characterization (IISWC), 2011 IEEE International Symposium on*. IEEE, 2011, pp. 205–215.

[182] G. Zheng, G. Kakulapati, and L. V. Kalé, "Bigsim: A parallel simulator for performance prediction of extremely large parallel machines," in *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*. IEEE, 2004, p. 78.