UiT
THE ARCTIC
UNIVERSITY
OF NORWAY

Department of Computer Science

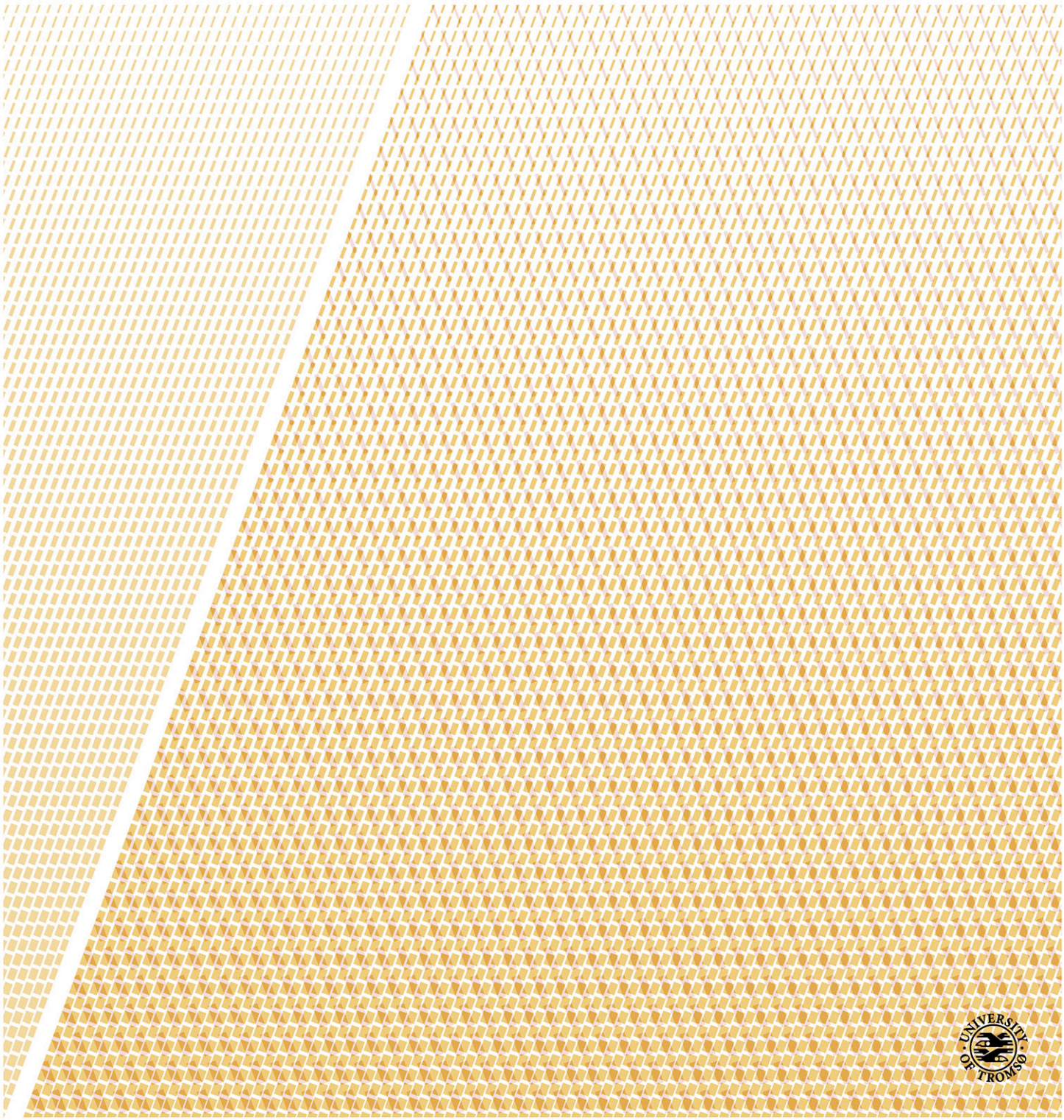
Faculty of Science and Technology

Modeling Energy Consumption of Computing Systems: from Homogeneous to Heterogeneous Systems

—

Vi Ngoc-Nha Tran

A dissertation for the degree of Philosophiae Doctor – September 2018



MODELING ENERGY CONSUMPTION OF COMPUTING SYSTEMS: FROM HOMOGENEOUS TO HETEROGENEOUS SYSTEMS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE FACULTY OF SCIENCE AND TECHNOLOGY
OF UiT THE ARCTIC UNIVERSITY OF NORWAY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Vi Ngoc-Nha Tran

Abstract

Nowadays, reducing energy consumption and improving energy efficiency of computing systems become ones of the main research topics in computer science. In order to improve energy efficiency, it is important to understand how computing systems consume energy and to characterize their energy consumption when running applications. Power and energy models are the essential tools to provide the prediction of the power and energy consumption of computing systems and insight into how they consume power and energy.

Devising models which can provide an accurate prediction of energy consumption requires the detailed understanding of the underlying platform and the communication and computation patterns of the considered application. Therefore, it is challenging to build accurate power and energy models that can be used for general devices and general applications.

This thesis addresses the above challenge by developing three approaches of devising power and energy models, varying from homogeneous systems including one type of devices (e.g., CPU, GPU, Ultra Low Power embedded system) to heterogeneous systems including several types of devices with different architectures.

- The thesis developed new fine-grained power models supporting architecture-application co-design by considering both platform and application properties. The models were trained and validated with data from a set of micro-benchmarks and application kernels on Movidius Myriad, an ultra-low power embedded system. The model predicted power consumption within 12% deviation from the real power consumption. We also proposed and validated a framework predicting when to apply race-to-halt (RTH) strategy to a given application.

- The thesis devised ICE, new energy complexity models for parallel (multi-threaded) algorithms that were validated on real multicore platforms and applicable to a wide range of parallel algorithms. We presented two case studies using the complexity models to characterize and compare the energy consumption of sparse matrix-vector multiplication and matrix multiplication kernels according to the three aspects: different algorithms, different input matrix types and different platforms. The experimental results regarding which algorithm consumes more energy with different inputs on different platforms confirmed the prediction by the new

models. The study also provided the platform parameters of the ICE models for eleven platforms including HPC, accelerator and embedded platforms to improve the model usability and accuracy.

- The thesis proposed REOH, the holistic tuning approach to choose the most energy-efficient configurations for heterogeneous systems including several types of devices with different architectures (e.g., CPUs, GPUs). REOH uses probabilistic network to predict the most energy-efficient configuration (i.e., which platform and its setting) of a heterogeneous system for running a given application. Based on the REOH approach, we developed an open-source energy-optimizing runtime framework for selecting an energy efficient configuration of a heterogeneous system for a given application at runtime.

Acknowledgments

This thesis work is only possible with the support of several people to whom I am deeply grateful. First and foremost, I would like to express my sincere gratitude to my advisor Phuong H. Ha for his guiding, support and encouragement during my PhD study.

I am also thankful to my second advisor Otto Anshus and Alexander Horsch for the helpful comments and discussions on the work presented in this thesis.

I would like to thank my thesis committee members: Per Stenström, Magnus Jahre and Randi Karlsen for the invaluable comments and suggestions to improve my thesis work.

I am grateful to the staffs of the Department of Computer Science including Svein Tore Jensen, Jan Fuglesteg, Maria Hauglann, Kai-Even Nilssen, Ken-Arne Jensen and Jon Ivar Kristiansen for their administrative and technical supports. A great help has been done to facilitate my research work.

I am also grateful to the AGC members: Ibrahim Umar, Saeed Shariati, Pradeep Kumar and Tommy Oines for all the discussions and the good time in our lab.

I would also like to thank everyone who worked with me in the EU EXCESS project, especially Christoph Kessler and Brendan Barry from whom I received a huge amount of constructive input on the deliverable works.

I own a deep gratitude to my family: my parents, my sister, my husband, and son for their unconditional love and support. They are the motivation for me to become a better person every day.

List of Papers and Reports

This thesis is based on the work described in the following publications and reports.

Chapter 3 revises the publications:

- [77]: Vi N.N. Tran, Brendan Barry, and Phuong Ha. Power models supporting energy-efficient co-design on ultra-low power embedded systems. In 2016 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS), pages 39-46, 2016.
- [80]: Vi N.N. Tran, B. Barry and Phuong Ha. RTHpower: Accurate fine-grained power models for predicting race-to-halt effect on ultra-low power embedded systems. In 2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pages 155-156, 2016.

Chapter 4 revises the publication:

- [78]: Vi N.N. Tran and Phuong Ha. Ice: A general and validated energy complexity model for multi- threaded algorithms. In 2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS), pages 1041-1048, 2016.

Chapter 5 revises the publication:

- [79] Vi N.N. Tran, Tommy Oines, Alexander Horsch and Phuong Ha. REOH: Using Probabilistic Network for Runtime Energy Optimization of Heterogeneous Systems. In 2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS), pages to appear.

Workshops and Technical Reports:

- Vi N.N. Tran, Phuong Ha. ICE: A General and Validated Energy Complexity Model for Multithreaded Algorithms. The 9th Nordic Workshop on Multi-core Computing (MCC '16)
- Yosandra Sandoval, Dennis Hoppe, Dmitry Khabi, Micheal Gienger, ChristophKessler, Lu Li, Usman Dastgeer, Vi N.N Tran, Ibrahim Umar, Phuong Ha, Philippas Tsigas, Anders Gidenstam, Ivan Walulya, Paul Renaud-Goud. EXCESS: Execution Models for Energy-Efficient

Computing Systems. fEEDBACk Workshop Energy Efficient Distributed and Parallel Computing (fEEDBACk '15)

- [35]: Phuong Ha, Vi N.N. Tran, Ibrahim Umar, Philippas Tsigas, Anders Gidenstam, Paul Renaud-Goud, Ivan Walulya, and Aras Atalar. D2.1 Models for energy consumption of data structures and algorithms. Technical Report FP7-611183 D2.1, EU FP7 Project EXCESS, August 2014.

- [32]: Phuong Ha, Vi N.N. Tran, Ibrahim Umar, Aras Atalar, Anders Gidenstam, Paul Renaud-Goud, and Philippas Tsigas. D2.2 White-box methodologies, programming abstractions and libraries. Technical Report FP7-611183 D2.2, EU FP7 Project EXCESS, February 2015.

- [34]: Phuong Ha, Vi N.N. Tran, Ibrahim Umar, Aras Atalar, Anders Gidenstam, Paul Renaud-Goud, Philippas Tsigas, and Ivan Walulya. D2.3 power models, energy models and libraries for energy- efficient concurrent data structures and algorithms. Technical Report FP7-611183 D2.3, EU FP7 Project EXCESS, February 2016.

- [33]: Phuong Ha, Vi N.N. Tran, Ibrahim Umar, Aras Atalar, Anders Gidenstam, Paul Renaud-Goud, Philippas Tsigas, and Ivan Walulya. D2.3 d2.4 report on the final prototype of programming abstractions for energy-efficient inter-process communication. Technical Report FP7-611183 D2.4, EU FP7 Project EXCESS, August 2016.

- [44]: Christoph Kessler, Lu Li, Usman Dastgeer, Philippas Tsigas, Anders Gidenstam, Paul Renaud- Goud, Ivan Walulya, Aras Atalar, David Moloney, Phuong Ha Hoai, and Vi N.N. Tran. D1.1 Early validation of system-wide energy compositionality and affecting factors on the EXCESS plat- forms. Technical Report FP7-611183 D1.1, EU FP7 Project EXCESS, April 2014.

- [43]: Christoph Kessler, Lu Li, Usman Dastgeer, Rosandra Cuello, Oskar Sjostrom, Phuong Ha Hoai, and Vi N.N. Tran. D1.3 Energy-tuneable domain-specific language/library for linear system solving. Technical Report FP7-611183 D1.3, EU FP7 Project EXCESS, February 2015.

- [45]: Dmitry Khabi, Vi N.N. Tran, and Ivan Walulya. D5.4 Report on the first evaluation results and discussion. Technical Report FP7-611183 D5.4, EU FP7 Project EXCESS, August 2015.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Along with performance optimization, energy efficiency is one of the main concerns of computing systems. Reducing energy consumption of computing systems, varying from homogeneous systems such as embedded systems, CPUs, GPUs to heterogeneous systems including different devices with different architectures becomes one of the top challenges in computer science.

Significant efforts have been focused on architectural energy-saving techniques. To further reduce the energy consumption of future computing systems, the co-design of software and hardware considering both applications and systems is essential to exploit both software and hardware energy-saving techniques [42].

One of the key research directions to improve energy efficiency is to understand how much energy a computing system consumes and characterize their energy consumption. By characterizing the energy consumption of computing systems, researchers and practitioners can design and implement new approaches to reduce the energy consumed by a certain algorithm on a specific platform.

The energy and power consumption of computing systems can be either measured by integrated sensors or external multi-meters or estimated by models. Energy and power measurement equipment and sensors are not always available and can be costly to deploy and set up. Therefore, energy and power models are the alternative and convenient methods to estimate the energy consumption of an application on a computing system [67]. Devising power and energy models is also crucial to gain insights into how a computer system consumes power and energy.

1.1 Research Questions

1.1.1 Research Question 1

Significant efforts have been devoted to devising power and energy models of computing systems, resulting in several seminal papers in the literature, such as [41, 53, 55, 10, 19, 18, 46, 47, 39, 63, 73]

modeling power of architectures or applications.

Jacobson et al. [41] proposed accurate power modeling methodologies for POWER-family processors while GPUWattch and McPAT are robust power models for GPUs and CPUs. Alonso et al. [10] proposed energy models for three key dense-matrix factorizations. Roofline model of energy [19, 18] considers both algorithmic and platform properties. However, the Roofline model does not consider the number of cores running applications as a model parameter (i.e., coarse-grained models). Theoretical models by Korthikanti et al. [47, 46] were based on strong theoretical assumptions and are not yet validated on real platforms. Koala model [73] requires the system supported dynamic voltage and frequency scaling (DVFS) and short frequency switching delay in order to gain energy saving from its methodology. However, only two x86-based platforms among 10 validated platforms gained energy saving results which are presented in the paper. Imes et al. [39] provided a portable approach to make real-time decision and run the chosen configuration to minimize energy consumption. However, the approach requires systems supporting hardware resource (e.g., model-specific register) to expose energy data to the software during run-time. Mishra et al. [63] used a probabilistic modeling approach to find the most energy-efficient configuration by combining online and offline machine-learning approaches. This approach requires a significant amount of data collected to feed to its probabilistic network.

Recently, novel and specific-purpose systems such as ultra-low power (ULP) embedded systems have become popular in the scientific community and industry, especially in media and wearable computing. ULP embedded systems have different architectures from the general-purpose architectures (e.g., CPU and GPU). As a result, the approach to model the power of ULP systems needs to be customized for their architecture. ULP systems can achieve low energy per instruction down to a few pJ [9]. Alioto [9] mentioned that techniques such as pipe-lining, hardware replication, ultra-low-voltage memory design, and leakage-reducing make a system ultra-low power. In order to model ULP systems where energy per instruction can be as low as few pJ, more accurate fine-grained approaches are needed. For instance, the dynamic power P^{dyn} of operations in Table 3.2, which is as low as 13 mW, cannot be measured by using the prior coarse-grained approaches [19, 18].

For embedded systems which has real-time constraint and limited energy supply, two of the most popular strategies to reduce the energy consumption are Dynamic Voltage and Frequency Scaling (DVFS) [51] and race-to-halt (RTH) (i.e, systems run at higher frequency to finish as soon as possible, and then put certain hardware parts to sleep to save energy) [13]. These two techniques are explained in Chapter 2. For new embedded systems which do not support DVFS features such as Movidius Myriad [40], RTH is one of the remaining choices for saving energy. RTH theory is used to let the CPU work at the highest performance levels then go back to a low energy-draw state. The process is repeated multiple times during program execution. In fact, Myriad supports a power management feature to power on/off individual cores. However, to the best of our knowledge, there is no fine-grained power model that supports investigating the trade-off between performance and

energy consumption on ULP embedded systems and whether the RTH strategy that is widely used in high-performance computing (HPC) systems is still applicable to ULP embedded systems.

The first part of this thesis work investigates the modeling methodology to answer the research question: *"RQ1: How to accurately model and estimate the power and energy consumptions and support energy-efficient co-design of ultra-low power embedded systems?"*

1.1.2 Research Question 2

The models which are able to estimate absolute values of power and energy consumption from RQ1 however, requires a significant detailed understanding of the targeted platform and its components to develop a set of micro-benchmarks. For other domains such as algorithm design, the absolute values of energy consumption estimation are not required. Instead, an analysis tool to provide an understanding of how an algorithm consumes energy as the input grows is more essential. In the next work of this thesis, we aim to provide the understanding of how an algorithm consumes energy via energy complexity models.

Understanding the energy complexity of algorithms is crucially important to improve the energy efficiency of algorithms [82, 81, 83, 49] and reduce the energy consumption of computing systems [80, 77, 50].

However, there are no analytic models for multithreaded algorithms that are both applicable to a wide range of algorithms and comprehensively validated yet (cf. Table 1.1). The existing *parallel* energy models are either theoretical studies without validation or only applicable for specific algorithms. Modeling energy consumption of *parallel* algorithms is difficult since the energy models must take into account the complexity of both parallel algorithms and parallel platforms. The algorithm complexity results from parallel computation, concurrent memory accesses and inter-process communication. The platform complexity results from multicore architectures with a deep memory hierarchy.

The existing models and their classification are summarized in Table 1.1 by three aspects: i) ability to analyze the energy complexity of parallel algorithms (i.e. Energy complexity analysis for parallel algorithms), ii) applicability to a wide range of algorithms (i.e., Algorithm generality), and iii) model validation (i.e., Validation). To the best of our knowledge, the energy model that covers all three aspects: Energy complexity analysis for parallel algorithms, Algorithm generality and Validation is missing.

The second study of this thesis answers the energy complexity question: *"RQ2: Given two parallel algorithms A and B for a given problem, how to identify which algorithm consumes less energy analytically?"*

Table 1.1: Energy Model Summary

Study	**Energy complexity analysis for parallel algorithms**	**Algorithm generality**	**Validation**
LEO [63]	No	General	Yes
POET [39]	No	General	Yes
Koala [73]	No	General	Yes
Roofline [19, 18]	No	General	Yes
Energy scalability [46, 47]	Yes	General	No
Sequential energy complexity [70]	No	General	Yes
Alonso et al. [10]	Yes	Algorithm-specific	Yes
Malossi et al. [62]	Yes	Algorithm-specific	Yes

To the best of our knowledge, the ICE model is the first *validated model that supports* energy complexity analysis for *general multithreaded algorithms.*

1.1.3 Research Question 3

So far, both the research questions RQ1 and RQ2 addresses the energy modeling questions for accurate models and complexity models conducted on homogeneous systems including one type of devices (e.g., embedded systems, CPU or GPU). Modeling the energy consumption of applications running on heterogeneous systems including different types of devices are more complex and challenging. In the next modeling approach, we want to estimate the energy consumption of an application running on heterogeneous systems and identify the system configurations to run the application to achieve the most energy efficiency.

The factors that have impacts on the application performance, energy-efficiency and its optimization strategies are algorithm design, implementation (i.e., control flow, memory types, memory access pattern, and instruction count), and its execution configuration [24]. When an application runs on a heterogeneous system, one of the strategies to reduce energy consumption is to run the application with an appropriate system configuration.

Several attempts [60, 92, 38, 63, 17, 6, 65, 61, 29, 58, 85] have been made to find the best configurations to run an application to achieve energy efficiency. However, available tuning approaches are mostly conducted for homogeneous systems while little research considers heterogeneous systems including several platform components (e.g., CPUs and GPUs) with different types of processing units and different architectures.

Table 1.2 summarizes the studies to optimize energy efficiency by choosing an appropriate configuration of computing systems for a given application. Table 1.2 lists the related works according to the four aspects: the optimization goal (i.e, Optimization), whether the optimization object is

configuration or code variant (i.e., Object), whether the targeted system is homogeneous or heterogeneous (i.e., System), and whether the approach is applicable to general or specific applications (i.e., Application). The details of the related work are described in Section 5.5.

The main goal of existing tuning approaches is to improve energy-efficiency. However, the existing models are mostly built for homogeneous systems, which has only one type of devices such as GPU [17, 6, 65, 29, 61, 85] or CPU [38, 92, 63]. There are also a set of studies [72, 91, 90] for heterogeneous systems (i.e., APUs) but they mainly focus on improving performance instead of energy-efficiency.

The existing heterogeneous approaches in the Table 1.2 are either for specific applications (i.e., iterative applications that can be divided to several iterations where execution time of the next iteration can be predicted based on the current iteration) [58, 59] or for finding a heterogeneous balance of datacenter [30] where the configuration at datacenter level is a mix of CPUs or microprocessors.

Among the available tuning approaches, probabilistic model-based approaches have their advantages of not requiring prior knowledge on the targeted application or the throughout understanding of system components like other approaches [65, 29]. By finding the similarity between a targeted application from sampling data and previously observed applications from training data, it can quickly provide the accurate estimation of energy consumption for the targeted application.

The previous probabilistic model-based approaches are only applicable to homogeneous systems (i.e., CPUs). Heterogeneous systems have complex structures containing different platform architectures (e.g., CPUs, GPUs, FPGAs, ASICs) where each platform has its own sets of settings and methods to change its configurations. Applying the probabilistic model-based approach [63] on each individual platform of a heterogeneous system requires the analysis of the available settings and a new configuration data for each platform. In the other words, it requires separated sets of training and sampling data, and separated runs of prediction for each platform. This results in more sampling runs than doing one prediction for a heterogeneous system with only one whole set of training and sampling data. Therefore, the probabilistic model based approaches for heterogeneous systems requires the analysis of the available settings of all included platforms within a heterogeneous system and finding the setting equivalence of one platform to another platform. The third part of this thesis aims to address the research question: *"RQ3: How to identify the most energy-efficient system configurations (i.e., platform and its setting) of a heterogeneous system containing platforms with different architectures to run the application?"*

1.2 Research Contributions

This thesis tackles the above three research questions by investigating and developing the three modeling approaches:

- Accurate Power Models Supporting Energy Efficient Co-design for Ultra-low Power Embedded Systems

Table 1.2: Auto-Tuning Framework

Study	**Optimization**	**Object**	**System**	**Application**
OSKI [84]	Time	Code variant	Homogeneous (i.e., CPU)	Specific (i.e., Sparse kernels)
Nitro [64]	Time	Code variant	Homogeneous	General
PowerCap [92]	Timeliness Energy-efficiency	Configuration	Homogeneous (i.e., CPU)	General
POET [38]	Energy-efficiency	Configuration	Homogeneous (i.e., CPU)	General
LEO [63]	Time Energy-efficiency	Configuration	Homogeneous (i.e., CPU)	General
HPC runtime framework [17]	Energy-efficiency	Configuration	Homogeneous (i.e., CPU)	General
GPU models [6]	Power	Configuration	Homogeneous (i.e., GPU)	General
CRISP [65]	Energy	Configuration	Homogeneous (i.e., GPGPU)	General
MPC [61]	Energy-efficiency	Configuration	Homogeneous (e.g., GPGPU)	General
GreenGPU [58, 59]	Energy-efficiency	Workload division Frequency	Heterogeneous (e.g., CPU and GPU)	Specific (i.e., Iterative applications)
GPGPU DVFS models [29]	Energy-efficiency	Configuration	Homogeneous (i.e., GPGPU)	General
GPGPU SVR models [85]	Energy-efficiency	Configuration	Homogeneous (i.e., GPGPU)	General
Market mechanism [30]	Service quality Energy-efficiency efficiency	High-level configurations (i.e., Datacenters)	Heterogeneous (e.g., CPUs and microprocessors) (e.g., CPU and GPU)	General

- Energy Complexity Models for Multithreaded Algorithms
- Runtime Energy Optimization for Heterogeneous Systems

In the remaining of this section, the brief descriptions of solutions and results to each of the three modeling approaches are described. The full details of the three modeling approaches can be found in Chapters 3, 4 and 5, respectively.

1.2.1 RQ1: Accurate Power Models Supporting Energy Efficient Co-design for Ultra-low Power Embedded Systems

In order to estimate the absolute power consumption of an application on ULP embedded system and investigate RTH strategy, we propose new RTHpower models which support architecture-application co-design by considering both platform and application properties. The RTHpower models are application-general since they characterize applications by their arithmetic intensity [87] which can be extracted from any application. The RTHpower models are also practical since they are built and validated on Movidius platform using application kernels. The main contributions of this modeling approach are three-fold as follows:

- We propose new application-general fine-grained power models (namely, RTHpower) that provide insights into how a given application consumes power and give hints to investigate the trade-offs between performance and power consumption on ULP embedded systems. The RTHpower models support co-design on ULP systems by considering three parameter groups: platform properties, application properties (e.g., arithmetic intensity and scalability) and execution settings (e.g., the number of cores executing a given application) (cf. Section 3.2).
- We validate the new RTHpower models on an ultra-low power embedded system, namely Movidius Myriad. The models are trained and validated with power data from different sets of micro-benchmarks, two computation kernels from Berkeley dwarfs [12] and one data-intensive kernel from Graph500 benchmarks [74]. The three chosen application kernels are dense matrix multiplication (Matmul), sparse matrix vector multiplication (SpMV) and breadth first search (BFS). The model validation has percentage error at most 8.5% for micro-benchmarks and 12% for application kernels (cf. Section 3.3).
- We investigate the RTH strategy on an ultra-low power embedded platform using the new RTHpower models. We propose a framework that is able to predict when to and when not to apply the RTH strategy in order to minimize energy consumption. We validate the framework using micro-benchmarks and application kernels. From our experiments, we show real scenarios when to use RTH and when not to use RTH. We can save up to 61% energy for dense matrix multiplication, 59% energy for SpMV by using RTH and up to 5% energy for BFS by not using RTH (cf. Section 3.4).

1.2.2 RQ2: Energy Complexity Models for Multithreaded Algorithms

The energy complexity model ICE proposed in this modeling approach is for general multithreaded algorithms and validated on three aspects: different algorithms for a given problem, different input types and different platforms. The proposed model is an analytic model which characterizes both algorithms (e.g., representing algorithms by their *work*, *span* and *I/O* complexity) and platforms (e.g., representing platforms by their static and dynamic energy of memory accesses and computational operations). By considering *work*, *span*, and *I/O* complexity, the new ICE model is applicable to any multithreaded algorithms.

Since the new ICE energy model focuses on analyzing the energy complexity of algorithms, the model does not give the estimation of absolute energy consumption. The new model, instead, provides the algorithm designers with the understanding of how an algorithm consumes energy and give insight into how to choose one algorithm over the others for different input types and platforms. The new ICE model is designed for analyzing the energy *complexity* of algorithms and therefore the model does not provide the estimation of absolute energy consumption. Hence, the details of underlying systems (e.g., runtime and architectures) are abstracted away to keep the ICE model simple and suitable for complexity analysis. O-notation represents an *asymptotic upper-bound* on energy complexity.

In this work, the following contributions have been made.

- Devising a new general energy model ICE for analyzing the energy complexity of a wide range of multithreaded algorithms based on their *work*, *span* and *I/O* complexity (cf. Section 4.2). The new ICE model abstracts away possible *multicore platforms* by their static and dynamic energy of computational operations and memory access. The new ICE model complements previous energy models such as energy roofline models [19, 18] that abstract away possible *algorithms* to analyze the energy consumption of different multicore platforms.

- Conducting two case studies (i.e., SpMV and matmul) to demonstrate how to apply the ICE model to find energy complexity of parallel algorithms. The selected parallel algorithms for SpMV are three algorithms: Compressed Sparse Column(CSC), Compressed Sparse Block(CSB) and Compressed Sparse Row(CSR)(cf. Section 4.3). The selected parallel algorithms for matmul are two algorithms: a basic matmul algorithm and a cache-oblivious algorithm (cf. Section 4.4).

- Validating the ICE energy complexity model with both data-intensive (i.e., SpMV) and computation-intensive (i.e., matmul) algorithms according to three aspects: different algorithms, different input types and different platforms. The results show the precise prediction on which validated SpMV algorithm (i.e., CSB or CSC) consumes more energy when using different matrix input types from Florida matrix collection [23] (cf. Section 4.5.6). The results also show the precise prediction on which validated matmul algorithm (i.e., basic or cache-oblivious) consumes more

energy (cf. Section 4.5.7). The model platform-related parameters for 11 platforms, including x86, ARM and GPU, are provided to facilitate the deployment of the ICE model. Moreover, the ICE models can also be applied to theoretical exascale systems and enable their energy complexity analysis.

1.2.3 RQ3: Using Probabilistic Network for Runtime Energy Optimization of Heterogeneous Systems

This study proposes holistic tuning approach based on probabilistic network to predict the most energy-efficient configuration of heterogeneous systems for a given application. Based on the application communication and computation patterns (i.e., Berkeley dwarfs [12], we choose the Rodinia benchmarks [4] for the experiments and devise a training data set. The objectives when choosing the benchmarks are to devise a training data set that covers a wide range of application patterns and characteristics.

In this modeling approach, we propose a way to unify the configurations of different platforms on a heterogeneous system in order to perform the prediction only once as compared to the previous approach for homogeneous systems. This way we save energy of the sampling runs. Even though we evaluate our probabilistic model-based approach (i.e., REOH) on a system containing CPU and GPU only, REOH is general for heterogeneous systems which contain any architectures (e.g., CPUs, GPUs, FPGAs, ASICS) where we can identify and change their configurations (i.e., the combination of number of cores, memory and frequency) in runtime.

We also provide an open-source energy-optimizing runtime framework to choose which configuration of a heterogeneous system to run a given application at runtime. Even though the open-source is for the experimented system including only one CPU and one GPU, the code is available and can be adjusted to heterogeneous systems containing other types of platforms as long as changing platform configurations during runtime is supported.

This study is for applications that run on one platform (e.g., CPU or GPU) at a time. The application has different executable files for different platforms (e.g., CPU or GPU) that can be chosen during runtime. For example, Rodinia benchmarks suite [4] supports programming models such as OpenCL which can provide different executable files of the same benchmark. This approach, however, can also apply to applications that can be divided to several phases. Each phase is wrapped in an executable file and can be considered as one application in REOH approach. Therefore, each phase of such applications only runs on one platform but the whole execution with different phases runs on several platforms.

The contributions of this study are as follows.

- Devise a new holistic tuning approach for heterogeneous systems using a probabilistic modeling approach, which is called REOH. In this study, we propose a method to unify the configurations

of different platform types (e.g., CPU and GPU), consider the total energy of both static and dynamic energy and devise a training data set containing 7074 samples by running a selected set of 18 applications based on the knowledge of application patterns from Berkeley dwarfs on a total of 393 system configurations.

- Validate the REOH approach on a heterogeneous system consisting of CPU and GPU, showing that REOH approach achieves the close energy consumption (i.e., within 5% different) to the optimal energy consumption by the brute-force approach when choosing the most energy-efficient system configuration for the applications while saving 17% number of sampling runs than the existing probabilistic network approaches [63].

- Develop an open-source energy-optimizing runtime framework for selecting an energy efficient configuration of a heterogeneous system for a given application at runtime. The framework takes as the input the executable files that the users want to run on a targeted heterogeneous system. Then the framework will choose an appropriate configuration of the targeted heterogeneous system to run the executable files energy-efficiently. This tool is provided as an open source for scientific research purposes.

1.3 Thesis Roadmap

The content of this thesis is organized as follows. Chapter 2 explains the background and important concepts mentioned in this thesis. The details of the three modeling approaches are reported in Chapter 3, 4 and 5. Chapter 3 describes the power models that provide the exact power estimation to support energy efficient co-design on ultra-low-power embedded systems. Chapter 4 presents the energy-complexity models to analyze the energy consumption of multithreaded algorithms. Chapter 5 explains the runtime energy optimization approach and framework to predict the most energy-efficient configurations for heterogeneous systems. Chapter 6 concludes the thesis and discusses the future work.

Chapter 2

Background

In this chapter, we give the descriptions of the concepts that the thesis work concerns. First, we explain the general concepts related to energy modeling including power, energy, energy efficiency, and the roles of energy models in Section 2.1. Second, the energy and power management techniques (i.e., DVFS and RTH) discussed in RQ1 are introduced in the Section 2.2. Then, the concepts related to parallel computing (i.e., multithreaded algorithms and application patterns) and used in RQ2 are described in Section 2.3. Finally, the concepts of homogeneous and heterogeneous computing systems mentioned in RQ3 are explained in Section 2.4.

2.1 Energy Modeling

2.1.1 Power, Energy and Energy Efficiency

Power in science is defined as the rate at which work is done per unit time and usually measured in watts. Power can be defined as $P = \frac{W}{T}$, with P denotes power, W denotes work and T denotes time.

Energy is measured in watt-hour (Wh) when the power of one watt running for one hour. Energy is defined as $E = P \times T$ where T is the period of time a power runs for.

Energy efficiency, according to the EU Energy Efficiency Directive, means "the ratio of output of performance, service, goods or energy, to input of energy" [76]. Examples of the mentioned output can be thermal comfort in a building; transport service of persons or of information as a service; and a smart phone as a good.

Since energy cost has increased dramatically and negatively impact the economy and ecology [93], improving energy efficiency is clearly a research emphasis.

2.1.2 Energy Models

For mobile and portable embedded systems, power and energy consumption is a major design constraint, where efficient power management affects the lifetime of battery. For high performance computing system, performance is also affected by energy-aware design.

Reducing energy consumption of computing systems has become one of the main research topics. Reducing energy consumption can be gained by thermal-aware hardware design or power-aware software design or the combination of both [76]. Energy-aware hardware design involves various levels from different hardware components such as memory hierarchies, interconnects and processor architecture, etc. Energy-aware software design also involves various levels, from operating systems to compiler and applications layers.

In order to improve energy efficiency and reduce the energy cost of computing systems, we need to understand how a computer system consumes energy when running different workloads. This understanding requires analysis tools to estimate how much energy a system consumes. Analysis tools can be performance or energy counter which are not always available. Modeling power and energy consumption is another alternative approach to estimate power and energy consumption. The models not only provide the estimation of power and energy cost, but also the understanding of how computing systems consume power and energy and the insight into how to reduce them.

2.2 Energy and Power Management Techniques

Traditionally, the power consumption of a CMOS integrated circuit is accounted by dynamic power and static leakage power consumption [51]. The dynamic power consumption is computed by Equation 2.1, where C is the capacitance of the transistor gates, f is the operating frequency and V is the operating voltage.

$$P = C \times f \times V^2 + P_{static} \tag{2.1}$$

2.2.1 Dynamic Voltage and Frequency Scaling

Dynamic voltage and frequency scaling (DVFS) reduce the operating frequency or the operating voltage of the processors in order to consume less power. In frequency scaling, the processor clock rate is reduced so that the processor consumes less power at the expense of reduced performance. When a frequency is reduced, the number of instructions run by processors per unit of time is reduced and therefore, performance decreases. In dynamic voltage scaling, the operating voltage is reduced so that the power consumption is also reduced. Frequency scaling and dynamic voltage scaling often work in conjunction since adjusting the operating frequency is related to the operating voltage. Voltage scaling is more advantageous because power consumed by a processor is directly proportional to the square of voltage values as in Equation 2.1.

Since there is static leakage power consumption, the reduced performance from reducing frequency or voltage increases static energy consumption. Therefore, DVFS is usually used when the workload is not CPU-bound. Previous research has proposed to use DVFS to reduce the energy consumption of processors [73, 88]. However, the energy advantage of using DVFS are diminishing in modern architectures due to several factors such as better memory performance, advanced idle/sleep modes and complexity of multi-core processors [51].

2.2.2 Sleep states/ Race-to-halt

Race-to-halt is another power management approach where workloads are run as fast as possible to finish earlier, then some parts of the hardware (e.g., processor, caches, DRAM) are put into sleep states or its lowest operating frequency to save energy. This process can repeat multiple times during a workload execution. That means the systems runs with its highest setting to finish the task, and then wait for another job without being halt. Race-to-halt aims to reduces the static leakage energy.

DVFS is usually used for memory-bound workload while Race-to-halt is used for CPU-bound workload. However, which power management approach is better depends on both workload patterns and the underlying hardware.

2.3 Parallel computing

A parallel computing system is a system containing and using multiple processors simultaneously to solve a computational problem by splitting a computing task into several subtasks and assign each processor (e.g., CPU or core) to solve each subtask.

In the scope of parallel computing, there are important concepts that are mentioned in this thesis, including multithreaded algorithms, application patterns, data-intensive and computation-intensive applications.

2.3.1 Multithreaded Algorithms

Multithreaded algorithms are algorithms that are designed for a computing system with multiple processors (e.g., CPU or core) and a shared memory. Multithreaded computation can be modeled by a computation DAG (Directed Acyclic Graph) represented by $G = (V, E)$ where V is a set of nodes represented for operations/instructions and E is a set of edges represented for the dependencies of the nodes [21]. Along with the definition of DAG, there are concepts of two metrics: **work** and **span**, which are the indications of the theoretical efficiency of a multithreaded algorithm. The **work** is the total time to execute the whole computation on one processor. The **span** is the time to execute the longest or the critical path in the DAG. The parallelism of the multithreaded computation is computed as the ratio of its **work** to its **span**.

2.3.2 Application Patterns

Asanovic et al.[12] have introduced classes (dwarfs) of computational methods which captures computation and communication common patterns of applications. They are the most common patterns in diverse sets of domains such as machine learning, graphics, database, etc. The classes are defined by the similarity in computation and data movement. Each dwarf is the high level of abstractions across a class of applications. The dwarfs and their example applications are as below:

- Dense Linear Algebra (E.g., Body Tracking, Kmeans)
- Sparse Linear Algebra (E.g., Support vector machines, quadratic programming)
- Spectral Methods (E.g., spectral clustering, FFT)
- N-Body Methods (E.g., Molecular dynamics)
- Structured Grids (E.g., GemsFDTD, Maxwell EM)
- Unstructured Grids (E.g., Belief propagation, Global illumination)
- Map Reduce (E.g.,Monte Carlo, Ray tracer)
- Combinational Logic (E.g., Hashing, IP Packet, Route Lookup)
- Graph Traversal (E.g., Bayesian networks, Decision trees)
- Dynamic Programming (E.g., Query optimization, SPEC Integer: Go)
- Backtrack and Branch+Bound (E.g., Kernel regression, 2D Path finding library)
- Construct Graphical Models (E.g., Hidden Markov models, Viterbi Decode)
- Finite State Machine (E.g., EEMBC Networking: QoS, SPECT Integer: text processing (perlbench))

Understanding whether the dwarfs are limited by computation or by memory is essential to make use of the architecture. This insight also helps to develop future architectures.

The applications can also be classified as data-intensive or compute-intensive. The applications considered as data-intensive when a limit factor of CPU power is the amount of data, the complexity of data and its changing speed [37]. An example of data-intensive application is sparse matrix multiplication which has a high demand for data transfer from memory. Compute-intensive applications are applications demanding high computation such as matrix multiplication.

2.4 Computing Systems

This section discusses the definitions of homogeneous and heterogeneous systems.

2.4.1 Homogeneous Systems

According to Lastovetsky et.al [8], there are three types of homogeneity:

- Homogeneous machine: a hardware whose each processor ensure the same storage presentation and guarantees the same results of operations on floating-point numbers.
- Homogeneous network: a collection of homogeneous machines where the communication layer among all processors ensures the exact transmittal of the floating-point values.
- Homogeneous computing environment: a platform where the softwares on each processor ensure the same storage representation and the same results of operations on floating-point numbers.

2.4.2 Heterogeneous Systems

Heterogeneous systems refer to the systems that include different types of computational units or processors and do not satisfy the homogeneity. E.g., the differences can come from unlike instruction set architectures, communication layer among processors, operation systems or compilers. The combinations of many different kinds of hardware and software aim to solve computation problems more efficiently. Heterogeneous systems exploit the advantages of each included hardware by using specialized processing capabilities for particular tasks and increases their performance and energy efficiency. Heterogeneous systems have more complex architectures and therefore, is more challenging to understand and model their performance and energy consumption.

Chapter 3

Power Models Supporting Energy-efficient Co-design on Ultra-low-Power Embedded Systems

The energy efficiency of computing systems can be enhanced via power models that provide insights into how the systems consume power. However, there are no application-general, fine-grained and validated power models which can provide insights into how a given application running on an ultra-low-power (ULP) embedded system consumes power.

This chapter of the thesis answers the Research Question 1 introduced in Chapter 1, Section 1.1.1: *"RQ1: How to accurately model and estimate the power and energy consumptions and support energy-efficient co-design of ultra-low-power embedded systems?"*. Section 1.1.1 also presents the state of the art of power and energy models of computing systems.

In this chapter, we devise new fine-grained power models that provide insights into how a given application consumes power on an ULP embedded system. The models support architecture-application co-design by considering both platform and application properties. The models are validated with data from 35 micro-benchmarks and three application kernels, namely dense matrix multiplication, sparse matrix vector multiplication and breadth first search, on Movidius Myriad, an ultra-low-power embedded system. The absolute percentage errors of the model are at most 8.5% for micro-benchmarks and 12% for application kernels. Based on the models, we propose a framework predicting when to apply race-to-halt (RTH) strategy (i.e., running an application with a maximum setting) to a given application. For the three validated application kernels, the proposed framework

Table 3.1: Descriptions of SHAVE Components

Component	Description
IRF	Integer register file
SRF	Scalar register file
VRF	Vector register file
IAU	Perform integer arithmetic operations
SAU	Perform scalar integer/floating point arithmetic operations
VAU	Perform vector integer/floating point arithmetic operations
LSU0	Perform memory access and IO instructions
LSU1	Perform memory access and IO instructions
CMU	Compare/move data among register files

is able to predict when to use RTH and when not to use RTH precisely. The experimental results show that we can save up to 61% energy for dense matrix multiplication, 59% energy for sparse matrix vector multiplication by using RTH and 5% energy for breadth first search by *not* using RTH.

The content of this chapter is organized as follows. Section 3.1 describes Myriad, the ULP embedded systems that are used to validate the proposed models. Section 3.2 presents the proposed power models. The model validation is described in Section 3.3. Section 3.4 presents the RTH framework and its validation. Section 3.5 concludes the chapter.

3.1 Movidius Myriad Platform

This section describes briefly Myriad platform which is used to devise and validate the new RTH-power models. Myriad is developed by Movidius company and belongs to a new generation of ultra-low-power processors in the mobile computing industry [40]. Myriad does not support DVFS. Instead, it supports power on/off each individual core.

In terms of processors, Myriad chip contains eight separate SHAVE (Streaming Hybrid Architecture Vector Engine) cores and one RISC core named LEON. Each SHAVE core resides on one solitary power island. A SHAVE core contains a set of register files (e.g., IRF, SRF and VRF) and a set of operation units, including arithmetic units (e.g., IAU for integer, SAU for scalar, VAU for vector) and load/store units (e.g., LSU0 and LSU1). The architecture of a SHAVE core is shown in Figure 3.1. Eight SHAVE cores can access Double Data Rate Random Access Memory (DDR RAM) via L2 cache or bypass L2 cache.

In this work, we consider the register files and operation units as described in Table 3.1. Myriad platform has an ultra-low-power design [1]. Myriad can provide theoretical peak energy efficiency at 45 Gflops/Watt [40]. It supports power management allowing individual cores to be powered on/off.

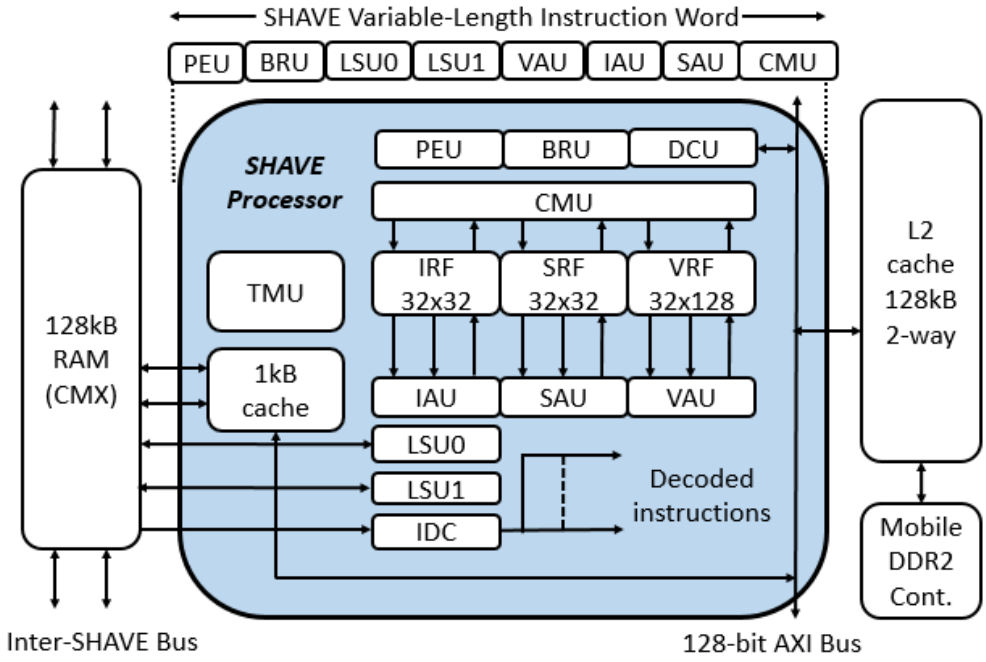


Figure 3.1: SHAVE Operation Units

However, the trade-off of Myriad energy efficiency is its difficulty to program. In order to achieve high energy efficiency, programmers have to write programs in an assembly language [40] and use control instruction pipelines and direct memory access (DMA) transfers between different memory modules. The scheduling part and computation part of applications are written on separate code files. The scheduling part is on LEON core and the computation part is on SHAVE cores. The compiler is a subset of GCC compiler and therefore, not all libraries are supported. This high complexity to program is also noticed in other ULP systems [69].

3.1.1 Measurement Set-up

We use a bench setup consisting of one Myriad MV153 board, one DC step down converter and one HAMEG multimeter to measure voltage, current and consumed power. The MV153 board is modified in order to measure the voltage, current and consumed power of only Myriad chip instead of the whole board as shown in Figure 3.2. For execution time, we can measure workload duration via a C library provided for Myriad1 platform.

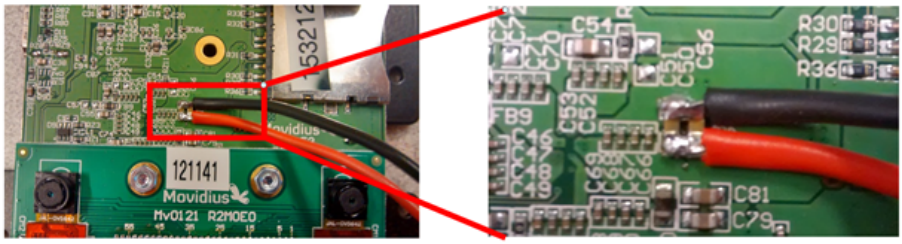


Figure 3.2: Myriad Power Supply Modification

3.1.2 Micro-benchmarking Methodology

We design different micro-benchmark suites and run each micro-benchmark with different numbers of cores (i.e., 1, 2, 4 and 8) to collect the power data. The micro-benchmarks are written in assembly language (ASM) to make the experimental results close to the actual hardware performance. In this study, we create a set of 35 micro-benchmarks categorized into two groups, i.e., arithmetic micro-benchmarks and intensity-based micro-benchmarks. Each arithmetic micro-benchmark runs one or multiple arithmetic units in parallel. For instance, SauXor runs exclusive-OR instruction on SAU unit. The intensity-based micro-benchmarks run both arithmetic unit and load store unit to simulate real applications. The ratio of arithmetic instructions to load-store instructions identifies the arithmetic intensity of micro-benchmarks. The micro-benchmark suites are elaborated more in Section 3.3.1.

3.2 RTHpower - Analytical Power Models

For a specific-purpose and novel ULP architectures such as Myriad1 platform, the supported hardware or software libraries to measure energy consumption is not available yet. However, as described in Section 3.1.1, we can measure power consumption by using an external multimeter. Therefore, we decided to measure and model power consumption for each operation unit instead of energy consumption.

Although the RTHpower models are devised and validated on a specific system (i.e., Myriad), the procedure to retrieve the models parameters is applicable to other ULP embedded systems as long as there is a mean to measure the power consumption of micro-benchmarks on the considered systems. In this Section, we first present a new power model for operation units and then develop it to the RTHpower models considering application properties.

3.2.1 A Power Model for Operation Units

The experimental results of the micro-benchmarks suite for operation units show that the power consumption of Movidius Myriad platform is ruled by Equation 3.3. In the equation, the static

Table 3.2: $P^{dyn}(op)$ of SHAVE Operation Units

Operation	Description	P^{dyn} (mW)
SAUXOR	Perform bitwise exclusive-OR on scalar	15
SAUMUL	Perform scalar multiplication	18
VAUXOR	Perform bitwise exclusive-OR on vector	35.6
VAUMUL	Perform vector multiplication	52.6
IAUXOR	Perform bitwise exclusive-OR on integer	15
IAUMUL	Perform integer multiplication	21
CMUCPSS	Copy scalar to scalar	20
CMUCPIVR	Copy integer to vector	13
LSULOAD	Load from a memory address to a register	28
LSUSTORE	Store from a register to a memory address	37

power P^{sta} is the required power when the Myriad chip is on, including memory storage power. The active power P^{act} is the power consumed when a core is on and actively performing computation work. The dynamic power $P^{dyn}(op)$ is the power consumed by each operation unit such as arithmetic units (e.g., IAU, VAU, SAU, CMU) or load/store units (e.g., LSU0, LSU1) in one SHAVE. In this power model, we focus on the experiments with the benchmarks performing different arithmetic operations such as IAU, VAU, SAU and CMU. The total dynamic power of a core is the sum of all dynamic power from involved units. If benchmarks or programs are executed with n cores, the active and dynamic power needs to be multiplied with the number of used cores.

The experimental results show that different operation units have different $P^{dyn}(op)$ values as listed in Table 3.2. The mean value of P^{sta} and P^{act} from all micro-benchmarks are computed as in Equation 3.1 and Equation 3.2. Table 3.3 provides the description of parameters and their values in the proposed models.

$$P^{sta} = 62.125mW \tag{3.1}$$

$$P^{act} = 30mW \tag{3.2}$$

$$P^{units} = P^{sta} + n \times \left(P^{act} + \sum_{i} P_i^{dyn}(op) \right) \tag{3.3}$$

3.2.2 RTHpower Models for Applications

Since typical applications require both computation and data movement, we use the concept of arithmetic intensity proposed by Williams et al. [87] to characterize applications. An application can be characterized by the amount of computational work W and data transfer Q. W is the number

Table 3.3: Model Parameter List

Parameter	Description
P^{sta}	Static power of a whole chip
P^{act}	Active power of a core
$P^{dyn}(op)$	Dynamic power of an operation unit
P^{LSU}	Dynamic power of Load Store Unit
P^{ctn}	Contention power of a core waiting for data
m	Average number of active cores accessing data
n	Number of assigned cores to the program
I	Arithmetic intensity of an application
α	Time ratio of data transfer to computation

of operations performed by an application. Q is the number of transferred bytes required during the program execution. Both W and Q define the arithmetic intensity I of applications as in Equation 3.4. Characterizing applications by their intensity values is a conventional approach used in recent energy and performance modeling studies [19, 18, 66].

$$I = \frac{W}{Q} \tag{3.4}$$

As the time required to perform one operation is different from the time required to transfer one byte of data, we introduce a parameter to the models: time ratio α of transferring one byte of data to performing one arithmetic operation. Ratio α is the property of an application on a specific platform and its value depends on the application.

Since the time to access data and time to perform computation work can be overlapped, during a program execution, the core can be in one of the three states: performing computation, performing data transfer or performing both computation and data transfer in parallel. An application either has data transfer time longer than computation time or vice versa. Therefore, there are two models for the two cases.

- If an application has data transfer time longer than computation time, it is memory-bound and follows Equation 3.5. The execution can be modeled as two (composed) periods: one is when computation and data transfer are performed in parallel and the other is when only data transfer is performed. Fraction $\frac{W}{\alpha \times Q}$ represents the overlapped time of computation and data transfer. Fraction $\frac{\alpha \times Q - W}{\alpha \times Q}$ represents the remaining time for data transfer.

$$P = P^{comp||data} \times \frac{W}{\alpha \times Q} + P^{data} \times \frac{\alpha \times Q - W}{\alpha \times Q} \tag{3.5}$$

- If an application has computation time longer than data transfer time, it is compute-bound

and follows Equation 3.6. The execution can be modeled as two periods: one is when computation and data transfer are performed in parallel and the other is when only computation is performed. Fraction $\frac{\alpha \times Q}{W}$ represents the overlapped time of computation and data transfer. Fraction $\frac{W-\alpha \times Q}{W}$ represents the remaining time for computation.

$$P = P^{comp||data} \times \frac{\alpha \times Q}{W} + P^{comp} \times \frac{W - \alpha \times Q}{W} \tag{3.6}$$

After converting W and Q to I by using Equation 3.4, the final models are simplified as Equation 3.7 and Equation 3.8,

$$P = P^{comp||data} \times \frac{I}{\alpha} + P^{data} \times \frac{\alpha - I}{\alpha} \tag{3.7}$$

$$P = P^{comp||data} \times \frac{\alpha}{I} + P^{comp} \times \frac{I - \alpha}{I} \tag{3.8}$$

where P^{data}, P^{comp}, and $P^{comp||data}$ are explained below:

Data transfer power P^{data}

P^{data} is the power consumed by the whole chip when only data transfer is performed. P^{data} is computed by Equation 3.9. In Equation 3.9, P^{sta} is the static power; P^{act} is the active power; n is the number of active cores assigned to run the application; m is the average number of cores accessing data in parallel during the application execution; contention power P^{ctn} is the power overhead occurring when a core waits for accessing data because of the limited memory ports (or bandwidth) or cache size in the platform architecture. Therefore, $n - m$ is the average number of cores waiting for memory access during the application execution. The data transfer power is the sum of the static power of the whole chip, the active power and dynamic power of the cores accessing data, and the contention power of the cores waiting for accessing data.

$$P^{data} = P^{sta} + min(m, n) \times (P^{act} + P^{LSU}) + max(n - m, 0) \times P^{ctn} \tag{3.9}$$

Computation power P^{comp}

P^{comp} is the power consumed by the whole chip when only computation is performed. P^{comp} is computed by Equation 3.10. Each core runs its arithmetic units (e.g. IAU, SAU, VAU) to perform computation work. There is no contention power due to no memory access. Therefore, all assigned cores are active and contribute to total power. The computation power is the sum of the static power of the whole chip, the active power and dynamic power of the cores performing computation work.

$$P^{comp} = P^{sta} + n \times (P^{act} + \sum_i P_i^{dyn}(op)) \tag{3.10}$$

Computation and data transfer power $P^{comp||data}$

$P^{comp||data}$ is the power consumed by the whole chip when computation and data transfer are performed in parallel. $P^{comp||data}$ is computed by Equation 3.11. In this case, there is contention power due to the data waiting. $P^{comp||data}$ is different from P^{data} in the aspect that the active cores also run arithmetic units that contribute to total power as $\sum_i P_i^{dyn}(op)$. To simplify the model, it is assumed that only the cores which has accesses to data can perform computation work and the cores which does not has accesses to data also wait to perform computation work. Therefore, the computation and data transfer power is the sum of the static power of the whole chip, the active power and dynamic power of the cores performing computation work and data transfer and the contention power of the cores waiting for accessing data.

$$P^{comp||data} = P^{sta} + min(m,n) \times (P^{act} + P^{LSU} + \sum_i P_i^{dyn}(op)) + max(n-m,0) \times P^{ctn} \quad (3.11)$$

The completed power models

After replacing P^{comp}, P^{data}, $P^{comp||data}$ from Equation 3.10, 3.9 and 3.11, respectively to Equation 3.7 and 3.8, the detailed equations to compute total power consumption are Equation 3.12 for memory-bound applications and Equation 3.13 for compute-bound applications.

$$\begin{aligned} P =& P^{sta} \\ &+ (min(m,n) \times (P^{act} + P^{LSU} + \sum_i P_i^{dyn}(op)) + max(n-m,0) \times P^{ctn}) \times \frac{I}{\alpha} \\ &+ (min(m,n) \times (P^{act} + P^{LSU}) + max(n-m,0) \times P^{ctn}) \times \frac{\alpha - I}{\alpha} \end{aligned} \quad (3.12)$$

$$\begin{aligned} P =& P^{sta} \\ &+ (min(m,n) \times (P^{act} + P^{LSU} + \sum_i P_i^{dyn}(op)) + max(n-m,0) \times P^{ctn}) \times \frac{\alpha}{I} \\ &+ (n \times (P^{act} + \sum_i P_i^{dyn}(op))) \times \frac{I - \alpha}{I} \end{aligned} \quad (3.13)$$

3.3 Model Training and Validation

This section presents the experimental results including two sets of micro-benchmarks (i.e., Operation Units and Application Intensities) and three application kernels (i.e., *matmul*, SpMV and BFS) that are used for training and validating the models.

3.3.1 Model Validation with Micro-benchmarks for Operation Units

Analyses of experimental results are performed based on a set of micro-benchmarks: 26 micro-benchmarks for operation units called *unit-suite*. The micro-benchmarks of *unit-suite* are listed in Table 3.4. The measured power data is collected by executing each micro-benchmark with different numbers of cores (i.e., 1, 2, 4, and 8 cores).

The assembly files used in the validation process have a fixed number of instructions in the loop for all the tests, meaning that each assembly file contains six instructions in the loop that is infinitely repeated. This convention was made in order to keep continuity and consistency of experiments, enabling the comparisons among different SHAVE units.

The power model for operation units (Equation 3.3) is trained and validated with measurement data from *unit-suite*. The model is trained with the power data collected by running *unit-suite* with one and two cores. By using measurement data of experiments running the 26 micro-benchmarks in unit-suite with one and two cores, we identify the static power of the platform, the dynamic power $P^{dyn}(op)$ of each operation unit and the active power of a SHAVE core.

First, when running the same benchmark on different numbers of SHAVE cores, we can identify the sum of SHAVE active power P^{act} and its dynamic power P^{dyn} which is the power difference of the two runs (e.g., the run with one SHAVE core and the run with two SHAVEs). Given the sum of P^{act} and P^{dyn}, P^{sta} is derived from Equation 3.3. The mean value of P^{sta} from all micro-benchmarks are computed as in Equation 3.1.

Second, dynamic power of a SHAVE core running multiple units is the sum of the dynamic power of all involved arithmetic units. E.g. $P^{dyn}_{(SauIau)}) = P^{dyn}_{(Sau)} + P^{dyn}_{(Iau)}$. For each operation unit, we obtain the two parameters P^{dyn}_{op} and P^{act} by the power consumption of the benchmark for individual units and multiple units. For example, P^{dyn}_{Iau}, P^{dyn}_{Sau}, and P^{act} can be identified from Equations 3.14, 3.15, 3.16. The dynamic power $P^{dyn}(op)$ of each operation unit is listed in Table 3.2.

$$P^{dyn}_{(Iau)} = P^{sta} + n \times (P^{act} + P^{dyn}_{(Iau)}) \tag{3.14}$$

$$P^{dyn}_{(Sau)} = P^{sta} + n \times (P^{act} + P^{dyn}_{(Sau)}) \tag{3.15}$$

$$P^{dyn}_{(SauIau)} = P^{sta} + n \times (P^{act} + P^{dyn}_{(Sau)} + P^{dyn}_{(Iau)}) \tag{3.16}$$

Then, the mean value of P^{act} from all micro-benchmarks are also computed as in Equation 3.2.

After computing the values of $P^{dyn}(op)$, P^{sta} and P^{act}, the power measurement data collected by running *unit-suite* with four and eight cores are used to validate the model. We validate the model and plot its percentage errors in Figure 3.3 and 3.4. Percentage error is calculated as $PE = \frac{measurement - estimation}{measurement}$. Figure 3.3 and 3.4 show the percentage error of all three categories: one unit, two pipe-lined units and three pipe-lined units. The *absolute* percentage error is the absolute value of the percentage error. The model for operation units has the absolute percentage errors at

Table 3.4: Micro-benchmarks for Operation Units

Description	Micro-benchmark Name
10 micro-benchmarks using one unit (cf. Table 2)	SAUXOR, SAUMUL, IAUXOR, IAUMUL, VAUXOR, VAUMUL, CMUCPSS, CMUCPIVR, LSULOAD, LSUSTORE
15 micro-benchmarks using two units	SAUXOR-CMUCPSS, SAUXOR-CMUCPIVR, SAUXOR-IAUMUL, SAUXOR-IAUXOR, SAUXOR-VAUMUL, SAUXOR-VAUXOR, SAUMUL-IAUXOR, IAUXOR-VAUXOR, IAUXOR-VAUMUL, IAUXOR-CMUCPSS, LOAD-STORE, DUALLOAD, DUALSTORE, SAUXOR-LOAD, SAUXOR-STORE
1 micro-benchmarks using three-units	SAUXOR-IAUXOR-CMUCPSS

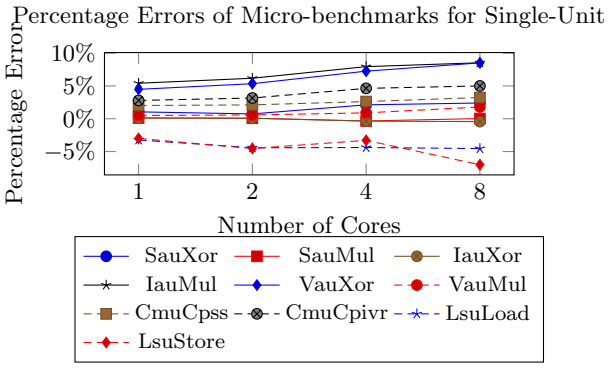


Figure 3.3: The percentage errors of the model validation for *unit-suite* for 10 one-unit micro-benchmark.

most 8.5%. The absolute percentage errors for one-unit operations and multiple-unit operations are 8.5% and 6%, respectively. These results prove that the model is applicable to micro-benchmarks using either a single (e.g., performing bitwise exclusive-OR on scalar unit: SauXor) or pipe-lined arithmetic units in parallel (e.g., performing Xor on scalar and integer units, in parallel with copying from scalar to scalar unit: SauXorCmuCpssIauXor). The model also shows the compositions of the power consumption not only for multiple cores but also for multiple operation units within a core.

The source of errors (i.e., 8.5% for one-unit operations and 6% for multiple-unit operations) can come from several factors. One factor is that the measurement data is read manually from the multimeter display. The power consumption values displayed by the multimeter can vary 1-2% during the execution of micro-benchmarks (e.g., if the displayed value varies from 128-130mW, the read value is the average as 129mW).

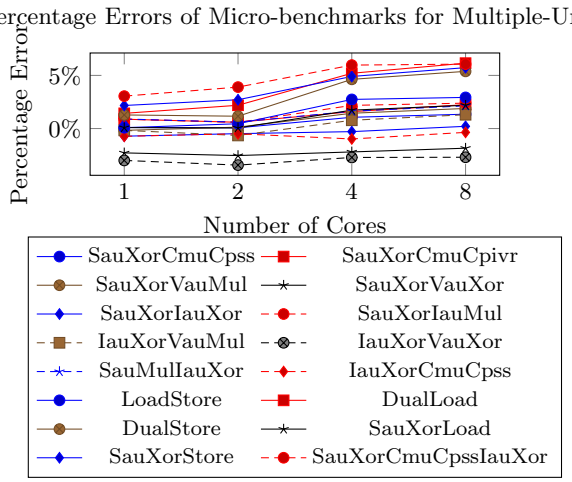


Figure 3.4: The percentage errors of the model validation for *unit-suite* for 16 multiple-units micro-benchmark.

3.3.2 Model Validation with Micro-benchmarks for Application Intensities

Since any application requires both computation and data movement, we design 9 intensity-based micro-benchmarks which execute both arithmetic units (e.g., SAU) and two data transfer units (e.g., LSU0, LSU1) in a parallel manner. They are implemented with parallel instruction pipeline supported by the platform. In order to validate the RTHpower models, this *intensity-suite* indicates different values of operation intensities (from 0.25 to 64). The arithmetic intensity I is retrieved from the assembly code by counting the number of arithmetic instructions and the number of load/store instructions.

In the models, there are platform-dependent parameters such as α, m and P^{ctn}. The parameter values for each application arithmetic intensity are derived from experimental results by using Matlab function $lsqcurvefit$. For the application intensities from 0.25 to 1, α is found bigger than arithmetic intensity I meaning that data transfer time is longer than computation time. The estimated power model follows Equation 3.7. For arithmetic intensity from 2 to 64, α is less than I meaning that data transfer time is less than computation time. The estimated power follows Equation 3.8.

We plot the percentage errors of the model fitting for intensity-based micro-benchmarks in Figure 3.5. The absolute percentage errors of intensity-based micro-benchmarks are at most 7% for the arithmetic intensity varying from 0.25 to 64. A parameter set, including α, P^{ctn}, and m is identified for each intensity value as in Table 3.5.

In order to obtain a full range of estimated power with a full value range of intensities and numbers of cores, a fuzzy logic approach, namely Takagi Sugeno Kang (TSK) mechanism [75], is applied to the RTHpower models.

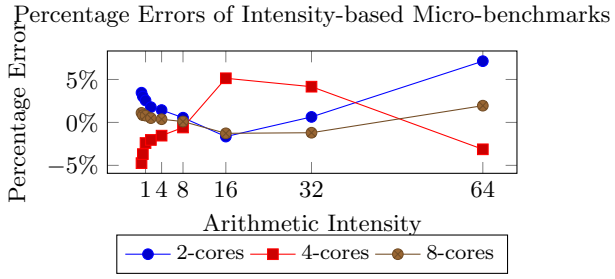


Figure 3.5: The absolute percentage errors of RTHpower model fitting for *intensity-suite*

Table 3.5: Model parameters values

Intensity	m	P^{ctn}	α
0.25	0.72	9.89	0.93
0.5	0.57	15.64	2.27
1	0.57	18.39	1.87
2	0.80	7.77	0.94
4	0.83	0.97	1.69
8	1.05	0.9	2.43
16	0.001	0.03	2.03
32	0.001	0.05	2.15
64	3.17	0.001	2.62

The Takagi-Sugeno-Kang Fuzzy Model

The Takagi-Sugeno-Kang fuzzy model (also known as the TSK fuzzy model) was proposed by Takagi, Sugeno, and Kang in order to approximate any nonlinear behavior. A fuzzy TSK model is constructed with a set of fuzzy rules. A typical fuzzy rule in a TSK fuzzy model with two inputs (a and b) and one output (y) has the form:

$$\text{If } a \text{ is } A_i \text{ and } b \text{ is } B_i \text{ then } y = f_i(a, b) \tag{3.17}$$

where A_i and B_i are the fuzzy set of the variables a and b, respectively, and $f_i(a, b)$ is the predefined function for rule i. By assuming that singleton fuzzier, product inference and centroid defuzzifier techniques, the output of the TSK fuzzy model can be calculated by aggregating n_r fuzzy rules:

$$y = \sum_{i=1}^{n_r} v_i(a, b) \cdot f_i(a, b) \tag{3.18}$$

Table 3.6: Centers of the intensity membership functions

i	1	2	3	4	5	6	7	8	9
$c_{I\{i\}}$	0.25	0.5	1	2	4	8	16	32	64

where v_i is the normalized firing strength of rule i:

$$v_i = \frac{w_{ai}(a) \cdot w_{bi}(b)}{\sum_{i=1}^{n_r} w_{ai}(a) \cdot w_{bi}(b)} \tag{3.19}$$

where $w_{ai}(a)$ $(i = 1..n_r)$ are the membership degrees that quantify the grade of membership of a to the fuzzy set A_i.

Power Estimation Using Fuzzy TSK Model

By using fuzzy TSK models, we provide a method to estimate the power for benchmarks which have the arithmetic intensities that are not from the available intensities in Table 3.5. Instead, the estimation is based on the TSK fuzzy logic mechanism by blending individual power functions of the available intensities. An individual power function is the RTHpower models as Equation 3.12 or Equation 3.13 with each parameter set provided in Table 3.5. The fuzzy model has two inputs: the intensity I and the number of cores n. The power is estimated with eight fuzzy rules having the following form:

$$\text{If } I \text{ is } I_i \text{ then } P = f_i(I, n) \tag{3.20}$$

where I_i $(i = 1, ..9)$ is the fuzzy set of the intensity for rule i and $f_i(I, n)$ is the power function when $I = I_i$. The membership functions (MF) of the fuzzy sets I_i are triangular and are defined by the centers of MF c_{Ii} (Figure 3.6). The values of the centers of MF are listed in Table 3.6. By using fuzzy inference mechanism (as in Equation 3.18 and Equation. 3.19), the power of the unmeasured intensities can be estimated as follows:

$$P = \frac{\sum_{i=1}^{9} w_i(I) \cdot f_i(I, n)}{\sum_{i=1}^{9} w_i(I)} \tag{3.21}$$

The full range of estimated power is obtained and presented in Figure 3.7. The dots in the figure represent measurement data. It is observed that when intensity value increases, the power-up (i.e., the power consumption ratio of the application executed with n cores to the application executed with 1 core) is also increased. The small dip in the diagram is due to the switch from Equation 3.12 to Equation 3.13 at the intensity $I = 2$.

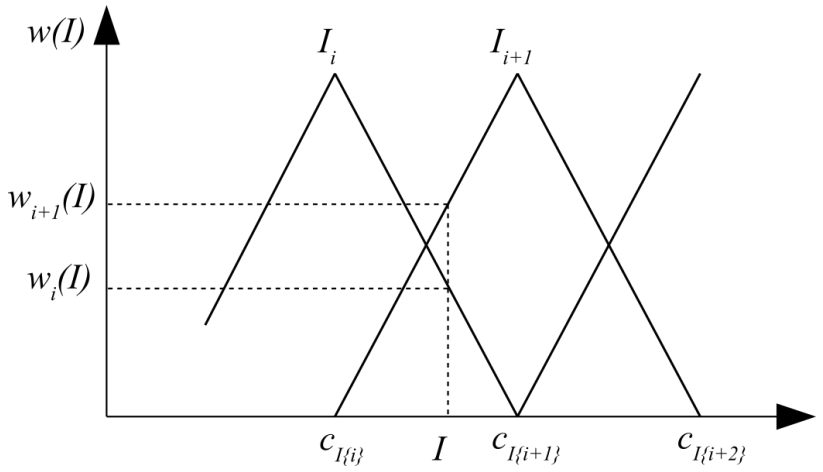


Figure 3.6: Membership functions of the intensity

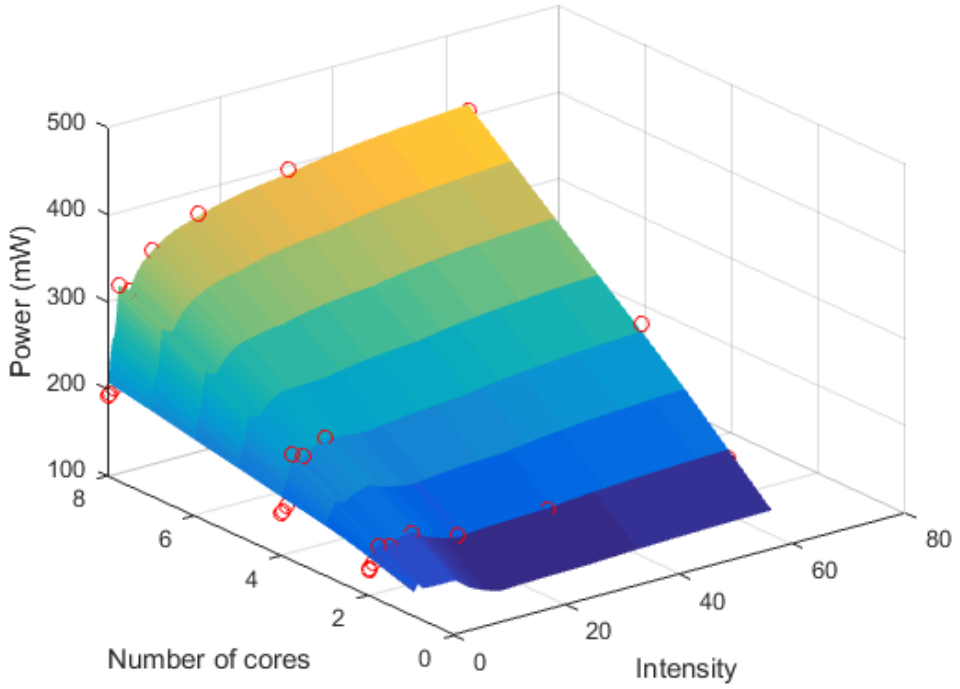


Figure 3.7: The power range of varied intensities and numbers of cores from RTHpower models.

3.3.3 Model Validation with Application Kernels

The following application kernels have been chosen to implement and validate the RTHpower models on Myriad: *matmul* (a computation-intensive kernel), SpMV (a kernel with dynamic access patterns),

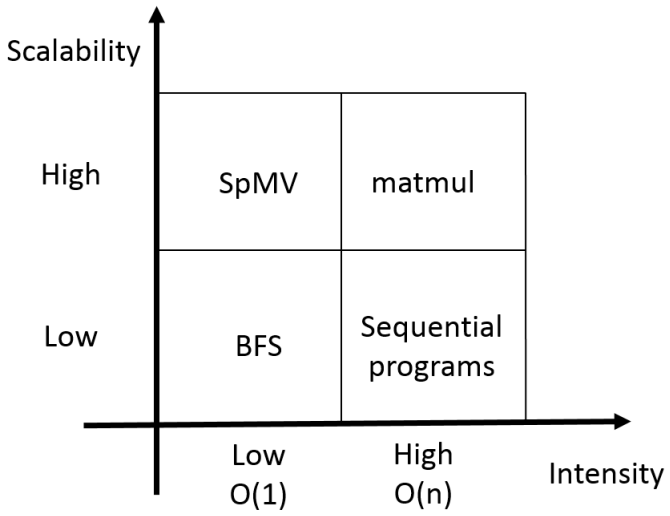


Figure 3.8: Application Categories

and BFS (a data-intensive kernel of Graph500 benchmarks [74]). All three kernels belong to the list of Berkeley dwarfs [12] and are able to cover the two dimensions of arithmetic intensity and speed-up as shown in Figure 3.8.

Matmul is proved to have high intensity and scalability [66]. SpMV has low arithmetic intensity and high speed-up due to its parallel scalability [86]. BFS, on the other hand, has low arithmetic intensity and saturated low scalability [20]. Since the available benchmark suites in literature are not executable on Myriad platform, the three mentioned kernels have been implemented by the authors using the Movidius Development Kit for Myriad. As the RTHpower models will be used to predict whether the RTH strategy is an energy efficient approach for a given application, we focus mainly on two settings: the 8-core setting representing the RTH strategy (i.e., using all available cores of Myriad) and the 1-core setting representing the other extreme (i.e., using a minimum number of cores).

Dense Matrix Multiplication

Matmul has been implemented on Myriad by using both C and assembly languages. The *matmul* algorithm computes matrix C based on two input matrices A and B $C = A \times B$. All three matrices in this benchmarks are stored in DDR RAM. Matrix elements are stored with float type equivalent to four bytes. The number of operations and data accesses are calculated based on matrix size n as $W = 2 \times n^3$ and $Q = 16 \times n^2$ [66]. The intensity of *matmul* is also varied with matrix size as $I = \frac{W}{Q} = \frac{n}{8}$. The experiments are conducted until matrix size 1024x1024, the largest size that Myriad RAM memory can accommodate.

We observe that arithmetic intensity is not enough to capture other factors affecting power

consumption such as the communication patterns and potential performance/power overheads due to the implementation. E.g., although a sequential version and a parallel version of a *matmul* algorithm have the same intensity, it is obvious that they have different communication pattern (intuitively, the sequential version does not have communication between cores). Since different parallel versions for the different number of cores have different communication patterns (e.g., sequential version vs. 8-core version), ignoring the mentioned factors contributes to the percentage errors. Therefore, we apply online-learning approaches, which are widely used to learn the characteristics of an application while it is running [63], to RTHpower models by introducing the parameter β to the models.

The parameter β is a constant value for an application running on a specific platform and represents communication patterns and potential performance/power overheads due to the implementation of an application. By running the kernels with different input sizes, the value of β can be identified from the average values of percentage errors (PE) from sample executions as in Equation 3.22. After this step, the β value is available and can be used to estimate the power consumption of the kernels on other desired input sizes as in Equation 3.23 and Equation 3.24.

$$\beta = \frac{1}{1 + \overline{PE}}. \tag{3.22}$$

$$P_{improved} = (P^{comp||data} \times \frac{I}{\alpha} + P^{data} \times \frac{\alpha - I}{\alpha}) \times \beta \tag{3.23}$$

$$P_{improved} = (P^{comp||data} \times \frac{\alpha}{I} + P^{comp} \times \frac{I - \alpha}{I}) \times \beta \tag{3.24}$$

Note that parameter β for each sequential/parallel version (e.g., 1-core version or 8-core version) is fixed across problem sizes and therefore it can be obtained during kernel installation and then saved as meta-data for each version in practice. E.g., the β values of *matmul* are $\frac{1}{1-28\%}$ for 1-core and $\frac{1}{1+13\%}$ for 8-cores. β is computed by using data from sample executions with two matrix sizes (i.e., 128x128, and 512x512). The model is then validated with power data from executions for four matrix sizes (i.e., 128x128, 256x256, 512x512, and 1024x1024). After applying the online-learning approach, the absolute percentage errors are at most 12% as shown in Figure 3.9.

Sparse Matrix Vector Multiplication

SpMV implementation on Myriad is written in C language. All input matrix and vector of this benchmark reside in DDR RAM. This implementation uses the common data layout of SpMV which is compressed sparse row (csr) format [71]. There is no random generator supported in the RISC core so five non-zero elements per row are fixed in all experiments. Each element of matrix and vector is stored with float type of four bytes. From our implementation analysis, the number of operations and accessed data are proportional to the size of a matrix dimension n as: $W = 5 \times 2 \times n$ and $Q = 5 \times 2 \times 4 \times n$. The arithmetic intensity of SpMV therefore, does not depend on matrix size

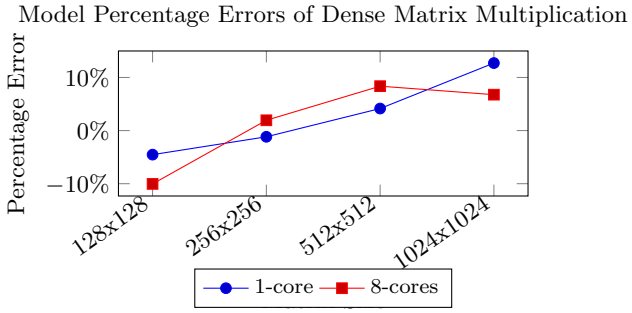


Figure 3.9: Absolute percentage errors of estimated power from measured power of *matmul*.

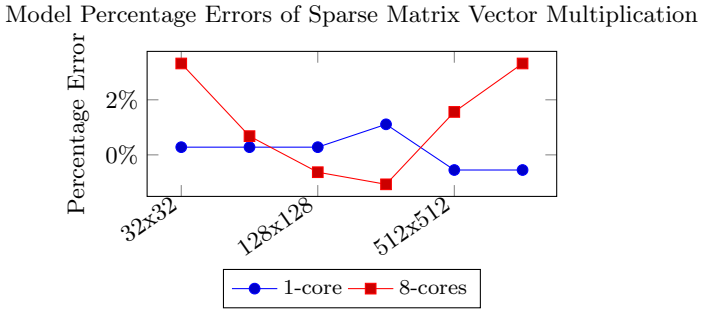


Figure 3.10: Absolute percentage errors of estimated power from measured power of SpMV.

and is a fixed value: $I = \frac{W}{Q} = 0.25$.

Figure 3.10 shows the percentage error of SpMV estimated power using Equation 3.23 and 3.24 compared to measured power. The β values for 1-core and 8-core versions of SpMV are computed from executions with three matrix sizes (i.e., 32x32, 128x128, and 512x512) as $\frac{1}{1+14\%}$ and $\frac{1}{1-8\%}$, respectively. The model is validated with test data from executions with six matrix sizes (i.e., 32x32, 64x64, 128x128, 256x256, 512x512, and 1024x1024) and has the absolute percentage errors at most 4% as shown in Figure 3.10. SpMV has lower modeling errors than *matmul* since SpMV has a fixed intensity value on different matrix sizes.

Breadth First Search

We also implemented BFS-a data-intensive Graph500 kernel, on Myriad. BFS is the graph kernel to explore the vertices and edges of a directed graph from a starting vertex. We use the implementation of current Graph500 benchmark (omp-csr) and port it to Myriad. The output BFS graphs after running BFS implementation on Myriad are verified by the verification step of the original Graph500 code to ensure the output graphs are correct.

The size of a graph is defined by its scale and edgefactor. In our experiments, we mostly use the default edge factor of 16 from the Graph500 so that each vertex of the graph has 16 edges in

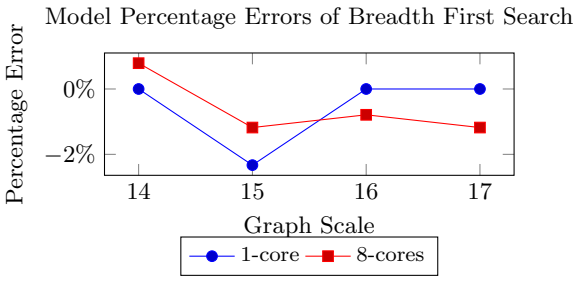


Figure 3.11: Absolute percentage errors of estimated power from measured power of BFS.

average. The graph scales are varied from 14 to 17 and the graphs have from 2^{14} to 2^{17} vertices. It is noted that graph scale 17 is the largest scale that the DDR RAM of Myriad can accommodate. From the implementation analysis, the arithmetic intensity of BFS is a fixed value: $I = \frac{W}{Q} = 0.257$ and does not depend on edge factor or scale.

Figure 3.11 shows the percentage error of BFS estimated power using Equation 3.23 and 3.24 compared to measured power. The β values for 1-core and 8-core versions of BFS are computed from executions with two graph scales (i.e., 14 and 16) as $\frac{1}{1+9\%}$ and $\frac{1}{1-18\%}$, respectively. The test set to validate the model is data from executions with four graph scales (i.e., 14-17) and has the absolute percentage errors at most 3% as shown in Figure 3.11.

3.3.4 Discussion

Modeling the absolute power consumption of the architectures is challenging and requires a significant understanding of the hardware. In this study, we attempt to estimate the power consumption of each hardware unit in the fine-grained level: cores and operation units in each core. The estimation data, however, has deviation from the actual measurement data due to the measurement approach and the made assumptions.

The measurement approach is accounted for the percentage error between the estimated data and the actual data of micro-benchmarks where measurement data is read manually from the multimeter display. The displayed values by the multimeter varying during the program execution, even within 1-2%, can affect the measurement accuracy. This measurement approach is the only approach that is applicable for Myriad1 platform when power/energy measurement library is not available.

Another source of the percentage error is the assumption that we made when finding the values of the parameters α, P^{ctn}, and m. It is assumed that SauXor operation represents for the computation instruction and LSULoad operation represents for the data movement instruction. The reason is that SauXor and LSULoad is the most used operation instruction after translated C code to assembly languages. However, since real applications contain a various set of operation instructions appeared with varied frequency, this assumption simplifies the models at the cost of some inaccuracy.

Another assumption when we validate the model with application kernels (i.e., Matmul, BFS and SpMV) is how the computational work W and data transfer Q of each kernel are computed. Since the profiling libraries to count the number of instructions on Myriad1 platform is not available, the values of W and Q are computed from static analysis based on the reading the code. Static analysis from the code does not reflect the exact values as compared to runtime profiling method and therefore, contribute to the inaccuracy of the models. This inaccuracy part of the models can also be improved with the platforms supporting profiling library and providing the exact values of W and Q.

3.4 Race-to-halt Prediction Framework

With the RTHpower models, we want to identify how many cores the system should use to run an application to achieve the least energy consumption. In order to answer the question, we need to consider the speed-up and power-up of an application on a specific platform.

From Amdahl's Law [36] the theoretical maximum speed-up of an application running on a multicore system is derived as Equation 3.25, where p denotes the fraction of the application that can be parallelized and n is the number of cores:

$$\textit{speed-up} \leq \frac{1}{(1-p) + \frac{p}{n}} \tag{3.25}$$

3.4.1 Framework Description

The purpose of this framework is to identify when to and when not to use RTH for a given application. The two required inputs for decision making are power-up and speed-up of the application executed with n cores, where n is the maximum number of cores.

- Step 1: Identify meta-data, including speed-up and arithmetic intensity, of a given application by one of the three main approaches listed: i) doing theoretical analysis to find the amount of computation work W, data transfer Q and arithmetic intensity I as well as identify the maximum speed-up of a given application; ii) executing the application on a targeted platform (e.g., Myriad) to measure its speed-up and extract its arithmetic intensity I; iii) using profiling tools [56] to extract the number of operations W and the amount of data transferred Q as well as the speed-up of an application on a common platform (e.g., Intel platform).

- Step 2: Compute power consumption of an application running with one core and with a maximum number of cores by the RTHpower models. Note that the RTHpower models are able to estimate power consumption for any number of cores by changing parameter n in the models. For verifying RTH strategy, we only need to apply the model for a single core and all cores.

- Step 3: Compare the energy consumption of the application between using one core and using a maximum number of cores to identify whether running a maximum number of cores is the most energy-efficient.

The framework is designed for kernels of libraries that will be executed several times. A kernel needs to pre-run twice (i.e., with one core and with a maximum number of cores) to find the speed-up while power-up is predicted by the RTHpower models. After deciding whether to use RTH for a kernel, the decision is beneficial for the remaining executions of the kernel.

3.4.2 Framework Validation

The framework is validated with three micro-benchmarks and three application kernels. In this validation, the values of arithmetic intensity I are extracted from the theoretical analysis of the implementations and speed-up is identified by executing the micro-benchmarks or application kernels with different numbers of cores.

Race-to-halt for Micro-benchmarks

We first validate the framework with micro-benchmarks. In this validation, we measure the power-up and speed-up of three micro-benchmarks: one with 60% parallel code, one with 100% parallel code and a small-size micro-benchmarks which has high overhead. All three micro-benchmarks have the arithmetic intensity $I = 0.25$. Namely, in the micro-benchmarks, each SauXor instruction is followed by a LsuLoad instruction which loads 4 bytes.

All three micro-benchmarks have the same assembly code wrapped inside a loop. The number of iterations to repeat the code is the difference among them. We run the micro-benchmarks on one core for 1 000 000 times. If the micro-benchmark has 100% parallel code, running it on n core requires each core performing $\frac{1}{n}$ of the amount of work (e.g., if performing the micro-benchmark on 8 cores, each core needs to run 125 000 times). Similarly, if the micro-benchmark has a parallel fraction of 60%, then running the program on n cores requires each core to perform $(1-0.6)+(\frac{0.6}{n})$ of the amount of work (e.g.,if performing the micro-benchmark on 8 cores, each core needs to run 475 000 times). For small-size micro-benchmark, the code is executed 8 times with 1 core and once with 8 cores. Since the amount of computation is small, the relative overhead of initializing the platform and executing the small-size micro-benchmark is high.

Figure 3.12 shows that the power-up of running n cores to the program running 1 core varies from 1 (1 core) to 1.71 (8 cores) for the arithmetic intensity $I = 0.25$. All three reported micro-benchmarks have speed-up less than platform power-up. If the speed-up is bigger than the power-up, RTH is an energy-saving strategy. If the speed-up is less than the power-up, running the program with the maximum number of cores consumes more energy than running it with 1 core. Note that when this happens, assigning one core to run the program is more energy-efficient and race-to-halt

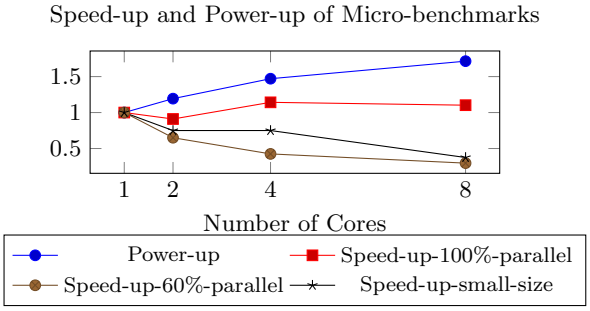


Figure 3.12: Speed-up and power-up of micro-benchmarks with the arithmetic intensity $I = 0.25$.

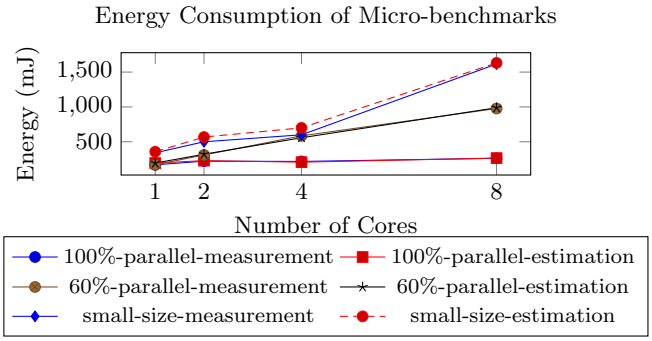


Figure 3.13: Energy consumption of micro-benchmarks with arithmetic intensity $I = 0.25$.

is no longer applicable for saving energy. For all three micro-benchmarks in this validation, the speed-up is identified by running them over different numbers of cores. The energy consumption of the three micro-benchmarks is shown in Figure 3.13. For all three reported micro-benchmarks, the programs executed with 1 core consume the least energy, compared to 2, 4, 8 cores, from both measured data and estimated data. The model estimation and actual measurement show that RTH is not applicable to the three micro-benchmarks.

Race-to-halt for Dense Matrix Multiplication

The *matmul* application has increasing values of arithmetic intensity over input sizes and its speed-up is higher than its power-up on Myriad. Therefore, running *matmul* with the 8 cores is more energy-efficient than running it with one core. Figure 3.14 shows how many percentages of energy-saving if executing *matmul* with 8 cores instead of 1 core, from both measured and estimated data. The energy saving percentage is computed based on the energy gap of running 1 core and 8 cores divided by energy consumed by running 1 core as in Equation 3.26. Since the energy-saving percentage is positive over different matrix sizes, RTH is an energy-saving strategy for *matmul*. Energy-saving

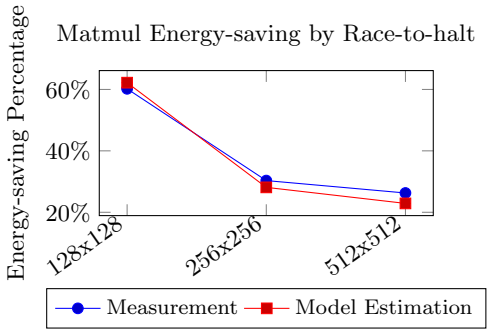


Figure 3.14: *Matmul* energy-saving by Race-to-halt.

percentage from model estimation for *matmul* has the standard deviation less than 3%.

$$ES = \frac{E^{1core} - E^{8cores}}{E^{1core}} \tag{3.26}$$

The framework can predict that RTH should be applied to *matmul* over different matrix sizes. By using RTH for *matmul*, we can save from 20% to 61% of *matmul* energy consumption. RTH is a good strategy for *matmul*. We observe that the energy saving reduces when matrix size increases due to the decrease of speed-up from size 128x128. The reason is that a matrix size bigger than 128x128 makes the data set no longer fit in the last level cache (or L2 cache of 64KB) and thereby lowers performance (in flops).

Race-to-halt for Sparse Matrix Vector Multiplication

SpMV has a fixed value of arithmetic intensity over input sizes. From the RTHpower models as well as measurement data, the power-up of SpMV is relatively constant. However, SpMV has speed-up higher than its power-up. Therefore, running SpMV with the maximum number of cores is more energy-efficient than running it with one core. Figure 3.15 shows how many percentages of energy-saving if executing SpMV with 8 cores instead of one core, from both measured and estimated data. Since the energy-saving percentage is positive over different matrix sizes, RTH is an energy-saving strategy for SpMV. The framework can predict that RTH should be applied to SpMV over different matrix sizes. By using RTH for SpMV, we can save from 45% to 59% of SpMV energy consumption. RTH is a good strategy for SpMV. The energy saving increases from size 32x32 to 128x128 since the data fits in L1 cache.

Race-to-halt for Breadth First Search

In our set of application kernels implemented on Myriad, BFS is the application kernel able to prove that running with a maximum number of cores does not always give the least energy consumption.

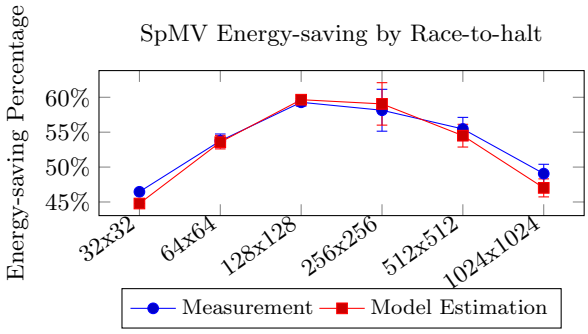


Figure 3.15: SpMV Energy-saving by Race-to-halt.

From both measured data and estimated data, the total energy consumed by running with one core is less than the total energy by running with 8 cores at scale 16 and 17. There are negative values of -5% and -3% in Figure 3.16 if running BFS with 8 cores instead of 1 core at scale 16 and 17, respectively. The RTHpower models can predict when to apply RTH for different scales. The positive percentages at scale 14 and 15 mean that RTH should be applied. The negative percentages at scale 16 and 17 mean that RTH should not be applied. The standard deviation of BFS energy-saving percentage is less than 3%, from scale 14 to 17.

The result can be explained by the relation between BFS power-up and speed-up. Since BFS has a fixed value of arithmetic intensities across graph scale, from RTHpower models (cf. Equation 3.23 and 3.24), it is understood that BFS power consumption does not depend on graph scales and its power-up is a fixed value. From the measurement results, we also observe that BFS power-scalability is relatively constant over the graph scales. However, BFS speed-up in our experiments decreases when the scale increases. The reason is that with the same graph degree, when scale increases, the graph becomes more sparse and disconnected. Compared to the Graph500 implementation, BFS search on Myriad are performed from a chosen subset of source nodes. The speed-up then becomes less than power-up at scale 16 and 17. Therefore, running BFS with 8 cores at bigger graph scales (i.e., 16 and 17 in our experiments) consumes more energy than running BFS with one core.

3.5 Conclusion

In this chapter, new fine-grained power models have been proposed to support architecture-application co-design. The models provide insights into how a given application consumes energy when executing on an ultra-low-power embedded system by considering both platform and application properties. The models have been validated on Movidius Myriad, an ultra-low-power embedded platform with data from 35 micro-benchmarks and three application kernels. We have also shown that by using the models, we can predict whether RTH is energy-efficient for an application on a ULP embedded

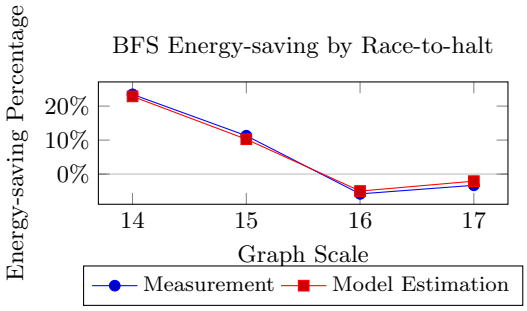


Figure 3.16: BFS Energy-saving by Race-to-halt.

system. We presented real scenarios when to use RTH and when not to use RTH and the framework based on the models could predict the scenarios precisely.

In the next chapter, we describe the energy complexity models which also consider both application and architecture properties but are more general to a wide range of high-performance platforms (e.g., CPU, GPU).

Chapter 4

ICE: A General and Validated Energy Complexity Model for Multithreaded Algorithms

Like time complexity models that have significantly contributed to the analysis and development of fast algorithms, energy complexity models for parallel algorithms are desired as crucial means to develop energy efficient algorithms for ubiquitous multicore platforms. Ideal energy complexity models should be validated on real multicore platforms and applicable to a wide range of parallel algorithms. However, existing energy complexity models for parallel algorithms are either theoretical without model validation or algorithm-specific without the ability to analyze energy complexity for a wide range of parallel algorithms. Section 1.1.2 provides the literature review of analytic models for energy consumption of multithreaded algorithms.

This chapter of the thesis answers the Research Question 2 introduced in Chapter 1, Section 1.1.2: *"RQ2: Given two parallel algorithms A and B for a given problem, which algorithm consumes less energy analytically?"*.

This chapter presents a new general validated energy complexity model for parallel (multithreaded) algorithms. The new model abstracts away possible multicore platforms by their static and dynamic energy of computational operations and data access, and derives the energy complexity of a given algorithm from its *work*, *span* and *I/O* complexity. The new model is validated by different sparse matrix vector multiplication (SpMV) algorithms and dense matrix multiplication (matmul) algorithms running on high-performance computing (HPC) platforms (e.g., Intel Xeon and Xeon Phi). The new energy complexity model is able to characterize and compare the energy consumption of SpMV and matmul kernels according to three aspects: different algorithms, different input matrix types and different platforms. The prediction of the new model regarding which algorithm

consumes more energy with different inputs on different platforms is confirmed by the experimental results. In order to improve the usability and accuracy of the new model for a wide range of platforms, the platform parameters of ICE model are provided for eleven platforms including HPC, accelerator and embedded platforms.

This chapter is organized as follows. Section 4.1 explains the shared memory machine model used in this study while Section 4.2 presents our energy complexity models ICE. Then, the two case studies (i.e., SpMV and matmul) to demonstrate how to apply the ICE model to find energy complexity of parallel algorithms are described in Section 4.3 and 4.4. In Section 4.5, we validate the ICE model with the experiment results.

4.1 ICE Shared Memory Machine Model

Generally speaking, the energy consumption of a parallel algorithm is the sum of i) static energy (or leakage) E_{static}, ii) dynamic energy of computation E_{comp} and iii) dynamic energy of memory accesses E_{mem}. The static energy E_{static} is proportional to the execution time of the algorithm while the dynamic energy of computation and the dynamic energy of memory accesses are proportional to the number of computational operations and the number of memory accesses of the algorithm, respectively [47]. As a result, in the new ICE complexity model, the energy complexity of a multithreaded algorithm is analyzed based on its *span complexity* [21] (for the static energy), *work complexity* [21] (for the dynamic energy of computation) and *I/O complexity* (for the dynamic energy of memory accesses) (cf. Section 4.2). This section describes shared-memory machine models supporting I/O complexity analysis for parallel algorithms.

The first memory model we consider is parallel external memory (PEM) model [11], an extension of the Parallel Random Access Machine (PRAM) model that includes a two-level memory hierarchy. In the PEM model, there are n cores (or processors) each of which has its own *private* cache of size Z (in bytes) and shares the main memory with the other cores (cf. Figure 4.1). When n cores access n distinct blocks from the shared memory *simultaneously*, the I/O complexity in the PEM model is $O(1)$ instead of $O(n)$. Although the PEM model is appropriate for analyzing the I/O complexity of parallel algorithms in terms of time performance [11], we have found that the PEM model is not appropriate for analyzing parallel algorithms in terms of the dynamic energy of memory accesses. In fact, even when the n cores can access data from the main memory simultaneously, the *dynamic* energy consumption of the access is proportional to the number n of accessing cores (because of the load-store unit activated within each accessing core and the energy compositionality of parallel computations [31, 54]), rather than a constant as implied by the PEM model.

As a result, we consider the ideal distributed cache (IDC) model [27] to analyze I/O complexity of multithreaded algorithms in terms of dynamic energy consumption. Since the cache complexity of m misses is $O(m)$ regardless of whether or not the cache misses are incurred simultaneously by the

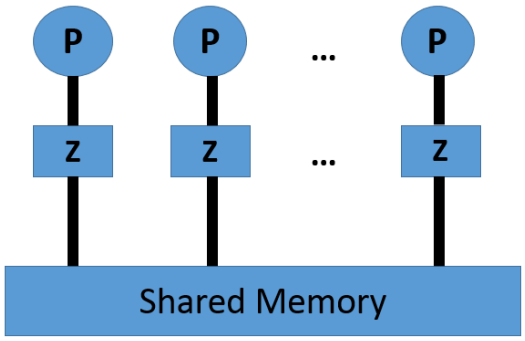


Figure 4.1: A Shared Memory Machine Model with Private Caches

cores, the IDC model reflects the aforementioned dynamic energy consumption of memory accesses by the cores.

However, the IDC model is mainly designed for analyzing the cache complexity of divide-and-conquer algorithms, making it difficult to apply to general multi-threaded algorithms targeted by our new ICE model. Constraining the new ICE model to the IDC model would limit the applicability of the ICE model to a wide range of multithreaded algorithms.

In order to make our new ICE complexity model applicable to a wide range of multithreaded algorithms, we show that the cache complexity analysis using the traditional (sequential) ideal cache (IC) model [26] can be used to find an upper bound on the cache complexity of the same algorithm using the IDC model (cf. Lemma 4.1.1). As the sequential execution of multithreaded algorithms is a valid execution regardless of whether they are divide-or-conquer algorithms, the ability to analyze the cache complexity of multithreaded algorithms via their sequential execution in the ICE complexity model improves the usability of the ICE model.

Let $Q_1(Alg, B, Z)$ and $Q_P(Alg, B, Z)$ be the cache complexity of a parallel algorithm Alg analyzed in the (uniprocessor) ideal cache (IC) model [26] with block size B and cache size Z (i.e, running Alg with a single core) and the cache complexity analyzed in the (multicore) IDC model with P cores each of which has a private cache of size Z and block size B, respectively. We have the following lemma:

Lemma 4.1.1. *The cache complexity $Q_P(Alg, B, Z)$ of a parallel algorithm Alg analyzed in the ideal distributed cache (IDC) model with P cores is bounded from above by the product of P and the cache complexity $Q_1(Alg, B, Z)$ of the same algorithm analyzed in the ideal cache (IC) model. Namely,*

$$Q_P(Alg, B, Z) \leq P * Q_1(Alg, B, Z) \tag{4.1}$$

Proof. (Sketch) Let $Q_P^i(Alg, B, Z)$ be the number of cache misses incurred by core i during the

parallel execution of algorithm Alg in the IDC model. Because caches do not interfere with each other in the IDC model, the number of cache misses incurred by core i when executing algorithm Alg in parallel by P cores is not greater than the number of cache misses incurred by core i when executing the whole algorithm Alg only by core i. That is,

$$Q_P^i(Alg, B, Z) \leq Q_1(Alg, B, Z) \tag{4.2}$$

or

$$\sum_{i=1}^{P} Q_P^i(Alg, B, Z) \leq P * Q_1(Alg, B, Z) \tag{4.3}$$

On the other hand, since the number of cache misses incurred by algorithm Alg when it is executed by P cores in the IDC model is the sum of the numbers of cache misses incurred by each core during the Alg execution, we have

$$Q_P(Alg, B, Z) = \sum_{i=1}^{P} Q_P^i(Alg, B, Z) \tag{4.4}$$

From Equations 4.3 and 4.4, we have

$$Q_P(Alg, B, Z) \leq P * Q_1(Alg, B, Z) \tag{4.5}$$

□

We also make the following assumptions regarding platforms.

- Algorithms are executed with the best configuration (e.g., maximum number of cores, maximum frequency) following the race-to-halt strategy.
- The I/O parallelism is bounded from above by the computation parallelism. Namely, each core can issue a memory request only if its previous memory requests have been served. Therefore, the work and span (i.e., critical path) of an algorithm represent the parallelism for both I/O and computation [21].

4.2 Energy Complexity in ICE model

This section describes two energy complexity models, a platform-supporting energy complexity model considering both platform and algorithm characteristics and a platform-independent energy complexity model considering only algorithm characteristics. The platform-supporting model is used when platform parameters in the model are available while the platform-independent model analyses energy complexity of algorithms without considering platform characteristics.

4.2.1 Platform-supporting Energy Complexity Model

This section describes a methodology to find energy complexity of algorithms. The energy complexity model considers three groups of parameters: machine-dependent, algorithm-dependent, and input-dependent parameters. The reason to consider all three parameter-categories is that only operational intensity [87] is insufficient to capture the characteristics of algorithms. Two algorithms with the same values of arithmetic intensity might consume different levels of energy. The reasons are their differences in data accessing patterns leading to performance scalability gap among them. For example, although the sequential version and parallel version of an algorithm may have the same arithmetic intensity, they may have different energy consumption since the parallel version would have less static energy consumption because of shorter execution time.

The energy consumption of a parallel algorithm is the sum of i) static energy (or leakage) E_{static}, ii) dynamic energy of computation E_{comp} and iii) dynamic energy of memory accesses E_{mem}: $E = E_{static} + E_{comp} + E_{mem}$ [19, 46, 47]. The static energy E_{static} is the product of the execution time of the algorithm and the static power of the whole platform. The dynamic energy of computation and the dynamic energy of memory accesses are proportional to the number of computational operations $Work$ and the number of memory accesses I/O, respectively. Pipelining technique in modern architectures enables overlapping computation with memory accesses [31]. Since computation time and memory-access time can be overlapped, the execution time of the algorithm is assumed to be the maximum of computation time and memory-access time [19]. Therefore, the energy consumption of algorithms is computed by Equation 4.6, where the values of ICE parameters, including ϵ_{op}, $\epsilon_{I/O}$, π_{op}, and $\pi_{I/O}$ are described in Table 4.1 and computed by the Equation 4.7, 4.8, 4.9, and 4.10, respectively.

$$E = P^{sta} \times max(T^{comp}, T^{mem}) + \epsilon_{op} \times Work + \epsilon_{I/O} \times I/O \tag{4.6}$$

$$\epsilon_{op} = P^{op} \times \frac{F}{Freq} \tag{4.7}$$

$$\epsilon_{I/O} = P^{I/O} \times \frac{M}{Freq} \tag{4.8}$$

$$\pi_{op} = P^{sta} \times \frac{F}{Freq} \tag{4.9}$$

$$\pi_{I/O} = P^{sta} \times \frac{M}{Freq} \tag{4.10}$$

The dynamic energy of one operation by one core ϵ_{op} is the product of the consumed power of one operation by one active core P^{op} and the time to perform one operation. Equation 4.7 shows how ϵ_{op} relates to frequency $Freq$ and the number of cycles per operation F. Similarly, the dynamic energy of a random access by one core $\epsilon_{I/O}$ is the product of the consumed power by one active core

Table 4.1: ICE Model Parameter Description

Machine	Description
ϵ_{op}	dynamic energy of one operation (average)
$\epsilon_{I/O}$	dynamic energy of a random memory access (1 core)
π_{op}	static energy when performing one operation
$\pi_{I/O}$	static energy of a random memory access
Algorithm	**Description**
$Work$	Number of work in flops of the algorithm [21]
$Span$	The critical path of the algorithm [21]
I/O	Number of cache line transfer of the algorithm [21]

performing one I/O (i.e., cache-line transfer) $P^{I/O}$ and the time to perform one cache line transfer computed as $M/Freq$, where M is the number of cycles per cache line transfer (cf. Equation 4.8). The static energy of operations π_{op} is the product of the whole platform static power P^{sta} and time per operation. The static energy of one I/O $\pi_{I/O}$ is the product of the whole platform static power and time per I/O, shown by Equation 4.9 and 4.10.

In order to compute *work*, *span* and *I/O* complexity of the algorithms, the input parameters also need to be considered. For example, SpMV algorithms consider input parameters listed in Table 4.3. Cache size is captured in the ICE model by the *I/O complexity* of the algorithm. Note that in the ICE machine model (Section 4.1), cache size Z is a constant and may disappear in the *I/O complexity* (e.g., O-notation).

The details of how to obtain the ICE parameters of recent platforms are discussed in Section 4.5.1. The actual values of ICE platform parameters for 11 recent platforms are presented in Table 4.2. The parameters of the first nine platforms are derived from [18] and the parameters of the two new platforms are found in this study.

The computation time of parallel algorithms is proportional to the span complexity of the algorithm, which is $T^{comp} = \frac{Span \times F}{Freq}$ where $Freq$ is the processor frequency, and F is the number of cycles per operation. The memory-access time of parallel algorithms in the ICE model is proportional to the I/O complexity of the algorithm divided by its I/O parallelism, which is $T^{mem} = \frac{I/O}{I/O-parallelism} \times \frac{M}{Freq}$. As I/O parallelism, which is the average number of I/O ports that the algorithm can utilize per step along the span, is bounded by the computation parallelism $\frac{Work}{Span}$, namely the average number of cores that the algorithm can utilize per step along the span (cf. Section 4.1), the memory-access time T^{mem} becomes: $T^{mem} = \frac{I/O \times Span \times M}{Work \times Freq}$ where M is the number of cycles per cache line transfer. If an algorithm has T^{comp} greater than T^{mem}, the algorithm is a CPU-bound algorithm. Otherwise, it is a memory-bound algorithm.

Table 4.2: Platform parameter summary.

Platform	Processor	ϵ_{op}(nJ)	π_{op}(nJ)	$\epsilon_{I/O}$(nJ)	$\pi_{I/O}$(nJ)
Nehalem i7-950	Intel i7-950	0.670	2.455	50.88	408.80
Ivy Bridge i3-3217U	Intel i3-3217U	0.024	0.591	26.75	58.99
Bobcat CPU	AMD E2-1800	0.199	3.980	27.84	387.47
Fermi GTX 580	NVIDIA GF100	0.213	0.622	32.83	45.66
Kepler GTX 680	NVIDIA GK104	0.263	0.452	27.97	26.90
Kepler GTX Titan	NVIDIA GK110	0.094	0.077	17.09	32.94
XeonPhi KNC	Intel 5110P	0.012	0.178	8.70	63.65
Cortex-A9	TI OMAP 4460	0.302	1.152	51.84	174.00
Arndale Cortex-A15	Samsung Exynos 5	0.275	1.385	24.70	89.34
Xeon	2xIntel E5-2650l v3	0.263	0.108	8.86	23.29
Xeon-Phi	Intel 31S1P	0.006	0.078	25.02	64.40

CPU-bound Algorithms

If an algorithm has computation time T^{comp} longer than data-accessing time T^{mem} (i.e., CPU-bound algorithms), the ICE energy complexity model becomes Equation 4.11 which is simplified as Equation 4.12.

$$E = P^{sta} \times \frac{Span \times F}{Freq} + \epsilon_{op} \times Work + \epsilon_{I/O} \times I/O \tag{4.11}$$

or

$$E = \pi_{op} \times Span + \epsilon_{op} \times Work + \epsilon_{I/O} \times I/O \tag{4.12}$$

Memory-bound Algorithms

If an algorithm has data-accessing time longer than computation time (i.e., memory-bound algorithms): $T^{mem} \geq T^{comp}$, energy complexity becomes Equation 4.13 which is simplified as Equation 4.14.

$$E = P^{sta} \times \frac{I/O \times Span \times M}{Work \times Freq} + \epsilon_{op} \times Work + \epsilon_{I/O} \times I/O \tag{4.13}$$

or

$$E = \pi_{I/O} \times \frac{I/O \times Span}{Work} + \epsilon_{op} \times Work + \epsilon_{I/O} \times I/O \tag{4.14}$$

Table 4.3: SpMV Input Parameter Description

SpMV Input	Description
n	Number of rows
nz	Number of nonzero elements
nr	Maximum number of nonzero in a row
nc	Maximum number of nonzero in a column
β	Size of a block

4.2.2 Platform-independent Energy Complexity Model

This section describes the energy-complexity model that is platform-independent and considers only algorithm characteristics. This complexity model is used when analyzing energy complexity of an algorithm without platform parameters. When the platform parameters (i.e., ϵ_{op}, $\epsilon_{I/O}$, π_{op}, and $\pi_{I/O}$) are unavailable, the energy-complexity model is derived from Equation 4.6 because the platform parameters are constants and can be removed. Assuming $\pi_{max} = max(\pi_{op}, \pi_{I/O})$, after removing platform parameters, the platform-independent energy complexity model is shown in Equation 4.15.

$$E = O(Work + I/O + max(Span, \frac{I/O \times Span}{Work})) \tag{4.15}$$

4.3 A Case Study of Sparse Matrix Multiplication

SpMV is one of the most common application kernels in Berkeley dwarf list [12]. It computes a vector result y by multiplying a sparse matrix A with a dense vector x: $y = Ax$. SpMV is a data-intensive kernel and has irregular memory-access patterns. The data access patterns for SpMV is defined by its sparse matrix format and matrix input types. There are several sparse matrix formats and SpMV algorithms in the literature. To name a few, they are Coordinate Format (COO), Compressed Sparse Column (CSC), Compressed Sparse Row (CSR), Compressed Sparse Block (CSB), Recursive Sparse Block (RSB), Block Compressed Sparse Row (BCSR) and so on. Three popular SpMV algorithms, namely CSC, CSB and CSR are chosen to validate the proposed energy complexity model. They have different data-accessing patterns leading to different values of I/O, work and span complexity. Since SpMV is a memory-bound application kernel, Equation 4.14 is applied. The input matrices of SpMV have different parameters listed in Table 4.3.

4.3.1 Compressed Sparse Row

CSR is a standard storage format for sparse matrices which reduces the storage of matrix compared to the tuple representation [48]. This format enables row-wise compression of A with size $n \times n$ (or

$n \times m$) to store only the non-zero nz elements. Let nz be the number of non-zero elements in matrix A. The *work* complexity of CSR SpMV is $\Theta(nz)$ where $nz >= n$ and *span* complexity is $O(nr + \log n)$ [16], where nr is the maximum number of non-zero elements in a row. The *I/O* complexity of CSR in the sequential I/O model of row-major layout is $O(nz)$ [14] namely, scanning all non-zero elements of matrix A costs $O(\frac{nz}{B})$ I/Os with B is the cache block size. However, randomly accessing vector x causes the total of $O(nz)$ I/Os. Applying the proposed model on CSR SpMV, their total energy complexity are computed as Equation 4.16.

$$E_{CSR} = O(\epsilon_{op} \times nz + \epsilon_{I/O} \times nz + \pi_{I/O} \times (nr + \log n)) \tag{4.16}$$

4.3.2 Compressed Sparse Column

CSC is the similar storage format for sparse matrices as CSR. However, it compresses the sparse matrix in column-wise manner to store the non-zero elements. The *work* complexity of CSC SpMV is $\Theta(nz)$ where $nz >= n$ and *span* complexity is $O(nc + \log n)$, where nc is the maximum number of non-zero elements in a column. The *I/O* complexity of CSC in the sequential I/O model of column-major layout is $O(nz)$ [14]. Similar to CSR, scanning all non-zero elements of matrix A in CSC format costs $O(\frac{nz}{B})$ I/Os. However, randomly updating vector y causing the bottleneck with total of $O(nz)$ I/Os. Applying the proposed model on CSC SpMV, their total energy complexity are computed as Equation 4.17.

$$E_{CSC} = O(\epsilon_{op} \times nz + \epsilon_{I/O} \times nz + \pi_{I/O} \times (nc + \log n)) \tag{4.17}$$

4.3.3 Compressed Sparse Block

Given a sparse matrix A, while CSR has good performance on SpMV $y = Ax$, CSC has good performance on transpose sparse matrix vector multiplication $y = A^T \times x$, Compressed sparse blocks (CSB) format is efficient for computing either Ax or $A^T x$. CSB is another storage format for representing sparse matrices by dividing the matrix A and vector x, y to blocks. A block-row contains multiple chunks, each chunks contains consecutive blocks and non-zero elements of each block are stored in Z-Morton-ordered [16]. From Beluc et al. [16], CSB SpMV computing a matrix with nz non-zero elements, size $n \times n$ and divided by block size $\beta \times \beta$ has span complexity $O(\beta \times \log \frac{n}{\beta} + \frac{n}{\beta})$ and *work* complexity as $\Theta(\frac{n^2}{\beta^2} + nz)$.

I/O complexity for CSB SpMV is not available in the literature. We do the analysis of CSB manually by following the master method [21]. The *I/O* complexity is analyzed for the algorithm CSB_SpMV(A,x,y) from Beluc et al. [16]. The I/O complexity of CSB is similar to *work* complexity of CSB $O(\frac{n^2}{\beta^2} + nz)$, only that non-zero accesses in a block is divided by B: $O(\frac{n^2}{\beta^2} + \frac{nz}{B})$, where B is cache block size. The reason is that non-zero elements in a block are stored in Z-Morton order which only requires $\frac{nz}{B}$ I/Os. The energy complexity of CSB SPMV is shown in Equation 4.18.

$$E_{CSB} = O(\epsilon_{op} \times (\frac{n^2}{\beta^2} + nz) + \epsilon_{I/O} \times (\frac{n^2}{\beta^2} + \frac{nz}{B}) + \pi_{I/O} \times \frac{(\frac{n^2}{\beta^2} + \frac{nz}{B}) \times (\beta \times \log \frac{n}{\beta} + \frac{n}{\beta})}{(\frac{n^2}{\beta^2} + nz)}) \quad (4.18)$$

Table 4.4: SpMV Complexity Analysis

Complexity	CSC-SpMV	CSB-SpMV	CSR-SpMV
Work	$\Theta(nz)$ [16]	$\Theta(\frac{n^2}{\beta^2} + nz)$ [16]	$\Theta(nz)$ [16]
I/O	$O(nz)$ [14]	$O(\frac{n^2}{\beta^2} + \frac{nz}{B})$ [this study]	$O(nz)$ [14]
Span	$O(nc + \log n)$ [16]	$O(\beta \times \log \frac{n}{\beta} + \frac{n}{\beta})$ [16]	$O(nr + \log n)$ [16]

From the complexity analysis of SpMV algorithms using different layouts, the complexity of CSR-SpMV, CSC-SpMV and CSB-SpMV are summarized in Table 4.4.

4.4 A Case Study of Dense Matrix Multiplication

Besides SpMV, we also apply the ICE model to dense matrix multiplication (matmul). Unlike SpMV, a data-intensive kernel, matmul is a computation-intensive kernel used in high-performance computing. It computes output matrix C (size n x p) by multiplying two dense matrices A (size n x m) and B (size m x p): $C = A \times B$. In this work, we implemented two matmul algorithms (i.e., a basic algorithm and a cache-oblivious algorithm [26]) and apply the ICE analysis to find their energy complexity. Both algorithms partition matrix A and C equally to N sub-matrices (e.g., A_i with i=(1, 2, .., N)), where N is the number of cores in the platform. The partition approach is shown in Figure 4.2. Each sub-matrix A_i has size $\frac{n}{N} \times m$ and each sub-matrix C_i has size $\frac{n}{N} \times p$. Each core computes a sub-matrix C_i: $C_i = A_i \times B$. Since matmul is a computation-bound application kernel, Equation 4.12 is applied.

4.4.1 Basic Matmul Algorithm

The basic matmul algorithm is described in Figure 4.3. Its work complexity is $\Theta(2nmp)$ [89] and span complexity is $\Theta(\frac{2nmp}{N})$ because the computational work is divided equally to N cores due to matrix partition approach. When matrix size of matrix B is bigger than platform cache size, the basic algorithm loads matrix B n times (i.e., once for computing each row of C), results in $\frac{nmp}{B}$ cache block transfer, where B is cache block size. In total, I/O complexity of the basic matmul algorithm is $\Theta(\frac{nm+nmp+np}{B})$. By applying the ICE model on this algorithm, the total energy complexity is computed as Equation 4.19.

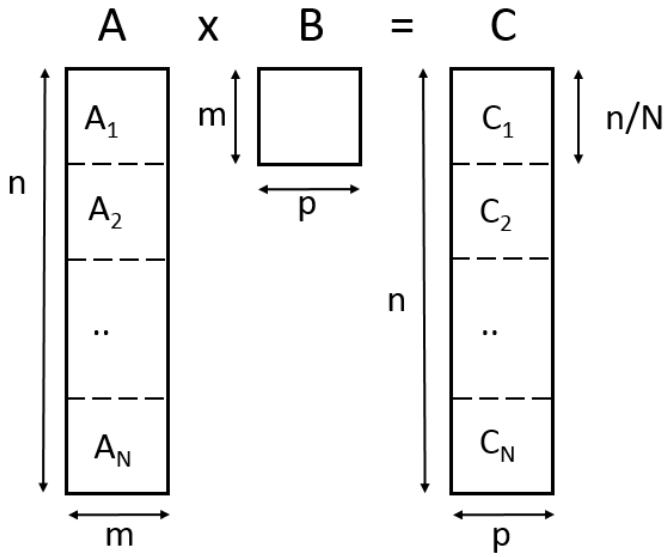


Figure 4.2: Partition approach for parallel matmul algorithms.

```
1  for i = 1 to n
2      for j = 1 to p
3          for k = 1 to m
4              C(i,j) = C(i,j) + A(i,k) * B(k,j)
```

Figure 4.3: Basic matmul algorithm, where sizes of matrix A, B, C are nxm, mxp, nxp, respectively.

$$E_{CO} = O(\epsilon_{op} \times 2nmp + \epsilon_{I/O} \times (n + m + p + \frac{nm + mp + np}{B} + \frac{nmp}{B\sqrt[2]{Z}}) + \pi_{op} \times \frac{2nmp}{N}) \quad (4.20)$$

$$E_{basic} = O(\epsilon_{op} \times 2nmp + \epsilon_{I/O} \times \frac{nm + nmp + np}{B} + \pi_{op} \times \frac{2nmp}{N}) \quad (4.19)$$

4.4.2 Cache-oblivious Matmul Algorithm

The cache-oblivious matmul (CO-matmul) algorithm [26] is a divide-and-conquer algorithm. It has work complexity the same as the basic matmul algorithm $\Theta(2nmp)$. Its span complexity is also $\Theta(\frac{2nmp}{N})$ because of the used matrix partition approach shown in Figure 4.2. The I/O complexity of CO-matmul, however, is different from the basic algorithm: $\Theta(n + m + p + \frac{nm+mp+np}{B} + \frac{nmp}{B\sqrt[2]{Z}})$ [26]. Applying the ICE model to CO-matmul, the total energy complexity is computed as Equation 4.20.

Table 4.5: Matmul Complexity Analysis

Complexity	Cache-oblivious Algorithm	Basic Algorithm
Work	$\Theta(2nmp)$ [26]	$\Theta(2nmp)$ [89]
I/O	$\Theta(n+m+p+\frac{nm+mp+np}{B}+\frac{nmp}{B\sqrt[2]{Z}})$ [26]	$\Theta(\frac{nm+nmp+np}{B})$ [this study]
Span	$\Theta(\frac{2nmp}{N})$ [this study]	$\Theta(\frac{2nmp}{N})$ [this study]

4.5 Validation of ICE Model

This section describes the experimental study to validate the ICE model, including: introducing the two experimental platforms and how to obtain their parameters for the ICE model (cf. Section 4.5.1), describing SpMV implementation and sparse matrix types used in this validation (cf. Section 4.5.3), and discussing the validation results of SpMV and matmul.

4.5.1 Experiment Set-up

For the validation of the ICE model, we conduct the experiments on two HPC platforms: one platform with two Intel Xeon E5-2650l v3 processors and one platform with Xeon Phi 31S1P processor. The Intel Xeon platform has two processors Xeon E5-2650l v3 with 2×12 cores; each processor has the frequency 1.8 GHz. The Intel Xeon Phi platform has one processor Xeon Phi 31S1P with 57 cores and its frequency is 1.1 GHz. To measure energy consumption of the platforms, we read the PCM MSR counters for Intel Xeon and MIC power reader for Xeon Phi.

4.5.2 Identifying Platform Parameters

the ICE parameter values are derived from the parameters of the roofline model [19, 18]. The energy roofline studies have provided a list of different platforms including CPU, GPU, embedded platforms with their parameters considered in the Roofline model. However, the parameter values of the two new HPC platforms (i.e., Intel Xeon E5-2650l v3 and Xeon-Phi 31S1P) used to validate the ICE model are not available from energy roofline studies [19, 18]. Therefore, we apply the energy roofline approach [19] to find the platform parameters for the two new experimental platforms.

Firstly, we create micro-benchmarks for the two platforms and measure their energy consumption and performance. The energy roofline studies provide the source codes of micro-benchmarks that they used to obtained time and energy consumption data of their considered platforms. We create micro-benchmarks for Intel Xeon E5-2650l v3 based on the Nehalem i7-950 code and micro-benchmarks for Xeon Phi 31S1P based on the code of Xeon Phi 5110P. The micro-benchmarks have

different combinations of a number of operations W and a number of memory accesses Q. Then we measure time and energy consumption of each micro-benchmark using Intel Performance Counter Monitor.

In the roofline model, the total energy consumption and time performance of an application are calculated as in Equation 4.21 and 4.22. There are 4 known variables such as W, Q, E and T where W is the total of operations, Q is a total of memory accesses, E is the total energy consumption and T is the time performance. There are also 6 unknown parameters such as ϵ_d, ϵ_{mem}, τ_d, τ_{mem}, π_1 and δ_π where π_1 is the static power and δ_π is the additional units of usable power to perform any operations. By using lsucurvefit function of Matlab, we identify the unknown parameters which give the best fit of the roofline model to input (i.e., W, Q) and output data (i.e., execution time and energy consumption).

$$E = \pi_1 \times max(W \times \tau_d, Q \times \tau_{mem}, \frac{W \times \epsilon_d + Q \times \epsilon_{mem}}{\delta_\pi}) + \epsilon_d \times Work + \epsilon_{mem} \times Q \tag{4.21}$$

$$T = max(W \times \tau_d, Q \times \tau_{mem}, \frac{W \times \epsilon_d + Q \times \epsilon_{mem}}{\delta_\pi}) \tag{4.22}$$

Then, the ICE parameter values are derived from the roofline parameters. Thanks to authors Choi et al. [18], we extract the required values of ICE parameters for the nine platforms presented in their study as follows: $\epsilon_{op} = \epsilon_d$, $\epsilon_{I/O} = \epsilon_{mem} \times B$, $\pi_{op} = \pi_1 \times \tau_d$, $\pi_{I/O} = \pi_1 \times \tau_{mem}$, where B is cache block size, ϵ_d, ϵ_{mem}, τ_d, τ_{mem} are defined by [18] as energy per flop, energy per byte, time per flop and time per byte, respectively. The values of ICE platform parameters are listed in Table 4.2. Along with the two HPC platforms (i.e., Intel Xeon E5-2650l v3 and Xeon Phi 31S1P) used in this validation, we provide parameters required in the ICE model for a total of 11 platforms in Table 4.2.

4.5.3 SpMV Implementation

We want to conduct complexity analysis and experimental study with two SpMV algorithms, namely CSB and CSC. Parallel CSB and sequential CSC implementations are available thanks to the study by Buluç et al. [16]. Since the optimization steps of available parallel SpMV kernels (e.g., pOSKI [2], LAMA[25]) might affect the work complexity of the algorithms, we decided to implement a simple parallel CSC using Cilk and pthread. To validate the correctness of our parallel CSC implementation, we compare the vector result y from $y = A * x$ of CSC and CSB implementation. The comparison shows the equality of the two vector results y. Moreover, we compare the performance of our parallel CSC code with Matlab parallel CSC-SpMV kernel. Matlab also uses CSC layout as the format for their sparse matrix [28] and Matlab is used as the baseline comparison for SpMV studies [16]. Our CSC implementation has out-performed Matlab parallel CSC kernel when computing the

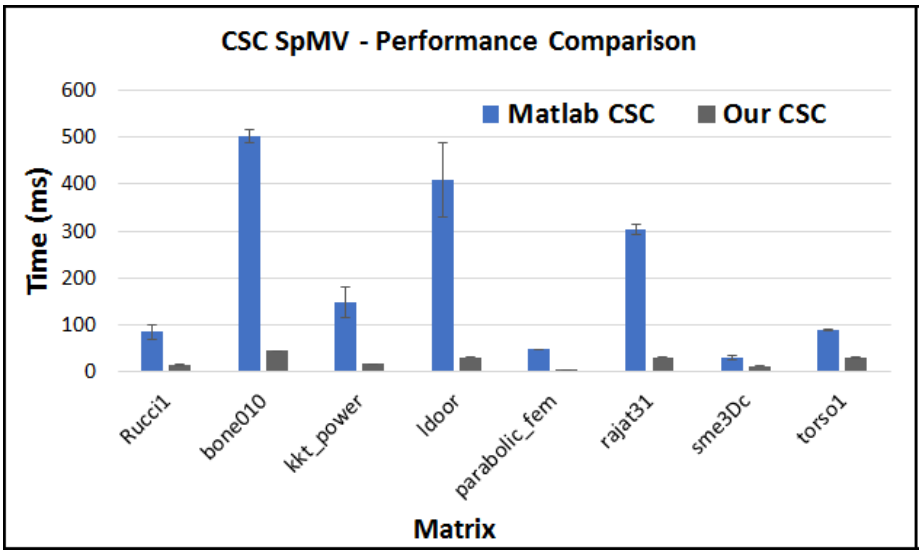


Figure 4.4: Performance (time) comparison of two parallel CSC SpMV implementations. For a set of different input matrices, the parallel CSC SpMV using Cilk out-performs Matlab parallel CSC.

same targeted input matrices at least 136% across different inputs from Table 4.6. The experimental study of SpMV energy consumption is then conducted with CSB SpMV implementation from Buluç et al. [16] and our CSC SpMV parallel implementation.

4.5.4 SpMV Matrix Input Types

We conducted the experiments with nine different matrix-input types from Florida sparse matrix collection [23]. Each matrix input has different properties listed in Table 4.3, including size of the matrix $n \times m$, the maximum number of non-zero of the sparse matrix nz, the maximum number of non-zero elements in one column nc. Table 4.6 lists the matrix types used in this experimental validation with their properties.

4.5.5 Validating ICE Using Different SpMV Algorithms

The model aims to compare the energy consumption of two algorithms. Therefore, we validate the ICE model by showing the comparison using the ratio of energy consumption of two algorithms. From the model-estimated data, CSB SpMV consumes less energy than CSC SpMV on both platforms. Even though CSB has higher work complexity than CSC, CSB SpMV has less I/O complexity than CSC SpMV. Firstly, the dynamic energy cost of one I/O is much higher than the energy cost of one operation (i.e., $\epsilon_{I/O} >> \epsilon_{op}$) on both platforms. Secondly, CSB has better parallelism than CSC, computed by $\frac{Work}{Span}$, which results in a shorter execution time. Both reasons contribute to the less energy consumption of CSB SpMV. The measurement data confirms that CSB SpMV algorithm

Table 4.6: Sparse matrix input types. The maximum number of non-zero elements in a column *nc* is derived from [16].

Matrix type	**n**	**m**	**nz**	**nc**
bone010	986703	986703	47851783	63
kkt_power	2063494	2063494	12771361	90
ldoor	952203	952203	42493817	77
parabolic_fem	525825	525825	3674625	7
pds-100	156243	517577	1096002	7
rajat31	4690002	4690002	20316253	1200
Rucci1	1977885	109900	7791168	108
sme3Dc	42930	42930	3148656	405
torso1	116158	116158	8516500	1200

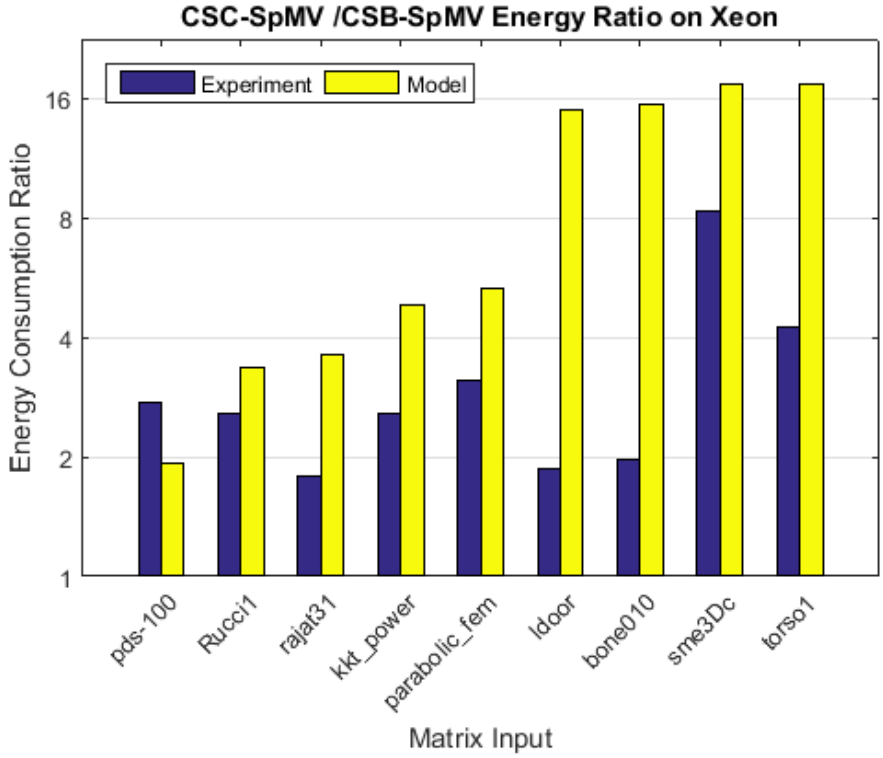


Figure 4.5: Energy consumption comparison between CSC-SpMV and CSB-SpMV on the Intel Xeon platform, computed by $\frac{E_{CSC}}{E_{CSB}}$.

consumes less energy than CSC SpMV algorithm, shown by the energy consumption ratio between CSC-SpMV and CSB-SpMV greater than 1 in the Figure 4.5 and 4.6. Both the ICE model estimation

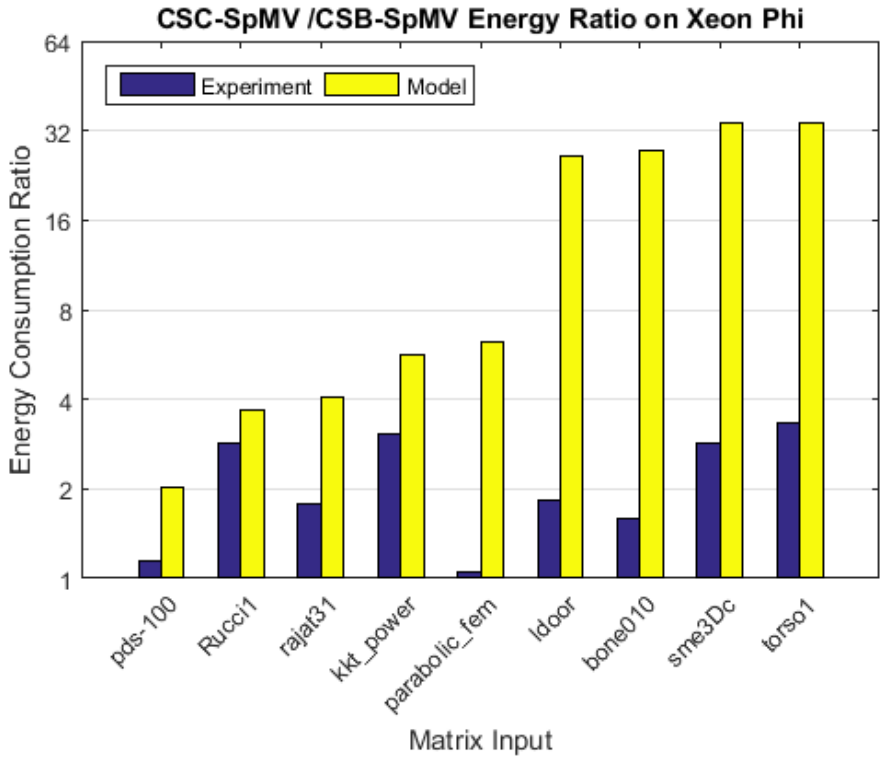


Figure 4.6: Energy consumption comparison between CSC-SpMV and CSB-SpMV on the Intel Xeon Phi platform, computed by $\frac{E_{CSC}}{E_{CSB}}$.

and experimental measurement on Intel Xeon and Xeon Phi platform show the consistent results that $\frac{E_{CSC}}{E_{CSB}}$ is greater than 1, meaning CSC SpMV algorithm consumes more energy than the CSB SpMV algorithm on different input matrices. For all input matrices, the ICE model has confirmed that CSB SpMV consumes less energy than CSC SpMV algorithm.

Because the model has abstracted possible platform by only 4 parameters (i.e., ϵ_{op}, $\epsilon_{I/O}$, π_{op}, and $\pi_{I/O}$), there are the differences between the model and experiment ratios shown in the Figure 4.5 and 4.6. Moreover, the *work, span*, and *I/O* complexities are the functions of these input sparse matrix parameters and so does the SpMV energy complexity. The sparse matrices used in the experiments have different patterns with different values of matrix parameters (i.e., n, m, nz and nc) defined in Table 4.3. Therefore, it is not expected that the ratio $\frac{E_{CSC}}{E_{CSB}}$ from the model follows the same increasing/decreasing order among input matrices as the ratio from experiments. For accurate models that provide the precise energy estimation, the platform parameters need to be highly detailed such as RTHpower model for embedded platforms [80, 77].

Table 4.7: Comparison of Energy Consumption of Different Matrix Input Types.

Algorithm	CSB	CSB	CSC	CSC	CSB	CSB	CSC	CSC
Platform	Xeon	Xeon	Xeon	Xeon	Xeon-Phi	Xeon-Phi	Xeon-Phi	Xeon-Phi
Model/Exprmt	model	exprmt	model	exprmt	model	exprmt	model	exprmt
Energy Consumption Increasing Order	sme3Dc	pds-100	pds-100	pds-100	sme3Dc	pds-100	pds-100	parabolic
	torso1	parabolic	sme3Dc	parabolic	torso1	parabolic	sme3Dc	pds-100
	pds-100	sme3Dc	parabolic	sme3Dc	pds-100	Rucci1	parabolic	Rucci1
	parabolic	Rucci1	Rucci1	Rucci1	parabolic	sme3Dc	Rucci1	sme3Dc
	Rucci1	kkt	torso1	kkt	ldoor	kktr	torso1	rajat31
	kkt	torso1	kkt	torso1	bone010	torso1	kkt	kkt
	ldoor	rajat31	rajat31	rajat31	Rucci1	rajat31	rajat31	ldoor
	bone010	ldoor	ldoor	ldoor	kkt	ldoor	ldoor	torso1
	rajat31	bone010	bone010	bone010	rajat31	bone010	bone010	bone010

Table 4.8: CSC Energy Comparison of Different Input Matrix Types on Xeon

Correctness	pds-100	parabolic	sme3Dc	Rucci1	kkt	torso1	rajat31	ldoor	bone010
pds-100	x	1	1	1	1	1	1	1	1
parabolic		x	0	1	1	1	1	1	1
sme3Dc			x	1	1	1	1	1	1
Rucci1				x	1	1	1	1	1
kkt					x	0	1	1	1
torso1						x	1	1	1
rajat31							x	1	1
ldoor								x	1
bone010									x

4.5.6 Validating ICE Using Different Input Types

To validate the ICE model regarding input types, the experiments have been conducted with nine matrix types listed in Table 4.6. The model can capture the energy-consumption relation among different inputs. The increasing order of energy consumption of different matrix-input types is shown in Table 4.7, from both model estimation and experimental study.

For instance, in order to validate the comparison of energy consumption for different input types, a validated table as Table 4.8 is created for CSC SpMV on Xeon to compare model prediction and experimental measurement. For nine input types, there are $\frac{9\times9}{2} - 9 = 36$ input relations. If the relation is correct, meaning both experimental data and model data are the same, the relation value in the table of two inputs is 1. Otherwise, the relation value is 0. From Table 4.8, there are 34 out of 36 relations are the same for both model and experiment, which gives 94% accuracy on the relation of the energy consumption of different inputs. Similarly, the input validation for CSC and CSB on both Xeon and Xeon Phi platforms is provided in Table 4.9.

Table 4.9: Comparison accuracy of SpMV energy consumption computing different input matrix types

Algorithm	CSB	CSC
Xeon	75%	94%
Xeon Phi	63.8%	80.5%

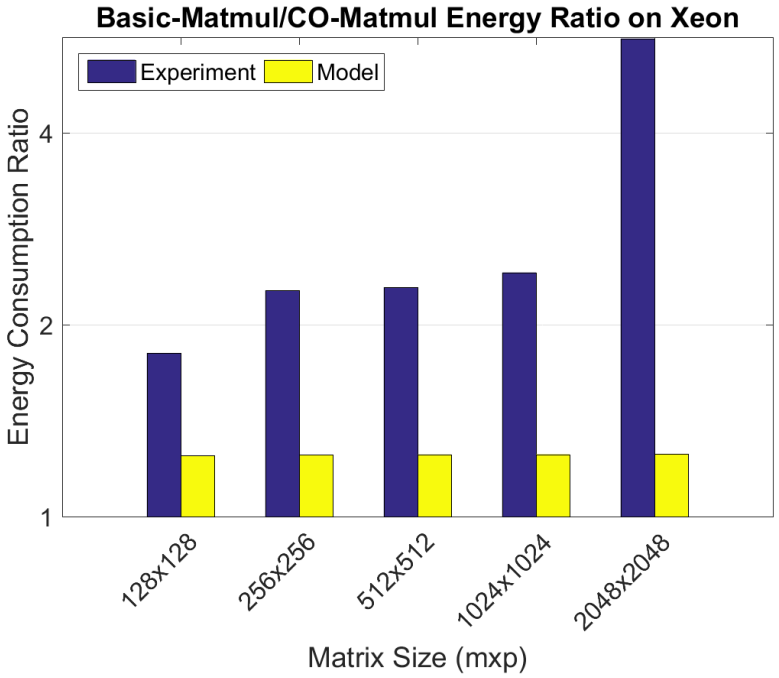


Figure 4.7: Energy consumption comparison between Basic-Matmul and CO-Matmul on the Intel Xeon platform, computed by $\frac{E_{Basic}}{E_{CO}}$.

4.5.7 Validating ICE With Matmul Algorithms

From the model-estimated data, Basic-Matmul consumes more energy than CO-Matmul on both platforms. Even though both algorithms have the same work and span complexity, Basic-Matmul has more I/O complexity than CO-Matmul, which results in higher energy consumption of Basic-Matmul compared to CO-Matmul algorithm. The measurement data confirms that Basic-Matmul algorithm consumes more energy than CO-Matmul algorithm, shown by the energy consumption ratio between Basic-Matmul and CO-Matmul greater than 1 in Figure 4.7 and 4.8. For all input matrices, the ICE model has confirmed that Basic-Matmul consumes more energy than CO-Matmul algorithm.

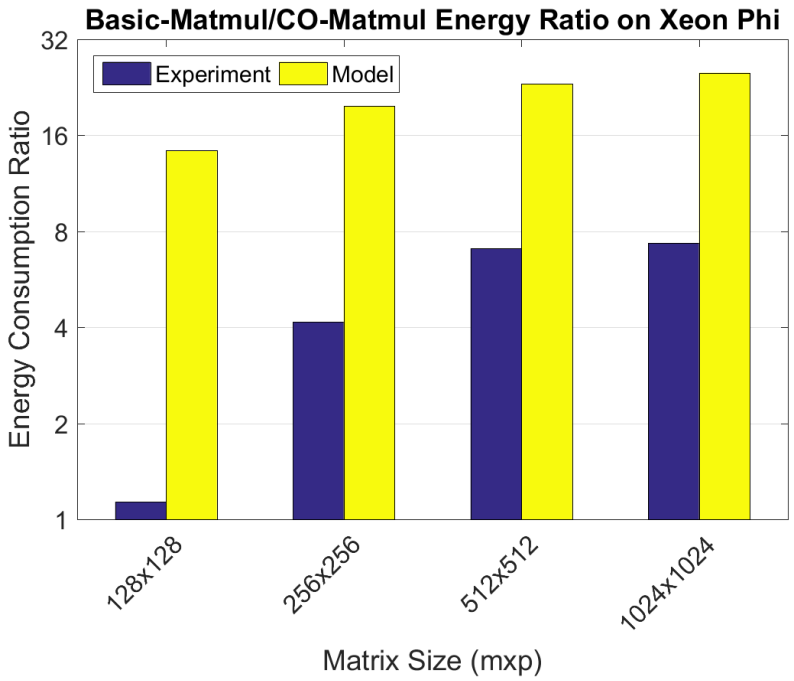


Figure 4.8: Energy consumption comparison between Basic-Matmul and CO-Matmul on the Intel Xeon Phi platform, computed by $\frac{E_{Basic}}{E_{CO}}$.

There are the differences between the model and experiment ratios shown in Figure 4.5 and 4.6 because the model has only 4 parameters (i.e., ϵ_{op}, $\epsilon_{I/O}$, π_{op}, and $\pi_{I/O}$) to represent the platform. However, the input matrices used in matmul experiments have increasing order of matrix sizes where the matrix dimension m or p ($m = p$) is proportional to n with a constant (i.e., the platform maximum number of cores). The matmul *work, span, I/O* and *energy* complexity become the complexity functions of one variable (i.e., n or m or p). Therefore, the ratios of $\frac{E_{Basic}}{E_{CO}}$ from the model captures the consistent order across increasing matrix sizes with the ratio in the experiments.

4.6 Applying the ICE Models to Exascale Systems

In the future, supercomputers are expected to be capable to perform at exaflops level such as a billion billion operations per second. To build such systems and achieve that huge computational power, there are also several challenges in four primary directions, namely energy and power, memory and storage, concurrency and locality as well as resiliency [15].

The first step to tackle the energy challenge of exascale systems is to understand their energy consumption. The ICE models proposed in this study enable energy complexity analysis of parallel

Table 4.10: Platform parameters of the exaflops system [15]

ϵ_{op}(nJ)	π_{op}(nJ)	$\epsilon_{I/O}$(nJ)	$\pi_{I/O}$(nJ)
0.0106	2899000	1.536	144950000

algorithms on theoretical exaflops systems. In the section, we demonstrate how to apply the ICE model to the exaflops system and predict energy consumption of two matmul algorithms: cache-oblivious and basic matmul.

From the platform features of the exaflops system provided in Table 7.10 of [15], we compute the platform parameters for the ICE models such as ϵ_{op}, $\epsilon_{I/O}$, π_{op}, $\pi_{I/O}$ from Equation 4.7, 4.8, 4.9, 4.10. The parameter values of the exaflops system are shown in Table 4.10.

The ICE model can predict energy ratio trend of two matmul algorithms when running on the exascale system as shown in Figure 4.11 over different sizes of input matrices. The cache-oblivious algorithm is more energy-efficient than the basic matmul algorithm. The ICE models can also predict which energy portion of the three energy components (i.e., static energy, dynamic energy of computation and dynamic energy of memory access) is more significant when running the algorithm. The three energy components of both basic and cache-oblivious algorithm are shown in Figure 4.10 and 4.9.

From the estimation of the basic matmul algorithm shown in Figure 4.10, the memory accesses (i.e., I/O) consumes the largest part of energy with almost 80% of the total energy consumption over various input matrix sizes. From the estimation of cache-oblivious algorithm shown in Figure 4.9, cache-oblivious matmul has more efficient mechanism for accessing memory and requires less cache line transfers, resulting in less energy consumption of memory accesses. As their Work and Span complexities are the same (cf. Table 4.5), their E_{static} and E_{comp} (E_{static} and E_{dynW}, respectively in Figure 4.10 and 4.9) are the same. That means 80% energy consumption of basic matmul related to I/O in Figure 4.10 (the green parts), has almost been eliminated in cache-oblivious matmul in Figure 4.9 with large matrix sizes. Knowing which algorithm component (i.e., Work, Span, I/O) contributes the large part to the total energy consumption also gives hints to algorithm designers where to improve the algorithms in order to save energy.

4.7 Related Work - Overview of energy models

Energy models for finding energy-optimized system configurations for a given application have been recently reported [12, 16, 19]. Imes et al. [39] used controller theory and linear programming to find energy-optimized configurations for an application with soft real-time constraints at runtime. Mishra

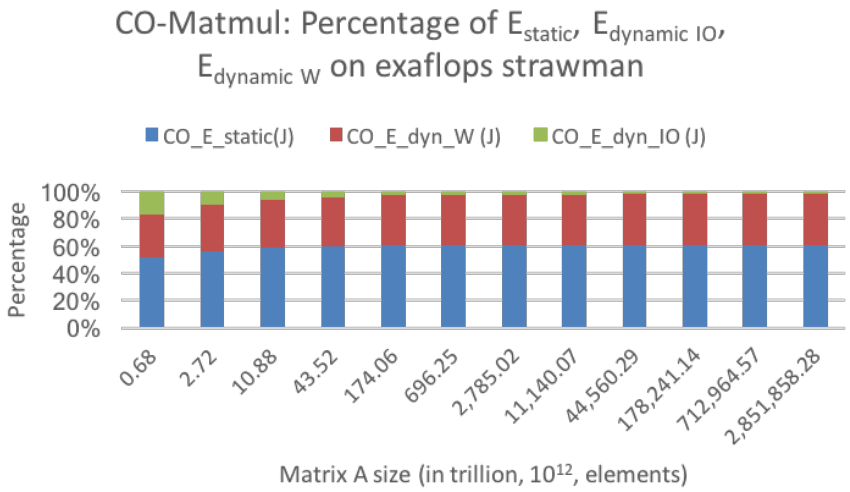


Figure 4.9: Energy percentage of Cache-oblivious Matmul

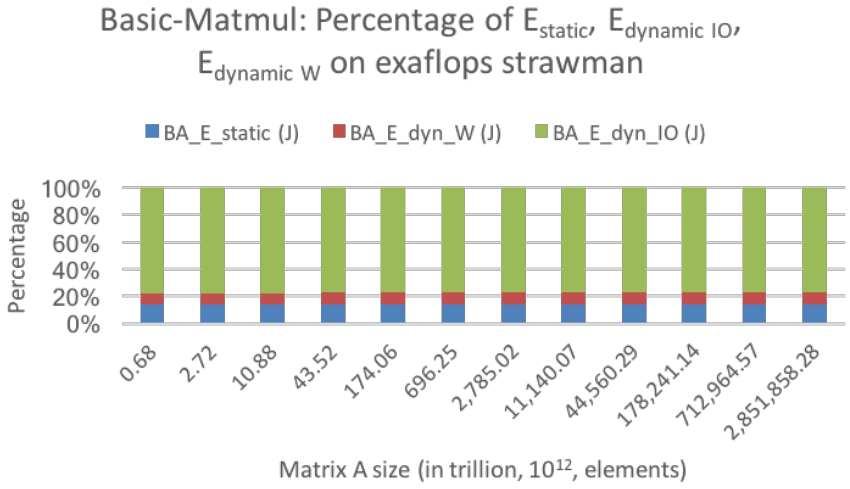


Figure 4.10: Energy percentage of Basic Matmul

et al. [63] used the hierarchical Bayesian model in machine learning to find energy-optimized configurations. Snowdon et al. [73] developed a power management framework called Koala which models the energy consumption of the platform and monitors an application' energy behavior. Although the energy models for finding energy-optimized system configurations have resulted in energy saving in practice, they focus on characterizing system platforms rather than applications and therefore are not appropriate for analyzing the energy complexity of application algorithms.

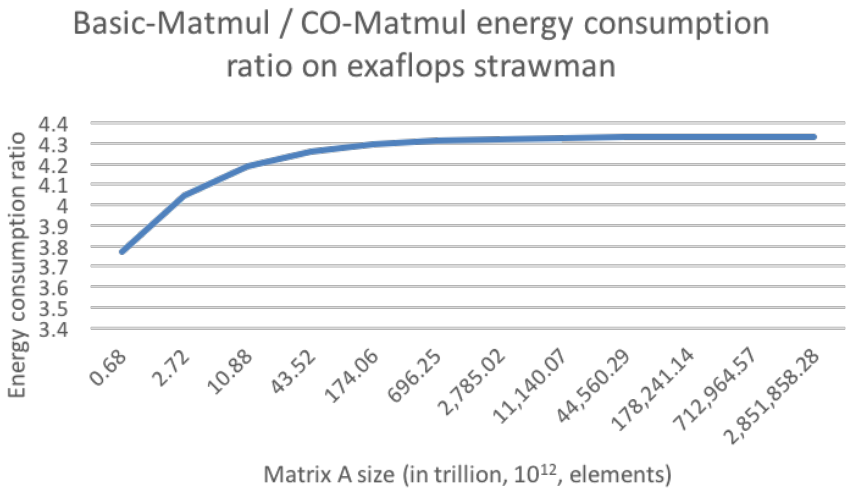


Figure 4.11: Energy ratio of Basic-Matmul to CO-Matmul running on the exascale system

Another direction of energy modeling study is to predict the energy consumption of applications by analyzing applications without actual execution on real platforms which we classify as analytic models.

Energy roofline models [19, 18] are some of the comprehensive energy models that abstract away possible algorithms in order to analyze and characterize different multicore platforms in terms of energy consumption. Our new energy model, which abstracts away possible multicore platform and characterize the energy complexity of algorithms based on their *work, span*, and *I/O* complexity, complements the energy roofline models.

Validated energy models for *specific* algorithms have been reported recently [10, 62]. Alonso et al. [10] provided an accurate energy model for three key dense matrix factorizations. Malossi et al. [62] focused on basic linear-algebra kernels and characterized the kernels by the number of arithmetic operations, memory accesses, reduction and barrier steps. Although the energy models for specific algorithms are accurate for the target algorithms, they are not applicable for other algorithms and therefore cannot be used as general energy complexity models for parallel algorithms.

The *energy scalability* of a parallel algorithm has been investigated by Korthikanti et al. [46, 47]. Unlike the energy scalability studies that have not been validated on real platforms, our new energy complexity model is validated on HPC and accelerator platforms, confirming its usability and accuracy.

The energy complexity of *sequential* algorithms on a *uniprocessor* machine with *several memory banks* has been studied by Roy et al. [70]. Our energy complexity studies complement Roy et al.'s studies by investigating the energy complexity of *parallel* algorithms on a *multiprocessor* machine

with *a shared memory bank* and private caches, a machine model that has been widely adopted to study parallel algorithms [27, 11, 47].

4.8 Conclusion

In this study, we have devised a new general model for analyzing the energy complexity of multithreaded algorithms. The energy complexity of an algorithm is derived from its *work*, *span* and *I/O* complexity. Moreover, two case studies are conducted to demonstrate how to use the model to analyze the energy complexity of SpMV algorithms and matmul algorithms. The energy complexity analyses are validated for two SpMV algorithms and two matmul algorithms on two HPC platforms with different input matrices. The experimental results confirm the theoretical analysis in terms of which algorithm consumes more energy. The ICE energy complexity model gives algorithm-developers the insight into which algorithm is analytically more energy-efficient.

The energy complexity models require finding platforms parameters which are more relevant to homogeneous systems. In order to model the energy consumption of applications running on heterogeneous systems, we devise a runtime energy optimization in the next chapter to estimate the energy consumption of applications based on probabilistic network techniques. From the energy estimation, we also develop a framework to identify the system configuration to run the application for achieving the most energy efficiency during runtime.

Chapter 5

REOH: Using Probabilistic Network for Runtime Energy Optimization of Heterogeneous Systems

Significant efforts have been devoted to choosing the best configuration of a computing system to run an application energy efficiently. However, available tuning approaches mainly focus on homogeneous systems and are inextensible for heterogeneous systems which include several components (e.g., CPUs, GPUs) with different architectures. Section 1.1.3 summarizes the existing auto-tuning approaches to find the most energy-efficient configurations of computing systems for a given application.

This chapter of the thesis answers the Research Question 3 introduced in Chapter 1, Section 1.1.3: *"RQ3: How to identify the most energy-efficient system configurations (i.e., platform and its setting) of a heterogeneous system containing platforms with different architectures to run the application?"*.

This study proposes a holistic tuning approach called REOH using a probabilistic modeling approach to predict the most energy-efficient configuration of a *heterogeneous* system for running a given application. Based on the computation and communication patterns from Berkeley dwarfs, we conduct experiments to devise the training set including 7074 data samples covering varying application patterns and characteristics. Validating the REOH approach on heterogeneous systems including CPUs and GPUs shows that the energy consumption by the REOH approach is close to the

optimal energy consumption by the Brute Force approach while saving 17% of sampling runs compared to the previous (homogeneous) approach using probabilistic networks. Based on the REOH approach, we develop an open-source energy-optimizing runtime framework for selecting an energy efficient configuration of a heterogeneous system for a given application at runtime. This chapter is organized as follows. Section 5.1 explains the mathematical background behind the Probabilistic Graphical Model (PGM). Section 5.2 describes REOH, the energy optimization approach for heterogeneous systems. In Section 5.3, we validate the approach on a heterogeneous system consisting of a CPU and a GPU. Based on the proposed energy optimization approach, Section 5.4 describes the energy-optimizing runtime framework and its implementation. The related work of this study is summarized in Section 5.5. Section 5.6 concludes the chapter.

5.1 Background

This section first explains the basic of probabilistic graphical models, and then how to use probabilistic graphical model-based approach for runtime energy optimization.

5.1.1 Probabilistic Graphical Model

Probabilistic Graphical Models (PGMs) use graphs to represent probability distributions. A graph denoted by G= (X, E) consists of a set of vertices (or nodes) X and a set of edges E. In PGMs, each node is a random variable. The edges are the relationships between the random variables. A connection between two nodes Xi and Xj in a graph is either a directed or an undirected edge.

The graphs are categorized into three types based on the types of edges: directed, undirected and mixed graph. A graph is directed or undirected if all the connections are directed or undirected. A graph containing both undirected and directed edges is a mixed graph [68]. An example of each graph type is given in Figure 5.1. There are also three types of PGMs: Bayesian networks, Markov networks and factor graphs which captures different properties of probabilistic models. The PGM used in this chapter is a directed Bayesian network. Bayesian networks (BN) are directed graphical models where its structure is the same over time. BN has a set of conditional independence assumptions described below.

Serial connection

A serial connection as shown in Figure 5.2 has a joint probability of three variables X_i, X_k, X_j as $P_B(X_i, X_k, X_j) = P(X_i)P(X_k|X_i)P(X_j|X_k)$ [68]. A join probability of a set of random variables is the probability where all the variables in the set are observed.

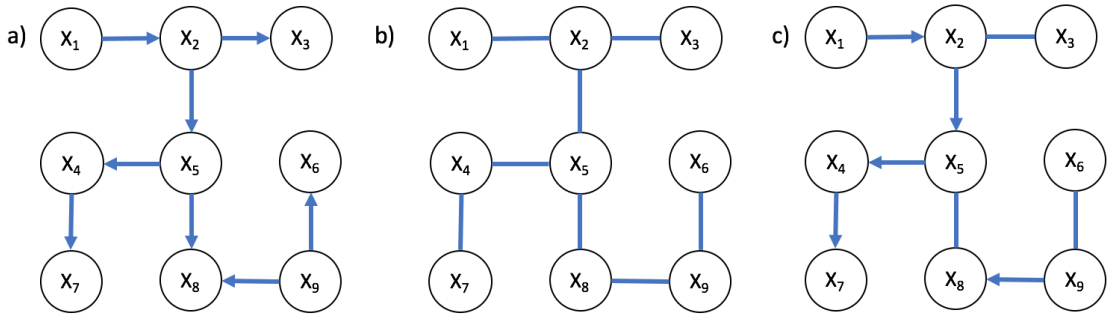


Figure 5.1: Graph types: a) Directed graph b) Undirected graph c) Mixed graph

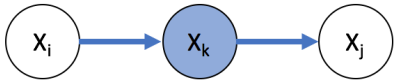


Figure 5.2: Serial Connection

Diverging connection

A diverging connection as shown in Figure 5.3 has a joint probability of three variables X_i, X_k, X_j as $P_B(X_i, X_k, X_j) = P(X_k)P(X_i|X_k)P(X_j|X_k)$ [68].

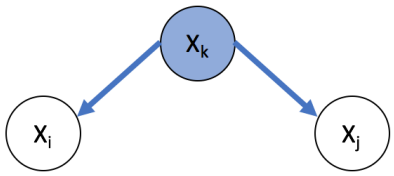


Figure 5.3: Diverging Connection

Converging connection

A converging connection as shown in Figure 5.4 has a joint probability of three variables X_i, X_k, X_j as $P_B(X_i, X_k, X_j) = P(X_i)P(X_j)P(X_k|X_i, X_j)$ [68].

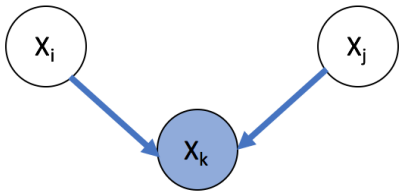


Figure 5.4: Converging Connection

5.1.2 Using Probabilistic Network Approach for Runtime Energy Optimization

The probabilistic graphical model-based approach for homogeneous systems [63] is to predict the energy consumption of all configurations from the offline training data and the online sampling data. The offline training data is the observed data from all previously observed applications. The offline training data is obtained by measuring both power and execution time of previously observed applications running on all system configurations. The online sampling data is the partially observed data of a target application. The online sampling data is partially observed because it is obtained by measuring both power and execution time of a target application running on a small set of system configurations (i.e., sample configurations) randomly selected and not all system configurations. In order to estimate power and execution time of the target application running on the remaining system configurations, we apply the probabilistic graphical model-based approach [63].

The probabilistic graphical model-based approach uses the hierarchy directed Bayesian network to exploit the conditional dependence of unobserved variables to the previously observed applications. In the context of this study, the unobserved variables are the power consumption and execution time of a target application on remaining system configurations that we want to estimate.

The Bayesian model in this approach is drawn in Figure 5.5, where $\mathbf{y}_M$ is the partially unobserved application (i.e., the target application whose only measurement of sample configurations are known) that need to be estimated; $\mathbf{y}_i$ to $\mathbf{y}_{M-1}$ is fully observed applications whose power and execution time are known and measured offline; and nodes $\mathbf{z}_i$ together with their root are the hidden nodes.

In this probabilistic approach [63], the system has n configurations. The vector $\mathbf{y}_i \in \mathbb{R}^n$ represents the power estimate of application i in all n configurations of the system. $\{\mathbf{y}_i\}_{i=1}^{M-1}$ represent the power consumption data for the known applications while $\mathbf{y}_M$ represents the power consumption for the unknown application. There are a small number of observations (i.e., power consumption) for the unknown application running on the same number of observed configurations Ω_M where $|\Omega_M| \ll n$.

The approach objective is to estimate the power for application M for all configurations that are

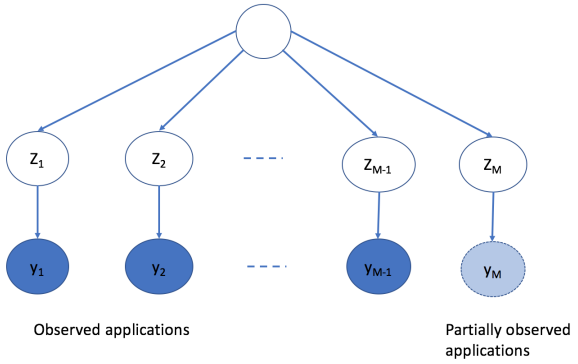


Figure 5.5: Bayesian Model

unobserved. The model is described below,

$$\begin{aligned} \mathbf{y}_i|\mathbf{z}_i &\sim N(\mathbf{z}_i, \sigma^2\mathbb{I}), \\ \mathbf{z}_i|\mu, \Sigma &\sim N(\mu, \Sigma) \\ \mu, \Sigma &\sim N(\mu_0, \Sigma/\pi)IW(\Sigma|\upsilon, \Psi) \end{aligned} \tag{5.1}$$

where $\mathbf{y}_i \in \mathbb{R}^n$, $\mathbf{z}_i \in \mathbb{R}^n$, $\mu \in \mathbb{R}^n$, $\Sigma \in \mathbb{R}^{n\times n}$.

The Equation 5.1 means that the power $\mathbf{y}_i$ for each of the ith application follows multivariate-Gaussian distribution with mean $\mathbf{z}_i$ and a diagonal covariance matrix $\sigma^2\mathbb{I}$; the hidden nodes $\mathbf{z}_i$ follows multivariate-Gaussian distribution with mean μ and covariance Σ. And, μ and Σ follows normal-inverse-Wishart distribution with parameters μ_0, π, Ψ, υ.

Since μ_0, π, Ψ, υ are the hyper-parameters, which were set as $\mu_0 = 0$, $\pi = 1$, $\Psi = 1$, $\upsilon = 1$, the main objective is now to find the remaining parameter set θ (including μ, Σ, and σ) that maximizes the likelihood function (i.e., the probability function of the observed outcomes given the parameter values).

In order to find θ, the EM (Expectation Maximization) algorithm, a popular statistics approach is applied. The EM algorithm is the iteration of the two steps: expectation (E) and maximization (M) until convergence. The Expectation step finds a function for the expectation of the log of the likelihood based on the current estimate for the parameters. The Maximization step computes the parameter set to maximize the expected log-likelihood from the Expectation step. Then, the found parameter set is forward to the next Expectation step. The Expectation step computes the expectation of the log of the likelihood $\mathbb{E}(\mathbf{z}_i)$ and covariance for $\mathbf{z}_i$ based on Equation 5.2 and Maximization step computes the parameter set θ (including μ, Σ, and σ) using Equation 5.3.

$$\begin{aligned} \mathrm{Cov}(\mathbf{z}_i) &= \left(\frac{\mathrm{diag}(L_i)}{\sigma^2} + \Sigma^{-1}\right)^{-1} \text{ and} \\ \mathbb{E}(\mathbf{z}_i) &= \mathrm{Cov}(\mathbf{z}_i)\left(\frac{\mathrm{diag}(L_i)\mathbf{y}_i}{\sigma^2} + \Sigma^{-1}\mu\right). \end{aligned} \tag{5.2}$$

$$\begin{aligned} \mu &= \frac{1}{M+\pi}\sum_{i=1}^{M}\mathbb{E}(\mathbf{z}_i), \\ \Sigma &= \frac{1}{M+1}\left(\sum_{i=1}^{M}\mathrm{Cov}(\mathbf{z}_i) + (\mathbb{E}(\mathbf{z}_i)-\mu)(\mathbb{E}(\mathbf{z}_i)-\mu)'\right) + \pi\mu\mu' + \mathbb{I}, \\ \sigma^2 &= \frac{1}{||L||_F^2}\sum_{i=1}^{M}\mathrm{tr}\left(\mathrm{diag}(L_i)(\mathrm{Cov}(\mathbf{z}_i) + (\mathbb{E}(\mathbf{z}_i)-\mathbf{y}_i)(\mathbb{E}(\mathbf{z}_i)-\mathbf{y}_i)')\right), \end{aligned} \tag{5.3}$$

In Equation 5.2 and 5.3, $|\Omega_i|$ is the set of observed indices for the ith application and L is an indicator matrix with $L(i,j) = 1$ if $j \in \Omega_i$ and 0 otherwise, meaning that $L(i,j) = 1$ if we have observed application i in system configuration j. $\mathbb{E}(\mathbf{z}_i)$ is the expectation of the log of the likelihood and $\mathrm{Cov}(\mathbf{z}_i)$ is covariance for $\mathbf{z}_i$. The time performance is estimated with the same EM algorithm.

It is noted that before starting the Expectation Maximization algorithm, a regression function (i.e., polynomial multivariate regression) using configuration values (the number of cores, memory control, and frequency index) as predictors is applied to initialize the data of unobserved configurations of the target application (i.e., the values of $\mathbf{y}_M$ vector).

The probabilistic graphical models, in general, targets applications which have long running time or many repeated instances as well as applications which have phases to change configuration online. Energy consumption of such applications can be reduced by using probabilistic graphical model-based approach. Besides, because the training data is obtained from the previously observed applications and the sampling data is obtained by running a target application directly with randomly selected system configurations, this modeling approach requires no prior knowledge of the target application or low-level details of system architecture (e.g., modeling power of each platform instruction units or components).

5.2 A Holistic Tuning Approach for Heterogeneous Systems

Based on the probabilistic graphical model-based approach for homogeneous systems [63] explained in Section 5.1, this Section describes the improvements of REOH, the holistic tuning approach for heterogeneous systems.

5.2.1 Unifying platform configurations

Unlike the previous (homogeneous) probabilistic graphical models approach [63], the REOH approach proposed in this study is for heterogeneous systems including different platforms with different architectures. The probabilistic modeling approach requires experimental data from a set of configurations that can be tuned during runtime.

The configurations must be pre-defined and provided in training data. For REOH, the configurations are the combination of the number of cores, the core frequency and the number of memory controllers. An example of CPU configuration is 24 cores running at frequency 1.7 GHz with two memory channels. Each platform architecture has its own hardware specification with different numbers of cores, the core frequencies or memory controllers [63]. For heterogeneous systems including several platforms with different architectures, in order to apply the probabilistic approach, finding the equivalence of configurations from different platforms is essential.

In this section, we propose a methodology to convert the configurations of different platforms. We consider the peak compute flops and peak memory bandwidth when finding the equivalence of the configurations of different platforms. The study by Lee et.al. [52] provided a comparison of CPU and GPU performance on 14 kernels considering architectural differences such as processing element (or PE) and bandwidth differences. The average performance (in flops) of each processing element is computed by dividing the platform computing flops by the total number of processing elements in the platform: $Flops_{PE} = \frac{PeakFlops}{TotalPE}$. In the context of this study, the total processing elements are the number of cores available in the platform. E.g., $Flops_{CPUcore} = \frac{PeakFlops_{CPU}}{TotalCores_{CPU}}$ and $Flops_{GPUcore} = \frac{PeakFlops_{GPU}}{TotalCores_{GPU}}$.

Therefore, to unify the number of cores in GPU (or $nGPUcore$) with an equivalent number of cores in CPU (or $nCPUcore$), we compare the performance of CPU cores and GPU cores as in Equation 5.4:

$$\begin{aligned} nGPUcore &= \frac{Flops_{GPUcore}}{Flops_{CPUcore}} \times nCPUcore \\ &= \frac{PeakFlops_{GPU}}{TotalCores_{GPU}} \times \frac{TotalCores_{CPU}}{PeakFlops_{CPU}} \times nCPUcore \end{aligned} \tag{5.4}$$

In our heterogeneous system, there are two platforms: CPU Xeon E5-2650Lv3 has 24 cores and peak performance as 115.2 GFlops while GPU Nvidia Quadro K620 has 384 cores with peak performance as 860 GFlops. The average performance for a CPU core is $\frac{115.2}{24} = 4.8$ GFlops while the average performance for a GPU core is $\frac{860}{384} = 2.24$ Gflops. One GPU core is equivalent to $\frac{24}{115.2} * \frac{860}{384} = 0.47$ CPU core, which is approximately half of the performance of one CPU core. Therefore, one GPU core is approximately equivalent to 0.5 CPU core.

Similarly, we convert the number of memory controllers of GPU (or $nGPUmem$) to the number of memory controllers in CPU (or $nCPUmem$) based on peak memory bandwidth of CPU and GPU

as in Equation 5.5. CPU Xeon E5-2650L and GPU Nvidia Quadro K620 has a peak bandwidth 68 GB/s and 28.8 GB/s respectively. Both CPU and GPU platforms have two memory controllers. The bandwidth of one memory controller of GPU ($GB_{GPUcore}$) is equivalent to $\frac{28.8}{68}$ CPU counterpart, which is approximately half of the bandwidth of a CPU memory controller.

$$nGPUmem = \frac{GB_{GPUcore}}{GB_{CPUcore}} \times nCPUmem \tag{5.5}$$

The frequencies in REOH approach are represented by integer numbers as indexes. The increasing order of frequency indexes reflects the increasing order of frequency values. For example, the experimented CPU has 8 frequencies (i.e., 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.81GHz) represented by the numbers (i.e., 0, 1, 2, 3, 4, 5, 7, 8, respectively). The experimented GPU has one frequency (i.e., 1.73 GHz) represented by the number 6.

5.2.2 Total energy consumption of heterogeneous systems

In the REOH holistic approach, we target to optimize the total energy consumption of heterogeneous systems, including both static (idle) and dynamic energy of every platform in the system while the existing (homogeneous) approaches only consider the energy consumption of individual platform in isolation.

Unlike the homogeneous approach that considers CPU energy and GPU energy in isolation, the holistic approach considers CPU energy and GPU energy together. It is because although the application runs on GPU (resp. CPU), idle CPU (resp. GPU) consumes energy as well (i.e., static energy). This is one of the reasons that make the most energy efficient configurations from homogeneous approaches not always the most energy efficient configurations in heterogeneous systems. By measuring energy consumption of 18 applications listed in Table 5.2, we show the difference between the optimal dynamic energy and the total energy including the static energy of the idle platform and dynamic energy. Figure 5.6 shows the optimal dynamic energy of CPU and GPU while 5.7 shows the optimal total energy of the running platform. The optimal configurations for each application from the two sets of data (i.e., the dynamic energy data and the total energy data) are not always the same. For example, from the dynamic energy data, running application 17 on GPU consumes less energy than running it on CPU while from the total energy data, running application 17 on CPU is more energy-efficient than on GPU.

The research question that the REOH approach wants to address is: which platform (CPU or GPU), together with its configuration, in a heterogeneous system is the most energy efficient for executing a given application. In our research context, when an application is executed by ones of the platforms (e.g., active platforms), the other platforms are in idle mode. The energy consumption of the active platforms includes their static and dynamic energy while the energy consumption of the idle platforms includes only their static energy. The total energy consumption of a whole

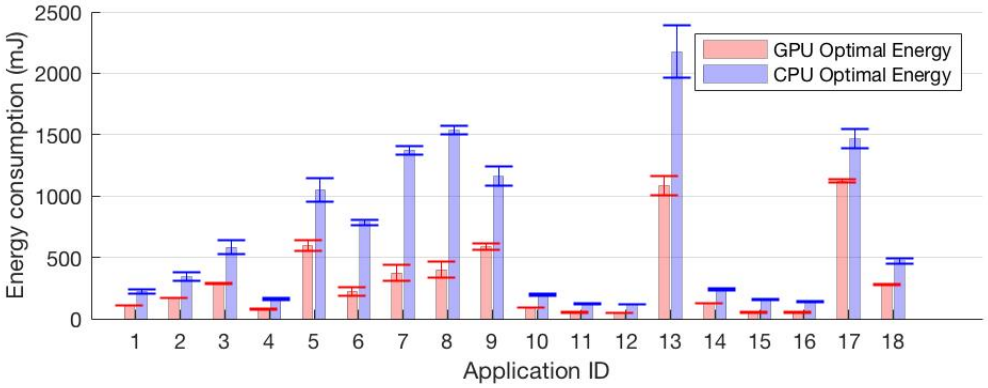


Figure 5.6: Optimized energy consumption of CPU and GPU from homogeneous approach

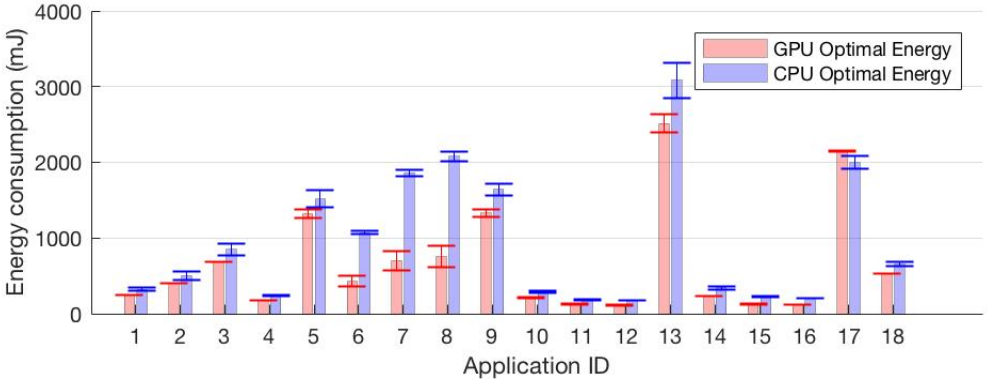


Figure 5.7: Optimized energy consumption of CPU and GPU from the heterogeneous approach, which considers both static and dynamic energy of each platform

heterogeneous system includes not only the energy of active platforms but also the energy of idle platforms as Equation 5.6. The energy consumption of active platforms includes static and dynamic energy while the energy consumption of idle platforms is the static energy. In Equation 5.6, the heterogeneous system has m platforms. The active platforms are platforms (1, 2, .., n) and the idle platforms are platforms (n+1, n+2, .., m).

$$E^{total} = \sum_{i=1}^{n}(E_i^{static} + E_i^{dynamic}) + \sum_{j=n+1}^{m} E_j^{static} \tag{5.6}$$

In our heterogeneous system used for validating the REOH approach, there are two platforms CPU and GPU. If an application is run on CPU while GPU is idle, the total energy is computed as $E_{CPU}^{total} = E_{CPU}^{static} + E_{CPU}^{dynamic} + E_{GPU}^{static}$. If an application is run on GPU while CPU is idle, the total energy is computed as $E_{GPU}^{total} = E_{GPU}^{static} + E_{GPU}^{dynamic} + E_{CPU}^{static}$. This is one of the improvements of

Table 5.1: Application categories based on dwarf list

Dwarf	**Performance Limit [12]**	**Benchmark [4]**
Graph Traversal	Memory Latency	B+Tree BFS
Structured Grid	Memory Bandwidth	HeartWall Particle Filter
Unstructured Grid	Memory Latency	CFD Solver Back Propagation
Dense linear algebra	Computation	LUD kmeans
Sparse Matrix	50%Computation 50% Memory Bandwidth (cf. Figure 9 in [12])	
Dynamic Programming	Memory Latency	Path Finder Needleman-Wunsch
N-body	Computation	LAVAMD
Spectral	Memory Latency	GPUDWT

REOH holistic approach compared to the existing (homogeneous) approaches.

In our experiments, we measured static energy when letting the application go to sleep mode using sleep() command from unistd.h, the POSIX operation system API. The static power is a fixed value for each platform. In our experiments shown in Figure 5.7, the static energy for each application on a specific platform is the product of the static power of the platform multiplied with the execution time of the application. The power-gating mechanism, when enable, shuts down the power supply of parts of the platforms that are in idle mode but not the whole platform. Therefore, when the power-gating is possible, the minimum power of the platform (when power-gating as many components as possible) is considered as the (fixed) static power in the REOH model.

5.2.3 Application categories

We propose a selected set of applications for experimenting and devising a general training data set which can cover a wide range of communication and computation patterns.

A training data set obtained offline is required by the probabilistic network approach. The main objectives of the training data set is to represent a wide range of computation and communication patterns and characteristics. In order to identify such varied set of patterns, we consider the pattern categories based on Berkeley dwarfs [12] and its corresponding benchmarks in the Rodinia benchmark suite [4].

We summarize the dwarf list and their corresponded benchmarks based on their categories and characteristics as in Table 5.1. Each of the dwarfs has performance limit due to computation, memory bandwidth or memory latency as shown in the second column (e.g., Performance Limit). The third column shows the benchmarks belonging to the dwarf.

There are several impact factors that affect the application performance and its optimization strategies such as algorithm design, execution configuration, control flow, memory types, memory access pattern and instruction count [24]. These factors are represented by three categories of performance limits: computation, memory bandwidth and memory latency [12]. In order to select the benchmarks that represent a wide range of applications behaviors, we choose a set of benchmarks that cover all three categories of the performance limits such as Kmeans, BFS, Particle Filter and CFD. The four benchmarks belong to the first four dwarfs in Table 5.1.

We chose Rodinia [4] benchmarks to validate our approach because it provides implementations for a variety of platforms (e.g., CPU and GPU) and programming models (e.g., OpenCL, CUDA, OpenMP). Among the supported programming models of Rodinia, OpenCL implementations are selected since OpenCL library is supported on various architectures such as CPU, GPU, and accelerators.

Moreover, the problem size can also impact the benchmark performance and its optimization strategy [24, 78]. For each chosen benchmarks, we also select a set of input that covers a varying range of benchmark patterns.

The selected input was generated using the data generators from Rodinia, in which the sample sizes were chosen to grow exponentially to cover a various range of input sizes. BFS has input graphs with sizes varying from 512kB to 8MB. CFD experiments are conducted with only three input sizes due to the unavailability of the input generator and limited input provided by Rodinia. Kmeans has the input generating from two parameters: the number of objects and the number of features. For instance, in Table 5.2, the input name 1000_34 means there are 1000 objects and each object has 34 features [7]. Particle Filter has the input generating from three parameters as its three dimensions. For instance, the input name 128_10_1000_dp means that the input dimensions are 128x128x10 with 1000 particles and particles are double type [22]. For each input size and configuration, each benchmark is performed five times and the measurement of average and deviation values are stored in the training data set.

5.3 Energy Saving - Experimental Results

In this section, we validate the REOH approach by experimental studies: how close to the optimal configuration (by the brute-force approach) the configuration by the REOH approach is. The optimal configuration means the best platform and its best setting in term of energy consumption. The REOH approach predicts the best configurations (i.e., the best platform and its best setting in term

Table 5.2: Application details

Application ID	Benchmark	Input
1	BFS	graph1M
2	BFS	graph2M
3	BFS	graph4M
4	BFS	graph512k
5	BFS	graph8M
6	CFD	fvcorr.domn.097K
7	CFD	fvcorr.domn.193K
8	CFD	missile.domn.0.2M
9	Kmeans	1000000_34
10	Kmeans	100000_34
11	Kmeans	10000_34
12	Kmeans	1000_34
13	Kmeans	3000000_34
14	ParticleFilter	128_10_100000_dp
15	ParticleFilter	128_10_10000_dp
16	ParticleFilter	128_10_1000_dp
17	ParticleFilter	128_2500_10000_dp
18	ParticleFilter	128_500_10000_dp

of energy consumption) based on the training data and sampling data.

5.3.1 Devise training data and sampling data

In REOH approach, the offline training data provides the knowledge of representative applications while the online sampling data provides the knowledge of the target application. The offline training data is the energy consumption and time performance of the selected applications running on all configurations of the heterogeneous system. By selecting the applications covering several categories explained in Section 5.2.3, we aim to cover a wide range of communication and computation patterns of applications in the offline training data. The online sampling data is the execution time and power consumption of a target application running on a small set of configurations (i.e., sample configurations) of the heterogeneous system to provide the REOH framework the knowledge about this target application. Base on both offline training data and online sampling data, a probabilistic graphical model estimates power consumption and execution time for the target application on all remaining configurations of the heterogeneous system. Based on the estimated results, the best configurations for energy consumption of the target application are identified.

In our experiments, the training data was obtained by running the 18 applications from Table 5.2 on all available configurations (i.e., 384 configurations of CPU and 9 configurations of GPU) of the targeted heterogeneous system including the two platforms CPU Xeon E5-2650L and GPU

Nvidia Quadro K620. The 384 configurations of CPU are the combination of 24 cores, 8 frequencies and 2 memory controllers. The CPU configurations (i.e., the combinations of cores, frequencies, memory controllers) are set by using *cpufrequtils* package and *numactl* library. The 9 configurations of GPU are the workgroup sizes assigned to applications, such as 1, 2, 4, 8, 16, 32, 64, 128, 256 work units, which affect the occupancy and the number of active multiprocessors of GPU. Time and energy measurements were performed with MeterPU [57] library using Intel PCM for CPU and Nvidia NVML for GPU. Each application was run five times for each configuration and the mean and standard deviation values of measured performance and consumed energy are stored. Note that the minimum number of cores (respectively memory controller) is one in order to ensure that the application always completes in a finite amount of time.

The sampling data is the power and time performance data measured for a target application running on a small set of configurations. This set of configurations are randomly selected among a whole configuration set of the considered heterogeneous system (i.e., 393 configurations in total for both CPU and GPU). The number of sample configurations is identified in Section 5.3.2.

5.3.2 Approach validation

Based on the training data and sampling data, the probabilistic model is applied to estimate the energy consumption of the remaining configurations (namely, all possible configurations except for sample configurations). Noted that when sampling an application A, A's data is removed from the training data set. From the estimated energy consumption of all configurations, the best configuration which consumes the least energy is selected.

We compare the result of the REOH approach with the LEO approach [63], the state-of-the-art (homogeneous) approach based on a similar probabilistic model. REOH approach is applied on a heterogeneous system with both CPU and GPU data while LEO approach is applied on homogeneous systems (i.e., either on CPU platform with CPU data or GPU platform with GPU data). The details (i.e., data from which platform and data size) of training and sampling set for each approach are summarized in Table 5.3.

The probabilistic approach uses regression diagnostics (i.e., regstats function) [3] with full quadratic [5] as an input model. For REOH and LEO-CPU prediction, the regstats function has 3 predictors (i.e., the number of cores, the frequency index and the number of memory controllers) which creates 10 (i.e., $\frac{(3+1)\times(3+2)}{2}$) predictor variables [5]. The model for REOH and LEO-CPU, therefore, requires at least 10 observations (i.e., the number of sampling data). Since the considered GPU has less than 10 configurations, we only use one predictor (i.e., workgroup size) for the regression function when applying the probabilistic approach for GPU platform with GPU data only. The model for LEO-GPU requires at least 3 sampling data.

The prediction was performed with the total number of samples varying from 10 (the minimum samples requirement) to 50 samples. The accuracy of the model increases when the number of

Table 5.3: Training and sampling data for each approach

	Training Data		**Sampling Data**	
Approach	**Platform**	**Size**	**Platform**	**Size**
LEO-CPU	CPU	384x18	CPU	15x1
LEO-GPU	GPU	9x18	GPU	3x1
REOH	CPU+GPU	393x18	CPU	15x1

samples increases to 15. After reaching 15 samples, the accuracy of the model does not significantly change when taking more samples. Therefore, we choose to sample 15 data on 15 configurations when performing model prediction with REOH and LEO-CPU approach. For LEO-GPU, we choose the number of sampling data as 3.

In this validation, we compare the most energy-efficient configuration by the REOH approach for a heterogeneous system containing a CPU and a GPU to the most energy-efficient configurations by the LEO approach for a homogeneous system with a CPU platform and the most energy-efficient configurations by the LEO approach for a homogeneous system with a GPU platform. Moreover, we also compare the REOH results with the optimal results by the brute-force approach that has all measured data of all platforms (i.e., CPU and GPU) available. The brute-force approach always chooses the optimal configuration.

Figure 5.8 shows the energy consumption (in mJ) of the configurations selected by the four approaches for 18 applications and Figure 5.9 shows the energy consumption difference between the three approaches (LEO-CPU, LEO-GPU [63] and REOH) and the Brute-force approach. The list of applications and their ID are summarized in Table 5.2.

The results show that for 17 out of 18 applications, the REOH approach predicts the close results to LEO-GPU approach and the Brute Force approach (up to 0.9% more energy consumption to LEO-GPU and within 5.7% deviation to Brute Force) except application 11. Unlike other applications where the performance increases when the number of cores increases, application 11 has the performance increased in the first 12 cores and decreased in the second 12 cores as shown in its experimental data (note that the platform has two 12-core CPUs). Application 11 has a different performance pattern than other applications which leads to the less precise prediction of REOH on application 11.

REOH also predicts better results than LEO-CPU except application 17. LEO-CPU approach has better prediction only on the application 17: 5.7% less energy consumption than the REOH approach. Application 17 has the best configuration on the CPU platform and the LEO-CPU approach, which considers only CPU data, is expected to be more accurate. However, its energy difference on the CPU platform between LEO-CPU and REOH approaches is marginal. Even though REOH approach predicts a configuration with higher energy consumption than LEO-CPU approach

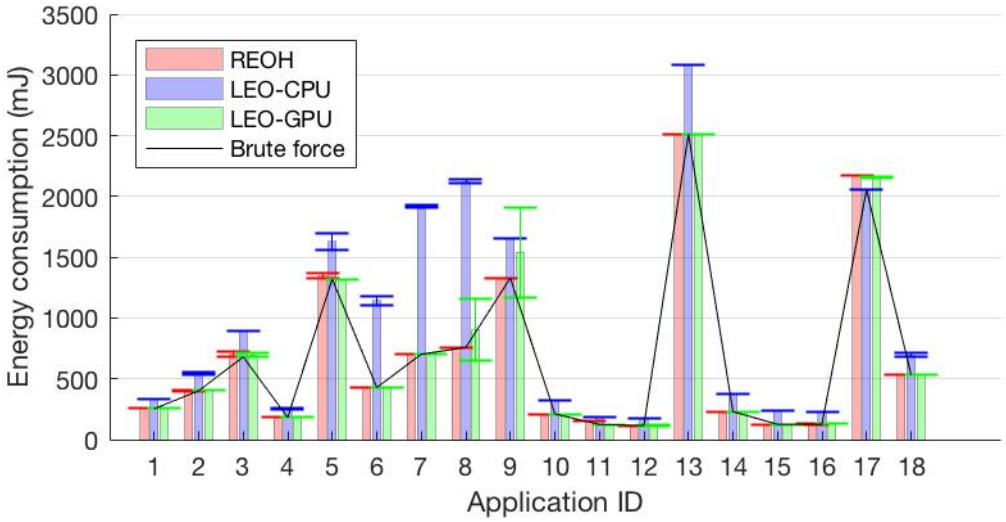


Figure 5.8: Energy comparison of the four approaches: REOH, LEO-CPU, LEO-GPU and Brute Force

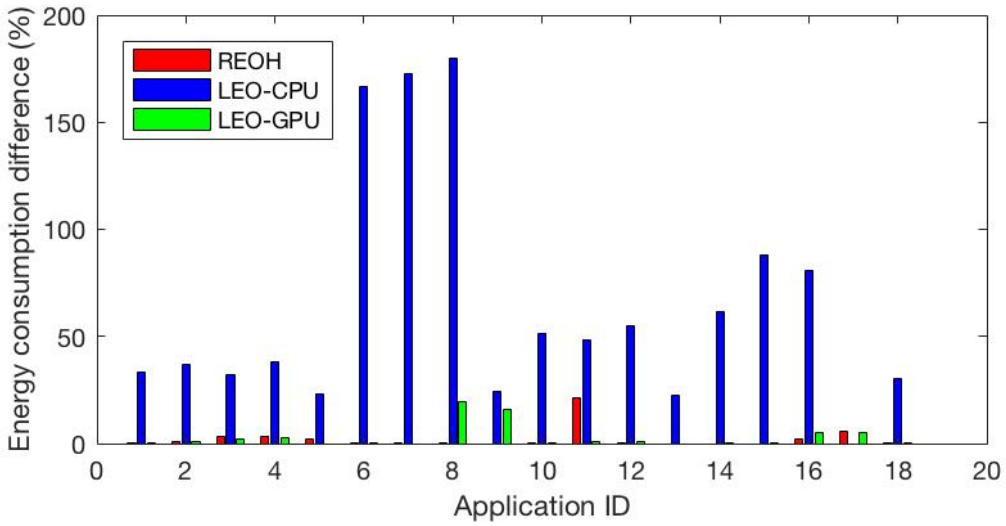


Figure 5.9: Percentage of the differences in energy consumption of REOH, LEO-CPU and LEO-GPU approach compared to Brute Force approach

at application 17, its energy consumption is also within 5.7% of the optimal energy consumption by the brute-force approach (cf. Figure 5.9).

The results have confirmed that the REOH approach can use the training set from selected applications to predict competitive configurations (within 5.7% of the optimal in 17 applications) in term of energy consumption. Moreover, the REOH approach only needs 15 samples from CPU data

to predict the most energy-efficient configuration while LEO requires two predictions on data from two separate platforms, either CPU or GPU data. The total number of samples when using LEO approach is $15 + 3 = 18$, which is 20% more sampling numbers as compared to REOH approach. By using REOH approach, the system is beneficial in two ways: not sampling GPU data and save 17% (i.e., $\frac{3}{15+3}$) the number of sampling runs.

5.3.3 Discussion

We have chosen the application set for the training phase as described in Section 5.2.3. The application set is chosen in the attempt of representing a wide range of application communication and computation patterns. However, the number of included benchmarks are limited. Each Rodinia benchmark has its own way of implementation and only some Rodinia benchmarks support specifying the platform of a heterogeneous system to run them. The framework requires separate executable files of the same benchmark for each of the platforms (e.g., CPU and GPU). Therefore, only four benchmarks which support specifying the platform of a heterogeneous system to run them are included in the training data. This limited number of benchmarks are compensated with the various input sizes of each benchmark included in the training data. Moreover, the four chosen benchmarks cover all three categories of performance limits [12] and belong to four dwarfs in Berkeley dwarf list [12]. Nevertheless, in the opinion of the author, the generality and accuracy of the approach will be improved when all represented benchmarks of all dwarfs in Berkeley list are added to the training data set.

5.4 Energy-optimizing Runtime Framework

Based on the new REOH approach, an open-source runtime framework has been developed to provide users with an energy-efficient system configuration for a given executable running on a heterogeneous system. The framework is publicly available at https://github.com/uit-agc/REOH.

5.4.1 Framework design

Figure 5.10 shows an overview of our framework.

Energy Wrapper The energy wrapper consists of an executable that is responsible for setting platform configurations and measuring energy and execution time of a given application. Each application should provide two executables: one for the CPU platform and one for the GPU platform, assuming the underlying heterogeneous system consists of CPU and GPU platforms. Time and energy measurements were performed using MeterPU [57], instantiated with Intel PCM for CPU and Nvidia NVML for GPU. The executables are executed using the POSIX `system()` command.

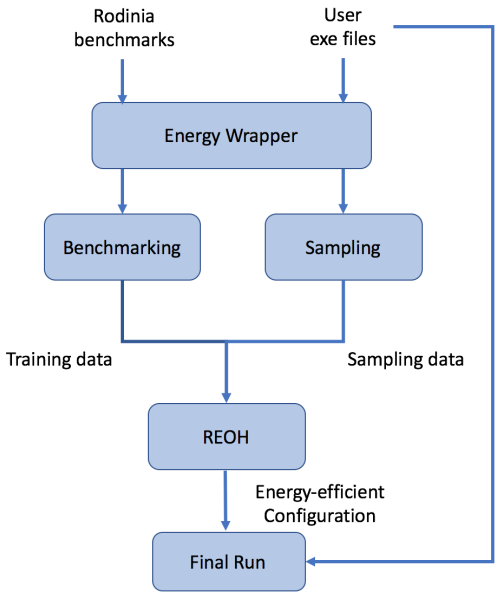


Figure 5.10: Prototype Overview

Benchmarking The module is to obtain the training data for a given heterogeneous system by executing the energy wrapper module for 18 applications (cf. Table 5.2) on all system configurations. This step only needs to perform once for different workloads.

Sampling The sampling is performed by executing the energy-wrapper for user executables on sample configurations. This module is to provide the sampling data in order to estimate the energy consumption of the executables on all configurations. This step is performed for every given application and its executables from users.

The output data of both the Benchmarking and Sampling module is converted to the appropriate format using the scripts provided in this framework. During transformation, we also add static energy consumed by CPU and GPU. The static energy was measured by recording the energy measurements over 20 seconds for each platform using MeterPU[57]. The measurement was done once to measure the static power of each platform in the heterogeneous system. The static power was stored for later use.

REOH The energy-optimizing module estimates the energy consumption of all configurations of the heterogeneous system based on the training data set and the sampling data set. Then it provides an appropriate energy-efficient configuration to run the given application.

Final Run From the configuration provided by REOH module, the Final Run module runs the appropriate executable file (e.g., the executable files for CPU or GPU) on the provided configuration

and measure its energy consumption.

5.4.2 Implementation details

In order to measure the energy consumption of Rodinia benchmarks, two main modifications have been made. i) Changing the OpenCL setup to target our specific platforms. ii) Reading work-group sizes from the environment for GPU. Executables were produced for each platform and then were sent to the Energy-Wrapper module.

All benchmarks were performed on a 24-core Intel(R) Xeon(R) CPU E5-2650L v3, with 64 GB RAM, running Linux 3.10.0. For GPU experiments a Nvidia Quadro K620 was used. The prototype was implemented using C, C++, and OpenCL for the Rodinia benchmarks and Energy Wrapper component. Data transformation and benchmarking tools were implemented using Bash and AWK.

5.5 Related Work

The related work of this study is summarized in Table 1.2. The summary shows that the previous approaches are either for tuning the code variants (i.e., implementations of an algorithm or an application) and therefore restrict to specific applications [84, 64]; or for tuning configurations for a general application but only applicable to homogeneous system with a single unit type (i.e., CPU or GPU) [38, 92, 63, 64, 17, 6, 65, 29, 85].

Study about PowerCap [92] chooses the most suitable settings for energy efficiency but still meet the performance requirement. It operates based on feedback, then observes, decide and act. The approach requires feedback on an individual platform (e.g., CPU) which is not applicable for heterogeneous systems. POET [38] also chooses the system configuration to meet the energy requirement based on feedback and controllers. However, POET is also applicable to homogeneous systems with a single unit type (e.g., CPU).

There are a group of studies that provides power and performance models for GPU and GPGPU to predict the most energy-efficient DVFS configuration of GPU to run an application [64, 17, 6, 65, 29, 85, 61]. The models, however, are for the considered GPUs and not for heterogeneous systems. Studies [72, 91, 90] develop frameworks for workload partitioning on a type of heterogeneous systems (i.e., APUs), but they are mainly focused on improving performance instead of energy-efficiency.

There are two approaches for heterogeneous systems in Table 1.2: GreenGPU [58, 59] and the market mechanism [30]. GreenGPU [58, 59] targets iterative applications (i.e., applications have several iterations where the next iteration execution time can be predicted based on the current iteration) which is different from REOH (i.e., REOH is applicable for general applications and requires no prior knowledge of applications). The market mechanism [30] requires three analysis and optimization phases to match the user profile to the architecture profile and distribute the application to the hardware. The chosen configuration from the market mechanism [30] is at datacenter level

(i.e., its targeted configuration is a mix of CPUs and microprocessors) while the chosen configuration of REOH is at platform level (i.e., REOH configuration is a combination of the number of cores, frequency and memory controllers).

This study is inspired by LEO framework [63]. LEO chooses the best system configuration depending on the application and its input. This approach uses probabilistic graphical models to estimate the energy consumption of applications. However, LEO only considers a set of configurations of a CPU-based homogeneous system. The present approach (this study) apply a probabilistic network approach to identify the most energy-efficient configuration for an application running on heterogeneous systems and tune the configurations in runtime.

5.6 Conclusion

This study has proposed and validated REOH, a new holistic approach using probabilistic model to predict and select the optimal configurations in term of energy consumption of heterogeneous systems for a given application. This study has demonstrated that REOH can achieve almost optimal energy consumption (within 5.7% of the optimal energy consumption by the brute-force approach) while saving the energy consumption of 17% less sample runs. Based on the REOH approach, a runtime framework for executing given executables energy-efficiently is developed and provided as open source software for scientific purposes.

Chapter 6

Conclusion

The energy consumed by worldwide computing systems increases annually and becomes a major concern in information technology society. In order to tackle this issue, the scientific community and industry have proposed several approaches to reduce the energy consumption of computing systems. Modeling energy consumption of applications running on computing systems providing the understanding of how applications consume energy and the insight into how to improve its energy efficiency.

This thesis presents three modeling approaches for energy consumption of computing systems varying from homogeneous to heterogeneous systems. The three approaches complement each other by targeting different types of computing systems such as homogeneous systems (e.g., embedded system, CPU or GPU) and heterogeneous systems (e.g., containing both CPU and GPU) and accomplishing different research objectives such as estimating absolute energy values, analyzing energy complexity of multithreaded algorithms and choosing the most energy-efficient configurations in runtime.

In the first study, we propose new application-general fine-grained power models (namely, RTHpower) that are able to investigate the trade-offs between performance and power consumption on ULP embedded systems. The RTHpower models consider both platform and application properties. We validate the new RTHpower models on Movidius Myriad, an ultra-low-power embedded system by developing different sets of micro-benchmarks and three application kernels such as dense matrix multiplication (Matmul), sparse matrix vector multiplication (SpMV) and breadth first search (BFS). We investigate the RTH strategy on an ultra-low power embedded platform using the new RTHpower models. We propose and validate a framework to predict when to use the race-to-halt (RTH) strategy to minimizes energy consumption for a given application.

In the second study, we devise a new general energy model ICE to provide an analysis tool to identify the energy complexity of a wide range of multithreaded algorithms on high-performance platforms based on their *work*, *span* and *I/O* complexity. We conduct two case studies (i.e., SpMV

and matmul) to demonstrate how to apply the ICE model to find energy complexity of parallel algorithms. The validation results show the precise prediction regarding which validated SpMV algorithm (i.e., CSB or CSC) consumes more energy when using different matrix input types from Florida matrix collection. The results also show the precise prediction on which validated matmul algorithm (i.e., basic or cache-oblivious) consumes more energy.

In the third study, we develop REOH, a new holistic tuning approach for heterogeneous systems. The approach uses a probabilistic network, a machine learning technique to predict energy consumption of an application on all possible configurations of the heterogeneous systems. In order for REOH to provide the energy estimation on heterogeneous systems, we propose a method to unify the configurations of different platform types (e.g., CPU and GPU) and devise a training data set with a set of applications based on the knowledge of application characteristics from Berkeley dwarfs. REOH can predict the energy consumption of all possible configurations of a heterogeneous system and identify the most energy-efficient configuration. REOH approach has its energy consumption close to the optimal energy consumption by the Brute Force approach while saving the number of sampling runs by running one prediction for the whole heterogeneous system instead of running separate predictions for every individual device in the heterogeneous system. Based on the approach, we also develop an energy-optimizing runtime framework as an open-source that is able to select an energy-efficient configuration of a heterogeneous system to run a given application at runtime.

6.1 Future Work

In this thesis, a machine learning technique (e.g., probabilistic network) has been used for modeling energy consumption of heterogeneous systems. For future computing systems containing more complex architectures, modeling energy consumption of large-scale systems becomes more challenging. Therefore, machines learning techniques are essential to be able to learn from available energy data to predict the energy consumption of such large-scale systems and suggests suitable system configurations to achieve the most energy efficiency. The accuracy of the modeling approaches can also be improved by identifying the most suitable techniques in a given context.

One of our future directions is to apply different machine learning techniques to model energy consumption, identify the most energy-efficient configuration and develop a more portable runtime framework. The probabilistic network approach used in this thesis requires a training data set obtained in advance for each considered system. When changing the underlying system, the training data set need to be collected again. This reduces the portability of the approach. In the context where energy training data can not be obtained in advance, investigating how to estimate energy consumption in runtime by using other machine learning techniques (e.g. reinforcement learning) is potential to improve both energy-efficiency and approach applicability.

Moreover, with heterogeneous systems, an application can be run coordinately by a task scheduler on multiple platforms simultaneously in the same execution. The modeling approaches presented in this thesis can be further developed to support a runtime scheduler to distribute the tasks of applications to different platforms in a heterogeneous system. By increasing the utility of each individual device in a heterogeneous system, we aim to reduce the static energy consumption and improve their energy efficiency.

Appendix A

Paper I

Power Models Supporting Energy-Efficient Co-Design on Ultra-Low Power Embedded Systems

Vi Ngoc-Nha Tran
Department of Computer Science
UiT The Arctic University of Norway
Tromso, Norway
Email: vi.tran@uit.no

Brendan Barry
Movidius Ltd.
Dublin, Ireland
Email: brendan.barry@movidius.com

Phuong Hoai Ha
Department of Computer Science
UiT The Arctic University of Norway
Tromso, Norway
Email: phuong.hoai.ha@uit.no

***Abstract*—The energy efficiency of computing systems can be enhanced via power models that provide insights into how the systems consume power. However, there are no application-general, fine-grained and validated power models which can provide insights into how a given application running on an ultra-low power (ULP) embedded system consumes power.**

In this study, we devise new fine-grained power models that provide insights into how a given application consumes power on an ULP embedded system. The models support architecture-application co-design by considering both platform and application properties. The models are validated with data from 35 micro-benchmarks and three application kernels, namely dense matrix multiplication, sparse matrix vector multiplication and breadth first search, on Movidius Myriad, an ultra-low power embedded system. The absolute percentage errors of the model are at most 8.5% for micro-benchmarks and 12% for application kernels. Based on the models, we propose a framework predicting when to apply race-to-halt (RTH) strategy (i.e., running an application with a maximum setting) to a given application. For the three validated application kernels, the proposed framework is able to predict when to use RTH and when not to use RTH precisely. The experimental results show that the prediction of our new RTH framework is accurate and we can save up to 61% energy for dense matrix multiplication, 59% energy for sparse matrix vector multiplication by using RTH and 5% energy for breadth first search by *not* using RTH.

I. Introduction

Devising accurate power models is crucial to gain insights into how a computer system consumes power and energy. By knowing the energy consumption of individual components on a specific computing architecture, researchers and practitioners can design and implement new approaches to reduce the energy consumed by a certain algorithm on a specific platform [1–4]. Significant efforts have been devoted to devising power and energy models, resulting in several seminal papers in the literature, such as [5–15] modeling power of architectures or applications.

Jacobson et al. [5] proposed accurate power modeling methodologies for POWER-family processors while GPUWattch [6] and McPAT [7] are robust power models for GPUs and CPUs. Alonso et al. [8] proposed energy models for three key dense-matrix factorizations. Roofline model of energy [9, 10] considers both algorithmic and platform properties. However, the Roofline model does not consider the number of cores running applications as a model parameter (i.e., coarse-grained models). Theoretical models by Korthikanti et al. [11, 12] are based on strong theoretical assumptions and are not yet validated on real platforms. Koala model [15] requires the system supported dynamic voltage and frequency scaling (DVFS) and short frequency switching delay in order to gain energy saving from its methodology. Only two x86-based platforms among 10 validated platforms gain energy saving results which are presented in the paper. Imes et al. [13] provided a portable approach to make real-time decision and run the chosen configuration to minimize energy consumption. However, the approach requires systems supporting hardware resource (e.g., model-specific register) to expose energy data to the software during run-time. Mishra et al. [14] used probabilistic approach to find the most energy-efficient configuration by combining online and offline machine-learning approaches. This approach requires a significant data collected to feed to its probabilistic network.

Recently, ultra-low power (ULP) embedded systems have become popular in the scientific community and industry, especially in media and wearable computing. ULP embedded systems have different architecture from the general-purpose architectures (e.g., CPU and GPU). As the result, the approach to model the power of ULP systems needs to be customized for their architecture. ULP systems can achieve low energy per instruction down to a few pJ [16]. Alioto [16] mentioned that techniques such as pipe-lining, hardware replication, ultra-low-voltage memory design and leakage-reducing make a system ultra-low power. In order to model ULP systems where energy per instruction can be as low as few pJ, more accurate fine-grained approaches are needed. For instance, the dynamic power P^{dyn} of operations in Table I, which is as low as 13 mW, cannot be measured by using the prior coarse-grained approaches.

Two of the most popular strategies to reduce the energy consumption are Dynamic Voltage/Frequency Scaling (DVFS) [17] and race-to-halt (RTH) (i.e, system is run as fast as possible, and then switched to idle state to save energy) [18]. For new embedded systems which do not support DVFS fea-

This work has received funding from the European Union Seventh Framework Programme (EXCESS project, grant no. 611183) and from the Research Council of Norway (PREAPP project, grant no. 231746/F20).

978-1-5090-3076-7/16/$31.00 ©2016 IEEE

tures such as Movidius Myriad [19], RTH is one of remaining choices for saving energy. In fact, Myriad supports a power management feature to power on/off individual cores. RTH strategy says that the system should execute the application as fast as possible and then go to sleep to save energy. For ultra-low power embedded systems, RTH might not be always true. However, to the best of our knowledge, there are no power models that supports investigating the trade-off between performance and energy consumption on ULP embedded system, and, particularly, whether the RTH strategy that is widely used in high-performance computing (HPC) systems is still applicable to ULP embedded systems.

In this study, we propose new RTHpower models which support architecture-application co-design by considering both platform and application properties. The RTHpower models are application-general since they characterize applications by their operational intensity [20] which can be extracted from any application. The RTHpower models are also practical since they are built and validated on Movidius platform with application kernels.

The contributions of this study are three-fold as follows:

- We propose new application-general fine-grained power models (namely, RTHpower) that provide insights into how a given application consumes power and give hints to investigate the trade-offs between performance and power consumption on ULP embedded systems. The RTHpower models support co-design on ULP systems by considering three parameter groups: platform properties, application properties (e.g., operational intensity and scalability) and execution settings (e.g., the number of cores executing a given application) (cf. Section II).
- We validate the new RTHpower models on an ultra-low power embedded system, namely Movidius Myriad. The models are trained and validated with power data from different sets of micro-benchmarks, two popular kernels from Berkeley dwarfs [21] and one data-intensive kernel from Graph500 benchmarks [22]. The three chosen application kernels are dense matrix multiplication (Matmul), sparse matrix vector multiplication (SpMV) and breadth first search (BFS). The model validation has percentage error at most 8.5% for micro-benchmarks and 12% for application kernels (cf. Section III).
- We investigate the RTH strategy on an ultra-low power embedded platform using the new RTHpower models. We propose a framework that is able to predict whether the RTH strategy minimizes energy consumption for a given application. We validate the framework using micro-benchmarks and application kernels and show that the framework prediction is accurate. From our experiments, we show real scenarios when to use RTH and when not to use RTH. We can save up to 61% energy for dense matrix multiplication, 59% energy for SpMV by using RTH and up to 5% energy for BFS by not using RTH (cf. Section IV).

TABLE I
$P^{dyn}(op)$ OF SHAVE OPERATIONS

Operation	Description	P^{dyn} (mW)
SAUXOR	Perform bitwise exclusive-OR on scalar	15
SAUMUL	Perform scalar multiplication	18
VAUXOR	Perform bitwise exclusive-OR on vector	35.6
VAUMUL	Perform vector multiplication	52.6
IAUXOR	Perform bitwise exclusive-OR on integer	15
IAUMUL	Perform integer multiplication	21
CMUCPSS	Copy scalar to scalar	20
CMUCPIVR	Copy integer to vector	13
LSULOAD	Load from a memory address to a register	28
LSUSTORE	Store from a register to a memory address	37

II. RTHPOWER - ANALYTICAL POWER MODELS

We first present a new power model for operation units and then develop it to the RTHpower models considering application properties.

A. A Power Model for Operation Units

The experimental results of the micro-benchmarks suite for operation units show that the power consumption of Movidius Myriad platform is ruled by Equation 1. In the equation, the static power P^{sta} is the required power when the Myriad chip is on, including memory storage power; the active power P^{act} is the power consumed when a core is on and actively performing computation work; the dynamic power $P^{dyn}(op)$ is the power consumed by each operation unit such as arithmetic units (e.g., IAU-integer, VAU-vector, SAU-scalar) or load/store units (e.g., LSU0, LSU1) in a SHAVE (Streaming Hybrid Architecture Vector Engine) core. The experimental results show that different operations have different $P^{dyn}(op)$ values as listed in Table I. Total dynamic power of a core is the sum of all dynamic power from involved units. If benchmarks or programs are executed with n cores, the active and dynamic power needs to be multiplied with the number of used cores. By using regression fitting techniques, the average value of P^{sta} and P^{act} from all micro-benchmarks are computed in Equation 2 and Equation 3. For ULP system such as Myriad, P^{sta} is significant low with only 62.125 mW. Table II provides the description of parameters in the proposed models.

$$P^{units} = P^{sta} + n \times \left(P^{act} + \sum_i P_i^{dyn}(op) \right) \quad (1)$$

$$P^{sta} = 62.125 \text{ mW} \quad (2)$$

$$P^{act} = 30 \text{ mW} \quad (3)$$

B. RTHpower Models for Applications

Since typical applications require both computation and data movement, we use the concept of operational intensity proposed by Williams et al. [20] to characterize applications.

TABLE II
MODEL PARAMETER LIST

Parameter	Description
P^{sta}	Static power of a whole chip
P^{act}	Active power of a core
$P^{dyn}(op)$	Dynamic power of an operation unit
P^{LSU}	Dynamic power of Load Store Unit
P^{ctn}	Contention power of a core waiting for data
m	Average number of active cores accessing data
n	Number of assigned cores to the program
I	Operational intensity of an application
α	Time ratio of data transfer to computation

An application can be characterized by the amount of computational work W and data transfer Q. W is the number of operations performed by an application. Q is the number of transferred bytes required during the program execution. Both W and Q define the operational intensity I of applications as in Equation 4. Characterizing applications by their intensity values is a conventional approach used in recent energy and performance modeling studies [9, 10, 23].

$$I = \frac{W}{Q} \tag{4}$$

As the time required to perform one operation is different from the time required to transfer one byte of data, we introduce a parameter to the models: time ratio α of transferring one byte of data to performing one arithmetic operation. Ratio α is the property of an application on a specific platform and its value depends on the application.

Since the time to access data and time to perform computation work can be overlapped, during a program execution, the core can be in one of the three states: performing computation, performing data transfer or performing both computation and data transfer in parallel. An application either has data transfer time longer than computation time or vice versa. Therefore, there are two models for the two cases.

- If an application has data transfer time longer than computation time, it is memory-bound and follows Equation 5. The execution can be modeled as two (composed) periods: one is when computation and data transfer are performed in parallel and the other is when only data transfer is performed. Fraction $\frac{W}{\alpha \times Q}$ represents the overlapped time of computation and data transfer. Fraction $\frac{\alpha \times Q - W}{\alpha \times Q}$ represents the remaining time for data transfer.

$$P = P^{comp||data} \times \frac{W}{\alpha \times Q} + P^{data} \times \frac{\alpha \times Q - W}{\alpha \times Q} \tag{5}$$

- If an application has computation time longer than data transfer time, it is compute-bound and follows Equation 6. The execution can be modeled as two periods: one is when computation and data transfer are performed in parallel and the other is when only computation is performed. Fraction $\frac{\alpha \times Q}{W}$ represents the overlapped time of computation and data transfer. Fraction $\frac{W - \alpha \times Q}{W}$ represents the remaining time for computation.

$$P = P^{comp||data} \times \frac{\alpha \times Q}{W} + P^{comp} \times \frac{W - \alpha \times Q}{W} \tag{6}$$

After converting W and Q to I by using Equation 4, the final models are simplified as Equation 7 and Equation 8,

$$P = P^{comp||data} \times \frac{I}{\alpha} + P^{data} \times \frac{\alpha - I}{\alpha} \tag{7}$$

$$P = P^{comp||data} \times \frac{\alpha}{I} + P^{comp} \times \frac{I - \alpha}{I} \tag{8}$$

where P^{data}, P^{comp} and $P^{comp||data}$ are explained below:

1) Data transfer power P^{data}: P^{data} is the power consumed by the whole chip when only data transfer is performed. P^{data} is computed by Equation 9. In Equation 9, m is the average number of cores accessing data in parallel during the application execution; contention power P^{ctn} is the power overhead occurring when a core waits for accessing data because of the limited memory ports (or bandwidth) or cache size in the platform architecture. Therefore, $n - m$ is the average number of cores waiting for memory access during the application execution.

$$\begin{aligned} P^{data} = {} & P^{sta} + min(m, n) \times (P^{act} + P^{LSU}) \\ & + max(n - m, 0) \times P^{ctn} \end{aligned} \tag{9}$$

2) Computation power P^{comp}: P^{comp} is the power consumed by the whole chip when only computation is performed. P^{comp} is computed by Equation 10. Each core runs its arithmetic units (e.g. IAU, SAU, VAU) to perform computation work. There is no contention power due to no memory access. Therefore, all assigned cores are active and contribute to total power.

$$P^{comp} = P^{sta} + n \times (P^{act} + \sum_i P_i^{dyn}(op)) \tag{10}$$

3) Computation and data transfer power $P^{comp||data}$: $P^{comp||data}$ is the power consumed by the whole chip when computation and data transfer are performed in parallel. $P^{comp||data}$ is computed by Equation 11. In this case, there is contention power due to the data waiting. $P^{comp||data}$ is different from P^{data} in the aspect that the active cores also run arithmetic units that contribute to total power as $\sum_i P_i^{dyn}(op)$.

$$\begin{aligned} P^{comp||data} = {} & P^{sta} \\ & + min(m, n) \times (P^{act} + P^{LSU} + \sum_i P_i^{dyn}(op)) \\ & + max(n - m, 0) \times P^{ctn} \end{aligned} \tag{11}$$

III. MODEL TRAINING AND VALIDATION

This section presents the experimental results of two sets of micro-benchmarks and three application kernels (i.e., *matmul*, SpMV and BFS) that are used for training and validating the models.

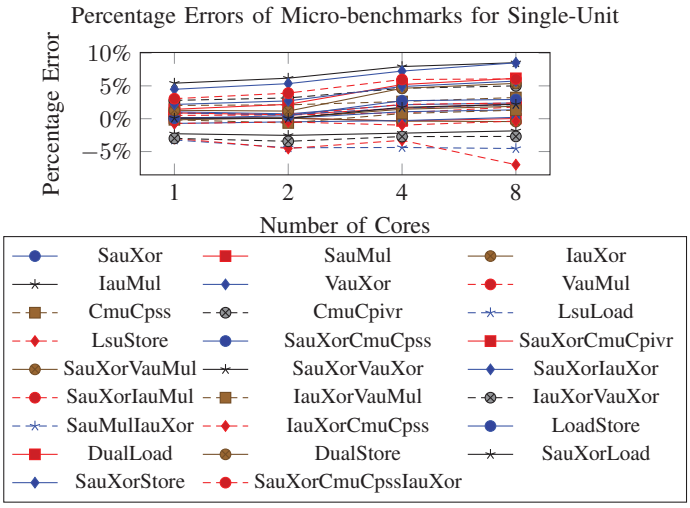


Fig. 1. The percentage errors of the model validation for *unit-suite* for 26 micro-benchmarks. The absolute percentage errors of unit-suite micro-benchmarks are at most 8.5%.

A. Model Validation with Micro-benchmarks

Analyses of experimental results are performed based on two sets of micro-benchmarks: 26 micro-benchmarks for operation units called *unit-suite* and 9 micro-benchmarks for different operational intensities called *intensity-suite*. The measured power data is collected by executing each micro-benchmark with different numbers of cores (i.e., 1, 2, 4, and 8 cores).

1) Micro-benchmarks for Operation Units: We assess the accuracy of the power model for operation units (Equation 1) using data from *unit-suite*. The model is trained with the power data collected when running *unit-suite* with one and two cores. The power data collected when running *unit-suite* with four and eight cores are used to validate the model. The micro-benchmarks of *unit-suite* are listed in Table III.

We validate the model and plot its percentage errors in Figure 1. Percentage error is calculated as $PE = \frac{measurement - estimation}{measurement}$. Figure 1 shows the percentage error of all three categories: one unit, two pipe-lined units and three pipe-lined units. The *absolute* percentage error is the absolute value of the percentage error. The model for operation units has the absolute percentage errors at most 8.5%. These results prove that the model is applicable to micro-benchmarks using either a single (e.g., performing bit wise exclusive-OR on scalar unit: SauXor) or pipe-lined arithmetic units in parallel (e.g., performing Xor on scalar and integer units, in parallel with copying from scalar to scalar unit: SauXorCmuCpssIauXor). The model also shows the compositionality of the power consumption not only for multiple cores but also for multiple operation units within a core.

2) Micro-benchmarks for Application Intensities: Since any application requires both computation and data movement, we design intensity-based micro-benchmarks which execute both arithmetic units (e.g., SAU) and two data transfer units (e.g., LSU0, LSU1) in a parallel manner. They are implemented with parallel instruction pipeline supported by the platform.

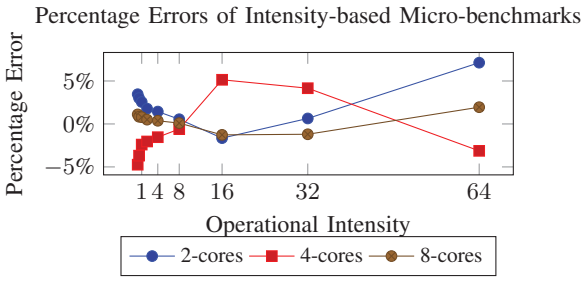


Fig. 2. The absolute percentage errors of RTHpower model fitting for *intensity-suite* (operational intensity I from 0.25 to 64) are at most 7%.

In order to validate the RTHpower models, this *intensity-suite* indicates different values of operation intensities (from 0.25 to 64). Operational intensity I is retrieved from the assembly code by counting the number of arithmetic instructions and the number of load/store instructions.

In the models, there are platform-dependent parameters such as α, m and P^{ctn}. The parameter values for each application operational intensity are derived from experimental results by using Matlab function $lsqcurvefit$. For the application intensities from 0.25 to 1, α is found bigger than operational intensity I meaning that data transfer time is longer than computation time. The estimated power model follows Equation 7. For operational intensity from 2 to 64, α is less than I meaning that data transfer time is less than computation time. The estimated power follows Equation 8.

We plot the percentage errors of the model fitting for intensity-based micro-benchmarks in Figure 2. In order to obtain a full range of estimated power with any values of intensities and numbers of cores, a fuzzy logic approach, namely Takagi Sugeno Kang (TSK) mechanism [24], is applied to the RTHpower models. Each intensity has a parameter set, including α, P^{ctn} and m. Based on the RTHpower models, each parameter set provides an individual function to estimate the power of an application based on its intensity value and a number of cores.

After the approach is implemented by using Matlab Fuzzy Logic toolbox, the full range of estimated power is obtained and presented in Figure 3. It is observed that when intensity value increases, the power-up (i.e., the power consumption ratio of the application executed with n cores to the application executed with 1 core) is also increased.

B. Model Validation with Application Kernels

The following application kernels have been chosen to implement and validate the RTHpower models on Myriad: *matmul* (a computation-intensive kernel), SpMV (a kernel with dynamic access patterns), and BFS (a data-intensive kernel) [22]. The three kernels are able to cover the two dimensions of operational intensity and speed-up as shown in Figure 4.

Matmul is proved to have high intensity and scalability [23]. SpMV has low operational intensity and high speed-up due to its parallel scalability [25]. BFS, on the other hand, has low operational intensity and saturated low scalability

TABLE III
MICRO-BENCHMARKS FOR OPERATION UNITS

Description	Micro-benchmark Name
10 micro-benchmarks using one unit (cf. Table I)	SAUXOR, SAUMUL, IAUXOR, IAUMUL, VAUXOR, VAUMUL, CMUCPSS, CMUCPIVR, LSULOAD, LSUSTORE
15 micro-benchmarks using two units	SAUXOR-CMUCPSS, SAUXOR-CMUCPIVR, SAUXOR-IAUMUL, SAUXOR-IAUXOR, SAUXOR-VAUMUL, SAUXOR-VAUXOR, SAUMUL-IAUXOR, IAUXOR-VAUXOR, IAUXOR-VAUMUL, IAUXOR-CMUCPSS, LOAD-STORE, DUALLOAD, DUALSTORE, SAUXOR-LOAD, SAUXOR-STORE
1 micro-benchmarks using three-units	SAUXOR-IAUXOR-CMUCPSS

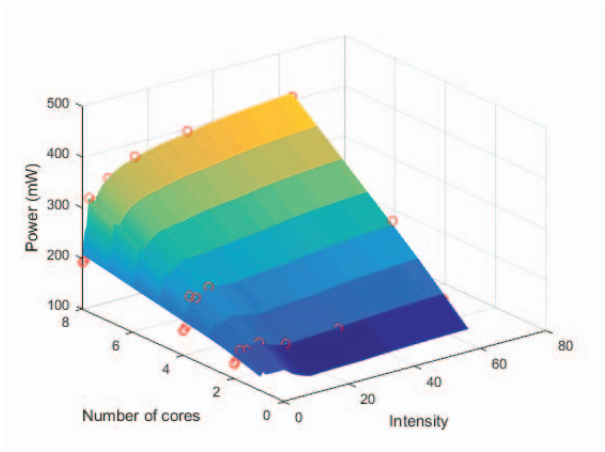


Fig. 3. The power range of varied intensities and numbers of cores from RTHpower models. The dots in the figure represent measurement data. The model switches from Equation 7 to Equation 8 at the intensity $I = 2$

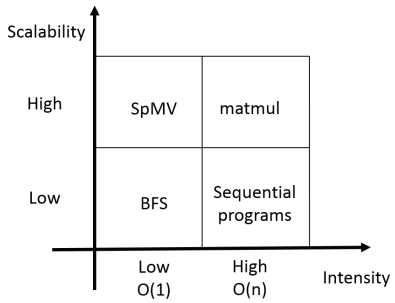


Fig. 4. Application Categories

[26]. Since the available benchmark suites in literature are not executable on Myriad platform, the three mentioned kernels have been implemented by the authors using the Movidius Development Kit for Myriad. As the RTHpower models will be used to predict whether the RTH strategy is an energy efficient approach for a given application, we focus mainly on two settings: the 8-core setting representing the RTH strategy (i.e., using all available cores of Myriad) and the 1-core setting representing the other extreme (i.e., using a minimum number of cores).

1) Dense Matrix Multiplication: Matmul has been implemented on Myriad by using both C and assembly languages. The *matmul* algorithm computes matrix C based on two input matrices A and B: $C = A \times B$. All three matrices in this benchmarks are stored in DDR RAM. Matrix elements are stored with float type equivalent to four bytes. The number of operations and accessed data are calculated based on matrix size n as: $W = 2 \times n^3$ and $Q = 16 \times n^2$ [23]. Intensity of *matmul* is also varied with matrix size as: $I = \frac{W}{Q} = \frac{n}{8}$. The experiments are conducted until matrix size 1024x1024, the largest size that Myriad RAM memory can accommodate.

We observe that operational intensity is not enough to capture other factors affecting power consumption such as the communication patterns and potential performance/power overheads due to the implementation. E.g., although a sequential version and a parallel version of a *matmul* algorithm have the same intensity, it is obvious that they have different communication pattern (intuitively, the sequential version does not have communication between cores). Since different parallel versions for different number of cores have different communication patterns (e.g., sequential version vs. 8-core version), ignoring the mentioned factors contributes to the percentage errors. Therefore, we apply online-learning approaches, which are widely used to learn characteristics of an application while it is running [14], to RTHpower models. Applying the online-learning approach results in Equation 12 and Equation 13, where β is computed by using few sample executions in Equation 14.

$$P_{improved} = (P^{comp||data} \times \frac{I}{\alpha} + P^{data} \times \frac{\alpha - I}{\alpha}) \times \beta \quad (12)$$

$$P_{improved} = (P^{comp||data} \times \frac{\alpha}{I} + P^{comp} \times \frac{I - \alpha}{I}) \times \beta \quad (13)$$

$$\beta = \frac{1}{1 + \overline{PE}}. \quad (14)$$

Note that parameter β for each sequential/parallel version (e.g., 1-core version or 8-core version) is fixed across problem sizes and therefore it can be obtained during kernel installation and then saved as meta-data for each version in practice. E.g., the average of matmul percentage errors is 28% for 1-core and 13% for 8-cores, therefore, the β values of *matmul* are $\frac{1}{1-28\%}$ for 1-core and $\frac{1}{1+13\%}$ for 8-cores. β is computed by using data from sample executions with two matrix sizes (i.e., 128x128, and 512x512). The test set to validate the model

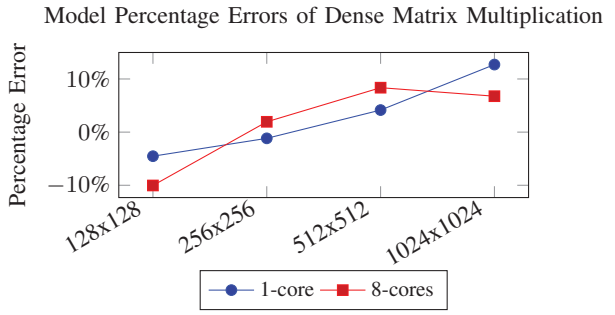


Fig. 5. Absolute percentage errors of estimated power from measured power of *matmul*. After applying the online-learning approach, the absolute percentage errors of *matmul* are at most 12%.

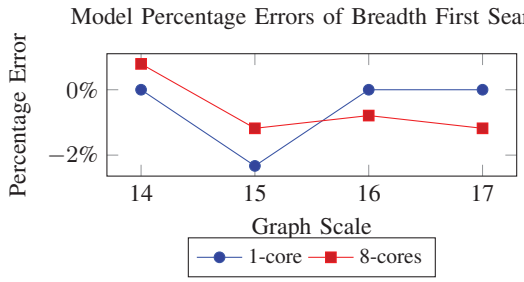


Fig. 6. Absolute percentage errors of estimated power from measured power of BFS. After applying the online-learning approach, the absolute percentage errors are at most 3%.

is data from executions with four matrix sizes (i.e., 128x128, 256x256, 512x512, and 1024x1024). The absolute percentage errors are at most 12% as shown in Figure 5.

2) Sparse Matrix Vector Multiplication: We also validate the model with SpMV implemented in Myriad platform. Due to the space constraint, we only summarize the final results. The model is validated by using the same method of *matmul* with power data from executions of six matrix sizes (i.e., 32x32, 64x64, 128x128, 256x256, 512x512, and 1024x1024). The model has the absolute percentage errors at most 4% for SpMV.

3) Breadth First Search: We implemented BFS, a data-intensive Graph500 kernel, on Myriad. The size of a graph is defined by its scale and edgefactor. In our experiments, we mostly use the default edgefactor of 16 from the Graph500 so that each vertex of the graph has 16 edges in average. The test set to validate the model is data from executions with four graph scales (i.e., 14-17) and has the absolute percentage errors at most 3% as shown in Figure 6. Both SpMV and BFS in our experiments have lower modeling errors than *matmul* since they have a fixed intensity value on different matrix sizes.

IV. Race-to-halt Prediction Framework

With RTHpower models, we want to identify whether RTH strategy is energy-efficient for an application and how many cores the system should use to run an application to achieve the least energy consumption. In order to answer the questions, we need to consider the speed-up and power-up of an application on a specific platform.

From Amdahl's Law [27] the theoretical maximum speed-up of an application running on a multicore system is derived as Equation 15, where p denotes the fraction of the application that can be parallelized and n is the number of cores:

$$\textit{speed-up} \leq \frac{1}{(1-p) + \frac{p}{n}} \tag{15}$$

A. Framework Description

The purpose of this framework is to identify when to and when not to use RTH for a given application. The two required inputs for making decision are power-up and speed-up of the application executed with n cores, where n is the maximum number of cores.

- Step 1: Identify meta-data, including speed-up and operational intensity, of a given application by one of the three main approaches listed: i) doing theoretical analysis to find the amount of computation work W, data transfer Q and operational intensity I as well as identify the maximum speed-up of a given application; ii) executing the application on a targeted platform (e.g., Myriad) to measure its speed-up and extract its operational intensity I; iii) using profiling tools [28] to extract the number of operations W and the amount of data transferred Q as well as the speed-up of an application on a common platform (e.g., Intel platform).
- Step 2: Compute power consumption of an application running with one core and with a maximum number of cores by the RTHpower models. Note that the RTHpower models are able to estimate power consumption for any number of cores by changing parameter n in the models. For verifying RTH strategy, we only need to apply the model for a single core and all cores.
- Step 3: Compare the energy consumption of the application between using one core and using a maximum number of cores to identify whether running a maximum number of cores is the most energy-efficient.

The framework is designed for kernels of libraries that will be executed several times. A kernel need to pre-run twice (i.e., with one core and with a maximum number of core) to find the speed-up while power-up is predicted by the RTHpower models. After deciding whether to use RTH for a kernel, the decision is beneficial for the remaining executions of the kernel.

B. Framework Validation

The framework is validated with three micro-benchmarks and three application kernels. In this validation, the values of operational intensity I are extracted from theoretical analysis of the implementations and speed-up is identified by executing the micro-benchmarks or application kernels with different numbers of cores.

1) Race-to-halt for Micro-benchmarks: We first validate the framework with micro-benchmarks. In this validation, we measure the power-up and speed-up of three micro-benchmarks: one with 60% parallel code, one with 100% parallel code and a small-size micro-benchmarks which has high overhead. All

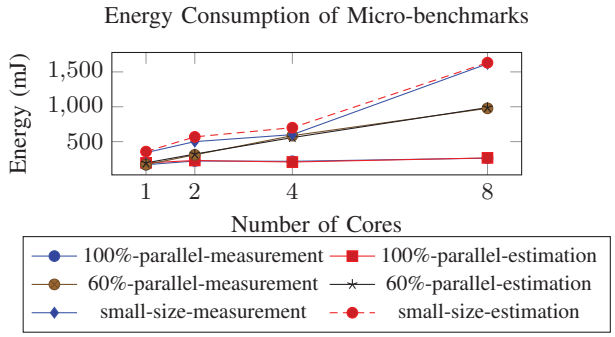


Fig. 7. Energy consumption of micro-benchmarks with operational intensity $I = 0.25$. For all three reported micro-benchmarks, the programs executed with 1 core consume the least energy, compared to 2, 4, 8 cores, from both measured data and estimated data.

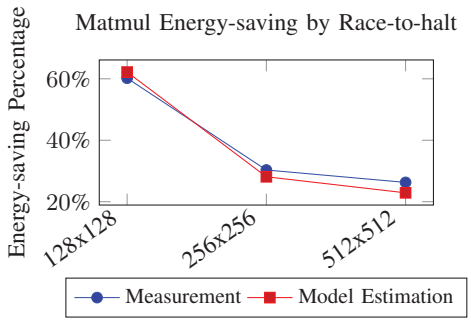


Fig. 8. *Matmul* energy-saving by Race-to-halt. This diagram shows how many percentages of energy-saving if execute *matmul* with 8 cores instead of 1 core. Since the energy-saving percentage is positive over different matrix sizes, RTH is a energy-saving strategy for *matmul*. Energy-saving percentage from model estimation for *matmul* has standard deviation less than 3%.

three micro-benchmarks have operational intensity $I = 0.25$. Namely, in the micro-benchmarks, each SauXor instruction is followed by a LsuLoad instruction which loads 4 bytes.

All three micro-benchmarks have the same assembly code wrapped inside a loop. The number of iterations to repeat the code are the difference among them. We run the micro-benchmarks on one core for 1 000 000 times. If the micro-benchmark has 100% parallel code, running it on n core requires each core performing $\frac{1}{n}$ of the amount of work (e.g., if performing the micro-benchmark on 8 cores, each core needs to run 125 000 times). Similarly, if the micro-benchmark has a parallel fraction of 60%, then running the program on n cores requires each core to perform $(1-0.6)+(\frac{0.6}{n})$ of the amount of work (e.g.,if performing the micro-benchmark on 8 cores, each core needs to run 475 000 times). For small-size micro-benchmark, the code is executed 8 times with 1 core and once with 8 cores. Since the amount of computation is small, the relative overhead of initializing the platform and executing the small-size micro-benchmark is high.

If the speed-up is bigger than the power-up, RTH is an energy-saving strategy. If the speed-up is less than the power-up, running the program with the maximum number of cores consumes more energy than running it with one core [29]. Note that when this happens, assigning one core to run the program is more energy-efficient and race-to-halt is no longer applicable for saving energy. For all three micro-benchmarks in this validation, the speed-up is identified by running them over different numbers of cores. The energy consumption of the three micro-benchmarks is shown in Figure 7. All three micro-benchmarks achieve the least energy consumption when executed with one core, from both measured and estimated data. The model estimation and actual measurement show that RTH is not applicable to the three micro-benchmarks.

2) Race-to-halt for Dense Matrix Multiplication: The *matmul* application has increasing values of operational intensity over input sizes and its speed-up is higher than its power-up on Myriad. Therefore, running *matmul* with the 8 cores is more energy-efficient than running it with one core. Figure 8 shows how many percentages of energy-saving if executing *matmul* with 8 cores instead of 1 core, from both measured and estimated data. The energy saving percentage is computed based on the energy gap of running 1 core and 8 cores divided by energy consumed by running 1 core as in Equation 16.

$$ES = \frac{E^{1core} - E^{8cores}}{E^{1core}} \tag{16}$$

The framework can predict that RTH should be applied to *matmul* over different matrix sizes. By using RTH for *matmul*, we can save from 20% to 61% of *matmul* energy consumption. RTH is a good strategy for *matmul*. We observe that the energy saving reduces when matrix size increases due to the decrease of speed-up from size 128x128. The reason is that a matrix size bigger than 128x128 makes the data set no longer fit in the last level cache (or L2 cache of 64KB) and thereby lowers performance (in flops).

3) Race-to-halt for Sparse Matrix Vector Multiplication: The framework can predict that RTH should be applied to SpMV over different matrix sizes. By using RTH for SpMV, we can save up to 59% of SpMV energy consumption.

4) Race-to-halt for Breadth First Search: In our set of application kernels implemented on Myriad, BFS is the application kernel able to prove that running with a maximum number of cores does not always give the least energy consumption. The negative values of -5% and -3% in Figure 9 mean that RTH should not be used at scale 16 and 17, respectively. The framework can predict when to apply RTH for different scales.

V. Conclusion

In this study, new fine-grained power models have been proposed to support architecture-application co-design. The models provide insights into how a given application consumes power when executing on an ultra-low power embedded system by considering both platform and application properties. The models have been validated on Movidius Myriad, an ultra-low power embedded platform with data from 35 micro-benchmarks and three application kernels. We have also shown that by using the models, we can predict whether the race-to-halt strategy (RTH) is energy-efficient for a given application on a ULP embedded system. We have presented real scenarios

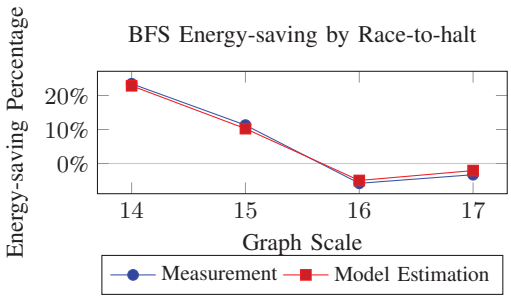


Fig. 9. BFS Energy-saving by Race-to-halt. This diagram shows how many percentages of energy-saving if executing BFS with 8 cores instead of 1 core. The positive percentages at scale 14 and 15 mean that RTH should be applied. The negative percentages at scale 16 and 17 mean that RTH should not be applied. The standard deviation of BFS energy-saving percentage is less than 3%, from scale 14 to 17.

when to use RTH and the framework based on the models could predict the scenarios precisely. Improving and applying the models and framework to other embedded platforms (e.g., ARM) and other application kernels are parts of our future work.

References

[1] I. Umar, O. J. Anshus, and P. H. Ha, "Greenbst: An energy-efficient concurrent search tree," in *Proc. of the 22nd Intl. European Conf. on Parallel and Distributed Computing (Euro-Par 16)*, p. pages to appear., 2016.

[2] I. Umar, O. J. Anshus, and P. H. Ha, "Deltatree: A locality-aware concurrent search tree," in *Proceedings of the 2015 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '15, 2015.

[3] I. Umar, O. J. Anshus, and P. H. Ha, "Effect of portable fine-grained locality on energy efficiency and performance in concurrent search trees," in *Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '16, 2016.

[4] J. Lagraviere, J. Langguth, M. Sourouri, P. H. Ha, and X. Cai, "On the performance and energy efficiency of the pgas programming model on multicore architectures," in *Proc. of the Intl. Workshop on Optimization of Energy Efficient HPC and Distributed Systems (OPTIM2016)*, 2016.

[5] H. Jacobson, A. Buyuktosunoglu, P. Bose, E. Acar, and R. Eickemeyer, "Abstraction and microarchitecture scaling in early-stage power modeling," in *IEEE 17th International Symposium on High Performance Computer Architecture (HPCA), 2011*, pp. 394–405, Feb 2011.

[6] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi, "Gpuwattch: Enabling energy optimizations in gpgpus," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ISCA '13, (New York, NY, USA), pp. 487–498, ACM, 2013.

[7] S. Li, J. H. Ahn, R. Strong, J. Brockman, D. Tullsen, and N. 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*, 2009.

[8] P. Alonso, M. F. Dolz, R. Mayo, and E. S. Quintana-Orti, "Modeling power and energy consumption of dense matrix factorizations on multicore processors," *Concurrency Computat.*, 2014.

[9] J. W. Choi, D. Bedard, R. Fowler, and R. Vuduc, "A roofline model of energy," in *Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, IPDPS '13, (Washington, DC, USA), pp. 661–672, 2013.

[10] J. Choi, M. Dukhan, X. Liu, and R. Vuduc, "Algorithmic time, energy, and power on candidate hpc compute building blocks," in *Proceedings of the 2014 IEEE 28th International Parallel and Distributed Processing Symposium*, IPDPS '14, (Washington, DC, USA), pp. 447–457, 2014.

[11] V. Korthikanti and G. Agha, "Analysis of parallel algorithms for energy conservation in scalable multicore architectures," in *International Conference on Parallel Processing, 2009. ICPP '09.*, pp. 212–219, Sept 2009.

[12] V. A. Korthikanti and G. Agha, "Towards optimizing energy costs of algorithms for shared memory architectures," in *Proceedings of the 22nd Annual ACM Symposium on Parallelism in Algorithms and Architectures*, 2010.

[13] C. Imes, D. Kim, M. Maggio, and H. Hoffmann, "Poet: a portable approach to minimizing energy under soft real-time constraints," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2015 IEEE*, pp. 75–86, April 2015.

[14] N. Mishra, H. Zhang, J. D. Lafferty, and H. Hoffmann, "A probabilistic graphical model-based approach for minimizing energy under performance constraints," in *Proceedings of the 20th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '15, ACM, 2015.

[15] D. C. Snowdon, E. Le Sueur, S. M. Petters, and G. Heiser, "Koala: A platform for os-level power management," in *Proceedings of the 4th ACM European Conference on Computer Systems*, EuroSys '09, (New York, NY, USA), pp. 289–302, ACM, 2009.

[16] M. Alioto, "Ultra-low power vlsi circuit design demystified and explained: A tutorial," *IEEE Transactions on Circuits and Systems I: Regular Papers, vol.59, no.1, pp.3-29, Jan. 2012*, vol. 59, pp. 3–29, Jan 2012.

[17] E. Le Sueur and G. Heiser, "Dynamic voltage and frequency scaling: The laws of diminishing returns," in *Proceedings of the 2010 International Conference on Power Aware Computing and Systems*, HotPower'10, (Berkeley, CA, USA), pp. 1–8, USENIX Association, 2010.

[18] M. A. Awan and S. M. Petters, "Race-to-halt energy saving strategies," *Journal of Systems Architecture*, vol. 60, no. 10, pp. 796 – 815, 2014.

[19] M. H. Ionica and D. Gregg, "The movidius myriad architecture's potential for scientific computing," *Micro, IEEE*, vol. 35, pp. 6–14, Jan 2015.

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

[21] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick, "The landscape of parallel computing research: A view from berkeley," *Technical Report No. UCB/EECS-2006-183, University of California, Berkeley*, 2006.

[22] T. Suzumura, K. Ueno, H. Sato, K. Fujisawa, and S. Matsuoka, "Performance characteristics of graph500 on large-scale distributed environment," in *IEEE International Symposium on Workload Characterization (IISWC), 2011*, pp. 149–158, 2011.

[23] G. Ofenbeck, R. Steinmann, V. Caparros, D. Spampinato, and M. Puschel, "Applying the roofline model," in *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), 2014*, pp. 76–85, March 2014.

[24] T. Takagi and M. Sugeno, "Fuzzy identification of systems and its applications to modeling and control," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. SMC-15, pp. 116–132, Jan 1985.

[25] S. Williams, L. Oliker, R. Vuduc, J. Shalf, K. Yelick, and J. Demmel, "Optimization of sparse matrix-vector multiplication on emerging multicore platforms," in *Proceedings of the 2007 ACM/IEEE Conference on Supercomputing, 2007. SC '07.*, pp. 1–12, Nov 2007.

[26] H. Cook, M. Moreto, S. Bird, K. Dao, D. A. Patterson, and K. Asanovic, "A hardware evaluation of cache partitioning to improve utilization and energy-efficiency while preserving responsiveness," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ISCA '13, 2013.

[27] M. Hill and M. Marty, "Amdahl's law in the multicore era," *Computer*, vol. 41, no. 7, pp. 33–38, 2008.

[28] X. Liu and J. Mellor-Crummey, "A tool to analyze the performance of multithreaded programs on numa architectures," in *Proceedings of the 19th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '14, 2014.

[29] V. N.-N. Tran, B. Barry, and Ha, "Rthpower: Accurate fine-grained power models for predicting race-to-halt effect on ultra-low power embedded systems," in *Proceedings of the 17th IEEE International Symposium on Performance Analysis of Systems and Software*, ISPASS 16, 2016.

Appendix B

Paper II

2016 IEEE 22nd International Conference on Parallel and Distributed Systems

ICE: A General and Validated Energy Complexity Model for Multithreaded Algorithms

Vi Ngoc-Nha Tran
Department of Computer Science
UiT The Arctic University of Norway
Tromso, Norway
Email: vi.tran@uit.no

Phuong Hoai Ha
Department of Computer Science
UiT The Arctic University of Norway
Tromso, Norway
Email: phuong.hoai.ha@uit.no

***Abstract*—Like time complexity models that have significantly contributed to the analysis and development of fast algorithms, energy complexity models for parallel algorithms are desired as crucial means to develop energy efficient algorithms for ubiquitous multicore platforms. Ideal energy complexity models should be validated on real multicore platforms and applicable to a wide range of parallel algorithms. However, existing energy complexity models for parallel algorithms are either theoretical without model validation or algorithm-specific without ability to analyze energy complexity for a wide-range of parallel algorithms.**

This paper presents a new general validated energy complexity model for parallel (multithreaded) algorithms. The new model abstracts away possible multicore platforms by their static and dynamic energy of computational operations and data access, and derives the energy complexity of a given algorithm from its *work*, *span* and *I/O* complexity. The new model is validated by different sparse matrix vector multiplication (SpMV) algorithms and dense matrix multiplication (matmul) algorithms running on high performance computing (HPC) platforms (e.g., Intel Xeon and Xeon Phi). The new energy complexity model is able to characterize and compare the energy consumption of SpMV and matmul kernels according to three aspects: different algorithms, different input matrix types and different platforms. The prediction of the new model regarding which algorithm consumes more energy with different inputs on different platforms, is confirmed by the experimental results. In order to improve the usability and accuracy of the new model for a wide range of platforms, the platform parameters of ICE model are provided for eleven platforms including HPC, accelerator and embedded platforms.

***Keywords*-power-aware and green computing; parallel algorithms; energy complexity; energy models;**

I. Introduction

Understanding the energy complexity of algorithms is crucial important to improve the energy efficiency of algorithms [16, 25–27] and reduce the energy consumption of computing systems [17, 22, 23]. One of the main approaches to understand the energy complexity of algorithms is to devise energy models.

Significant efforts have been devoted to developing power and energy models in literature [1, 6, 7, 13–15, 19, 21]. However, there are no analytic models for multithreaded algorithms that are both applicable to a wide range of algorithms and comprehensively validated yet (cf. Table I). The existing *parallel* energy models are either theoretical studies without validation or only applicable for specific algorithms. Modeling energy consumption of *parallel* algorithms is difficult since the energy models must take into account the complexity of both parallel algorithms and parallel platforms. The algorithm complexity results from parallel computation, concurrent memory accesses and inter-process communication. The platform complexity results from multicore architectures with deep memory hierarchy.

The existing models and their classification are summarized in Table I. To the best of our knowledge, the proposed ICE (Ideal Cache Energy) complexity model is the first energy model that covers all three aspects: i) ability to analyze the energy complexity of parallel algorithms (i.e. Energy complexity analysis for parallel algorithms), ii) applicability to a wide range of algorithms (i.e., Algorithm generality), and iii) model validation (i.e., Validation). The more details of related works and how the ICE model complements the other currently used models are described in the long version of this study [24].

The energy complexity model ICE proposed in this study is for general multithreaded algorithms and validated on three aspects: different algorithms for a given problem, different input types and different platforms. The proposed model is an analytic model which characterizes both algorithms (e.g., representing algorithms by their *work*, *span* and *I/O* complexity) and platforms (e.g., representing platforms by their static and dynamic energy of memory accesses and computational operations). By considering *work*, *span* and *I/O* complexity, the new ICE model is applicable to any multithreaded algorithms.

The new ICE model is designed for analyzing the energy *complexity* of algorithms and therefore the model does not provide the estimation of absolute energy consumption. The goal of the ICE model is to answer energy complexity question: *"Given two parallel algorithms A and B for a given problem, which algorithm consumes less energy analytically?"*. Hence, the details of underlying systems

1521-9097/16 $31.00 © 2016 IEEE
DOI 10.1109/ICPADS.2016.136

Table I
ENERGY MODEL SUMMARY

Study	Energy complexity analysis for parallel algorithms	Algorithm generality	Validation
LEO [19]	No	General	Yes
POET [13]	No	General	Yes
Koala [21]	No	General	Yes
Roofline [6, 7]	No	General	Yes
Energy scalability [14, 15]	Yes	General	No
Sequential energy complexity [20]	No	General	Yes
Alonso et al. [1]	Yes	Algorithm-specific	Yes
Malossi et al. [18]	Yes	Algorithm-specific	Yes
ICE model (this study)	**Yes**	**General**	**Yes**

To the best of our knowledge, the ICE model is the first *validated model that supports* energy complexity analysis for *general multi-threaded algorithms.*

(e.g., runtime and architectures) are abstracted away to keep ICE model simple and suitable for complexity analysis.

In this work, the following contributions have been made.

- Devising a new general energy model ICE for analyzing the energy complexity of a wide range of multithreaded algorithms based on their *work*, *span* and *I/O* complexity (cf. Section III). The new ICE model abstracts away possible *multicore platforms* by their static and dynamic energy of computational operations and memory access. The new ICE model complements previous energy models such as energy roofline models [6, 7] that abstract away possible *algorithms* to analyze the energy consumption of different multicore platforms.
- Conducting two case studies (i.e., SpMV and matmul) to demonstrate how to apply the ICE model to find energy complexity of parallel algorithms. The selected parallel algorithms for SpMV are three algorithms: Compressed Sparse Column(CSC), Compressed Sparse Block(CSB) and Compressed Sparse Row(CSR)(cf. Section IV). The selected parallel algorithms for matmul are two algorithms: a basic matmul algorithm and a cache-oblivious algorithm (cf. Section V).
- Validating the ICE energy complexity model with both data-intensive (i.e., SpMV) and computation-intensive (i.e., matmul) algorithms according to three aspects: different algorithms, different input types and different platforms. The results show the precise prediction on which validated SpMV algorithm (i.e., CSB or CSC) consumes more energy when using different matrix input types from Florida matrix collection [9] (cf. Section VI-A). The results also show the precise prediction on which validated matmul algorithm (i.e., basic or cache-oblivious) consumes more energy (cf. Section VI-B). The model platform-related parameters for 11 platforms, including x86, ARM and GPU, are provided to facilitate the deployment of the ICE model.

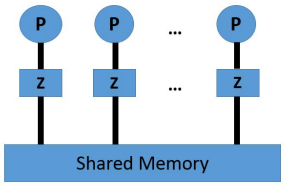


Figure 1. A shared memory model with private caches. Each core P has its own private cache of size Z and shares the (unlimited) main memory with the other cores.

II. ICE SHARED MEMORY MACHINE MODEL

Generally speaking, the energy consumption of a parallel algorithm is the sum of i) static energy (or leakage) E_{static}, ii) dynamic energy of computation E_{comp} and iii) dynamic energy of memory accesses E_{mem}. The static energy E_{static} is proportional to the execution time of the algorithm while the dynamic energy of computation and the dynamic energy of memory accesses are proportional to the number of computational operations and the number of memory accesses of the algorithm, respectively [15]. As a result, in the new ICE complexity model, the energy complexity of a multithreaded algorithm is analyzed based on its *span complexity* [8] (for the static energy), *work complexity* [8] (for the dynamic energy of computation) and *I/O complexity* (for the dynamic energy of memory accesses) (cf. Section III). This section describes shared-memory machine models supporting I/O complexity analysis for parallel algorithms.

We consider two available memory models that are used for multithreaded algorithms such as parallel external memory (PEM) model [2] and ideal distributed cache (IDC) model [11]. Both follows the memory model as described in Figure 1. However, both of the models are not applicable for analyzing I/O complexity for dynamic energy consumption [24].

In order to make our new ICE complexity model applicable to a wide range of multithreaded algorithms, we show that the cache complexity analysis using the traditional (sequential) ideal cache (IC) model [10] can be used to find an upper

bound on the cache complexity of the same algorithm using the IDC model (cf. Lemma II.1). Note that the cache complexity and the I/O complexity in the private-cache memory model (cf. Figure 1) can be used interchangeably since one cache miss results in one memory access to the shared memory. As the sequential execution of multithreaded algorithms is a valid execution regardless of whether they are divide-or-conquer algorithms, the ability to analyze the cache complexity of multithreaded algorithms via their sequential execution in the ICE complexity model improves the usability of the ICE model.

Let $Q_1(Alg,B,Z)$ and $Q_P(Alg,B,Z)$ be the cache complexity of a parallel algorithm Alg analyzed in the (uniprocessor) ideal cache (IC) model [10] with block size B and cache size Z (i.e, running Alg with a single core) and the cache complexity analyzed in the (multicore) IDC model with P cores each of which has a private cache of size Z and block size B, respectively. We have the following lemma:

Lemma II.1. *The cache complexity $Q_P(Alg,B,Z)$ of a parallel algorithm Alg analyzed in the ideal distributed cache (IDC) model with P cores is bounded from above by the product of P and the cache complexity $Q_1(Alg,B,Z)$ of the same algorithm analyzed in the ideal cache (IC) model. Namely,*

$$Q_P(Alg,B,Z) \leq P * Q_1(Alg,B,Z) \tag{1}$$

Proof: (Sketch) Let $Q_P^i(Alg,B,Z)$ be the number of cache misses incurred by core i during the parallel execution of algorithm Alg in the IDC model. Because caches do not interfere with each other in the IDC model, the number of cache misses incurred by core i when executing algorithm Alg in parallel by P cores is not greater than the number of cache misses incurred by core i when executing the whole algorithm Alg only by core i. That is,

$$Q_P^i(Alg,B,Z) \leq Q_1(Alg,B,Z) \tag{2}$$

or

$$\sum_{i=1}^{P} Q_P^i(Alg,B,Z) \leq P * Q_1(Alg,B,Z) \tag{3}$$

On the other hand, since the number of cache misses incurred by algorithm Alg when it is executed by P cores in the IDC model is the sum of the numbers of cache misses incurred by each core during the Alg execution, we have

$$Q_P(Alg,B,Z) = \sum_{i=1}^{P} Q_P^i(Alg,B,Z) \tag{4}$$

From Equations 3 and 4, we have

$$Q_P(Alg,B,Z) \leq P * Q_1(Alg,B,Z) \tag{5}$$

■

We also make the following assumptions regarding platforms.

- Algorithms are executed with the best configuration (e.g., maximum number of cores, maximum frequency) following the race-to-halt strategy.
- The I/O parallelism is bounded from above by the computation parallelism. Namely, each core can issue a memory request only if its previous memory requests have been served. Therefore, the work and span (i.e., critical path) of an algorithm represent the parallelism for both I/O and computation [8].

III. Energy Complexity in ICE model

This section describes the energy complexity model to find energy complexity of algorithms and consider both platform and algorithm characteristics. The energy complexity model considers three groups of parameters: machine-dependent, algorithm-dependent and input-dependent parameters. The reason to consider all three parameter-categories is that only operational intensity [28] is insufficient to capture the characteristics of algorithms. Two algorithms with the same values of operational intensity might consume different levels of energy. The reasons are their differences in data accessing patterns leading to performance scalability gap among them. For example, although the sequential version and parallel version of an algorithm may have the same operational intensity, they may have different energy consumption since the parallel version would have less static energy consumption because of shorter execution time.

The energy consumption of a parallel algorithm is the sum of i) static energy (or leakage) E_{static}, ii) dynamic energy of computation E_{comp} and iii) dynamic energy of memory accesses E_{mem}: $E = E_{static} + E_{comp} + E_{mem}$ [7, 14, 15]. The static energy E_{static} is the product of the execution time of the algorithm and the static power of the whole platform. The dynamic energy of computation and the dynamic energy of memory accesses are proportional to the number of computational operations $Work$ and the number of memory accesses I/O, respectively. Pipelining technique in modern architectures enables overlapping computation with memory accesses [12]. Since computation time and memory-access time can be overlapped, the execution time of the algorithm is assumed to be the maximum of computation time and memory-access time [7]. Therefore, the energy consumption of algorithms is computed by Equation 6, where the values of ICE parameters, including ε_{op}, $\varepsilon_{I/O}$, π_{op}, and $\pi_{I/O}$ are described in Table II and computed by the Equation 7, 8, 9, and 10, respectively.

$$E = P^{sta} \times max(T^{comp}, T^{mem}) + \varepsilon_{op} \times Work + \varepsilon_{I/O} \times I/O \tag{6}$$

$$\varepsilon_{op} = P^{op} \times \frac{F}{Freq} \tag{7}$$

$$\varepsilon_{I/O} = P^{I/O} \times \frac{M}{Freq} \tag{8}$$

Table II
ICE MODEL PARAMETER DESCRIPTION

Machine	Description
ε_{op}	dynamic energy of one operation (average)
$\varepsilon_{I/O}$	dynamic energy of a random memory access (1 core)
π_{op}	static energy when performing one operation
$\pi_{I/O}$	static energy of a random memory access
Algorithm	**Description**
Work	Number of work in flops of the algorithm [8]
Span	The critical path of the algorithm [8]
I/O	Number of cache line transfer of the algorithm [8]

$$\pi_{op} = P^{sta} \times \frac{F}{Freq} \tag{9}$$

$$\pi_{I/O} = P^{sta} \times \frac{M}{Freq} \tag{10}$$

The dynamic energy of one operation by one core ε_{op} is the product of the consumed power of one operation by one active core P^{op} and the time to perform one operation. Equation 7 shows how ε_{op} relates to frequency $Freq$ and the number of cycles per operation F. Similarly, the dynamic energy of a random access by one core $\varepsilon_{I/O}$ is the product of the consumed power by one active core performing one I/O (i.e., cache-line transfer) $P^{I/O}$ and the time to perform one cache line transfer computed as $M/Freq$, where M is the number of cycles per cache line transfer (cf. Equation 8). The static energy of operations π_{op} is the product of the whole platform static power P^{sta} and time per operation. The static energy of one I/O $\pi_{I/O}$ is the product of the whole platform static power and time per I/O, shown by Equation 9 and 10.

In order to compute *work*, *span* and *I/O* complexity of the algorithms, the input parameters also need to be considered. For example, SpMV algorithms consider input parameters including size of the matrix $n \times m$, the maximum number of non-zero of the sparse matrix nz, the maximum number of non-zero elements in one column nc. Cache size is captured in the ICE model by the *I/O complexity* of the algorithm. Note that in the ICE machine model (Section II), cache size Z is a constant and may disappear in the *I/O complexity* (e.g., O-notation). The details of how to obtain the ICE parameters of recent platforms are discussed in the long version of this study [24]. The actual values of ICE platform parameters for 11 recent platforms are presented in Table III.

The computation time of parallel algorithms is proportional to the span complexity of the algorithm, which is $T^{comp} = \frac{Span \times F}{Freq}$ where $Freq$ is the processor frequency, and F is the number of cycles per operation. The memory-access time of parallel algorithms in the ICE model is proportional to the I/O complexity of the algorithm divided by its I/O parallelism, which is $T^{mem} = \frac{I/O}{I/O-parallelism} \times \frac{M}{Freq}$. As I/O parallelism, which is the average number of I/O ports that the algorithm can utilize per step along the span, is bounded by the computation parallelism $\frac{Work}{Span}$, namely the average number of cores that the algorithm can utilize per step along the span (cf. Section II), the memory-access time T^{mem} becomes: $T^{mem} = \frac{I/O \times Span \times M}{Work \times Freq}$ where M is the number of cycles per cache line transfer. If an algorithm has T^{comp} greater than T^{mem}, the algorithm is a CPU-bound algorithm. Otherwise, it is a memory-bound algorithm.

1) CPU-bound Algorithms: If an algorithm has computation time T^{comp} longer than data-accessing time T^{mem} (i.e., CPU-bound algorithms), the ICE energy complexity model becomes Equation 11 which is simplified as Equation 12.

$$E = P^{sta} \times \frac{Span \times F}{Freq} + \varepsilon_{op} \times Work + \varepsilon_{I/O} \times I/O \tag{11}$$

or

$$E = \pi_{op} \times Span + \varepsilon_{op} \times Work + \varepsilon_{I/O} \times I/O \tag{12}$$

2) Memory-bound Algorithms: If an algorithm has data-accessing time longer than computation time (i.e., memory-bound algorithms): $T^{mem} \geq T^{comp}$, energy complexity becomes Equation 13 which is simplified as Equation 14.

$$E = P^{sta} \times \frac{I/O \times Span \times M}{Work \times Freq} + \varepsilon_{op} \times Work + \varepsilon_{I/O} \times I/O \tag{13}$$

or

$$E = \pi_{I/O} \times \frac{I/O \times Span}{Work} + \varepsilon_{op} \times Work + \varepsilon_{I/O} \times I/O \tag{14}$$

IV. A CASE STUDY OF SPARSE MATRIX MULTIPLICATION

SpMV is one of the most common application kernels in Berkeley dwarf list [3]. It computes a vector result y by multiplying a sparse matrix A with a dense vector x: $y = Ax$. SpMV is a data-intensive kernel and has irregular memory-access patterns. The data access patterns for SpMV is defined by its sparse matrix format and matrix input types. There are several sparse matrix formats and SpMV algorithms in literature. To name a few, they are Coordinate Format (COO), Compressed Sparse Column (CSC), Compressed Sparse Row (CSR), Compressed Sparse Block (CSB), Recursive Sparse Block (RSB), Block Compressed Sparse Row (BCSR) and so on. Three popular SpMV algorithms, namely CSC, CSB and CSR are chosen to validate the proposed energy complexity model. They have different data-accessing patterns leading to different values of I/O, work and span complexity. Since SpMV is a memory-bound application kernel, Equation 14 is applied. Due to the space constraint, only the details of computing complexity of CSB and CSC are described in the paper. CSR is the similar storage format for sparse matrices as CSC [24].

Table III
PLATFORM PARAMETER SUMMARY. THE PARAMETERS OF THE FIRST NINE PLATFORMS ARE DERIVED FROM [6] AND THE PARAMETERS OF THE TWO NEW PLATFORMS ARE FOUND IN THIS STUDY.

Platform	Processor	ε_{op}(nJ)	π_{op}(nJ)	$\varepsilon_{I/O}$(nJ)	$\pi_{I/O}$(nJ)
Nehalem i7-950	Intel i7-950	0.670	2.455	50.88	408.80
Ivy Bridge i3-3217U	Intel i3-3217U	0.024	0.591	26.75	58.99
Bobcat CPU	AMD E2-1800	0.199	3.980	27.84	387.47
Fermi GTX 580	NVIDIA GF100	0.213	0.622	32.83	45.66
Kepler GTX 680	NVIDIA GK104	0.263	0.452	27.97	26.90
Kepler GTX Titan	NVIDIA GK110	0.094	0.077	17.09	32.94
XeonPhi KNC	Intel 5110P	0.012	0.178	8.70	63.65
Cortex-A9	TI OMAP 4460	0.302	1.152	25.92	87.00
Arndale Cortex-A15	Samsung Exynos 5	0.275	1.385	24.70	89.34
Xeon	2xIntel E5-2650l v3	0.263	0.108	8.86	23.29
Xeon-Phi	Intel 31S1P	0.006	0.078	25.02	64.40

A. Compressed Sparse Column

CSC is a storage format for sparse matrices which reduces the storage of matrix compared to the tuple representation. This format compresses the sparse matrix in column-wise manner to store the non-zero elements. The *work* complexity of CSC SpMV is $\Theta(nz)$ where $nz >= n$ and *span* complexity is $O(nc+\log n)$, where nc is the maximum number of non-zero elements in a column. The *I/O* complexity of CSC in the sequential I/O model of column-major layout is $O(nz)$ [4]. Similar to CSR, scanning all non-zero elements of matrix A in CSC format costs $O(\frac{nz}{B})$ I/Os. However, randomly updating vector y causing the bottle neck with total of $O(nz)$ I/Os. Applying the proposed model on CSC SpMV, their total energy complexity are computed as Equation 15.

$$E_{CSC} = O(\varepsilon_{op} \times nz + \varepsilon_{I/O} \times nz + \pi_{I/O} \times (nc + \log n)) \quad (15)$$

B. Compressed Sparse Block

Given a sparse matrix A, while CSR has good performance on SpMV $y = Ax$, CSC has good performance on transpose sparse matrix vector multiplication $y = A^T \times x$, Compressed sparse blocks (CSB) format is efficient for computing either Ax or $A^T x$. CSB is another storage format for representing sparse matrices by dividing the matrix A and vector x, y to blocks. A block-row contains multiple chunks, each chunks contains consecutive blocks and non-zero elements of each block are stored in Z-Morton-ordered [5]. From Beluc et al. [5], CSB SpMV computing a matrix with nz non-zero elements, size $n \times n$ and divided by block size $\beta \times \beta$ has span complexity $O(\beta \times \log \frac{n}{\beta} + \frac{n}{\beta})$ and *work* complexity as $\Theta(\frac{n^2}{\beta^2} + nz)$.

I/O complexity for CSB SpMV is not available in the literature. We do the analysis of CSB manually by following the master method [8]. The *I/O* complexity is analyzed for the algorithm CSB_SpMV(A,x,y) from Beluc et al. [5]. The I/O complexity of CSB is similar to *work* complexity of CSB $O(\frac{n^2}{\beta^2} + nz)$, only that non-zero accesses in a block is divided by B: $O(\frac{n^2}{\beta^2} + \frac{nz}{B})$, where B is cache block size. The reason is that non-zero elements in a block are stored in Z-Morton order which only requires $\frac{nz}{B}$ I/Os. The energy complexity of CSB SPMV is shown in Equation 16.

From the complexity analysis of SpMV algorithms using different layouts, the complexity of CSR-SpMV, CSC-SpMV and CSB-SpMV are summarized in Table IV.

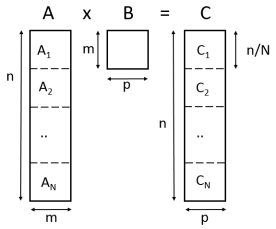


Figure 2. Partition approach for parallel matmul algorithms. Each sub-matrix A_i has size $\frac{n}{N} \times m$ and each sub-matrix C_i has size $\frac{n}{N} \times p$.

V. A CASE STUDY OF DENSE MATRIX MULTIPLICATION

Besides SpMV, we also apply the ICE model to dense matrix multiplication (matmul). Unlike SpMV, a data-intensive kernel, matmul is a computation-intensive kernel used in high performance computing. It computes output matrix C (size n x p) by multiplying two dense matrices A (size n x m) and B (size m x p): $C = A \times B$. In this work, we implemented two matmul algorithms (i.e., a basic algorithm and a cache-oblivious algorithm [10]) and apply the ICE analysis to find their energy complexity. Both algorithms partition matrix A and C equally to N sub-matrices (e.g., A_i with i=(1,2,..,N)), where N is the number of cores in the platform. The partition approach is shown in Figure 2. Each core computes a sub-matrix C_i: $C_i = A_i \times B$. Since matmul is a computation-bound application kernel, Equation 12 is applied.

A. Basic Matmul Algorithm

The basic matmul algorithm is described in Figure 3. Its work complexity is $\Theta(2nmp)$ [29] and span complexity is

$$E_{CSB} = O(\varepsilon_{op} \times (\frac{n^2}{\beta^2} + nz) + \varepsilon_{I/O} \times (\frac{n^2}{\beta^2} + \frac{nz}{B}) + \pi_{I/O} \times \frac{(\frac{n^2}{\beta^2} + \frac{nz}{B}) \times (\beta \times \log \frac{n}{\beta} + \frac{n}{\beta})}{(\frac{n^2}{\beta^2} + nz)}) \quad (16)$$

Table IV
SPMV COMPLEXITY ANALYSIS

Complexity	CSC-SpMV	CSB-SpMV	CSR-SpMV
Work	$\Theta(nz)$ [5]	$\Theta(\frac{n^2}{\beta^2} + nz)$ [5]	$\Theta(nz)$ [5]
I/O	$O(nz)$ [4]	$O(\frac{n^2}{\beta^2} + \frac{nz}{B})$ [this study]	$O(nz)$ [4]
Span	$O(nc + \log n)$ [5]	$O(\beta \times \log \frac{n}{\beta} + \frac{n}{\beta})$ [5]	$O(nr + \log n)$ [5]

```
1  for i = 1 to n
2      for j = 1 to p
3          for k = 1 to m
4              C(i,j) = C(i,j) + A(i,k) * B(k,j)
```

Figure 3. Basic matmul algorithm, where sizes of matrix A, B, C are nxm, mxp, nxp, respectively.

$\Theta(\frac{2nmp}{N})$ because the computational work is divided equally to N cores due to matrix partition approach. When matrix size of matrix B is bigger than platform cache size, the basic algorithm loads matrix B n times (i.e., once for computing each row of C), results in $\frac{nmp}{B}$ cache block transfer, where B is cache block size. In total, I/O complexity of the basic matmul algorithm is $\Theta(\frac{nm+nmp+np}{B})$. Applying the ICE model on this algorithm, the total energy complexity is computed as Equation 17.

$$E_{basic} = O(\varepsilon_{op} \times 2nmp + \varepsilon_{I/O} \times \frac{nm + nmp + np}{B} + \pi_{op} \times \frac{2nmp}{N}) \quad (17)$$

B. Cache-oblivious Matmul Algorithm

The cache-oblivious matmul (CO-matmul) algorithm [10] is a divide-and-conquer algorithm. It has work complexity the same as the basic matmul algorithm $\Theta(2nmp)$. Its span complexity is also $\Theta(\frac{2nmp}{N})$ because of the used matrix partition approach shown in Figure 2. The I/O complexity of CO-matmul, however, is different from the basic algorithm: $\Theta(n + m + p + \frac{nm+mp+np}{B} + \frac{nmp}{B\sqrt[2]{Z}})$ [10]. Applying the ICE model to CO-matmul, the total energy complexity is computed as Equation 18.

VI. VALIDATION OF ICE MODEL

This section describes the experimental study to validate the ICE model with different SpMV algorithms (i.e., CSC-SpMV and CSB-SpMV) and different matmul algorithms (Basic-Matmul and CO-Matmul) on two platforms (i.e, Xeon and Xeon Phi). We provide parameters required in the ICE model for a total of 11 platforms in Table III. We validate SpMV algorithms with nine different matrix-input types

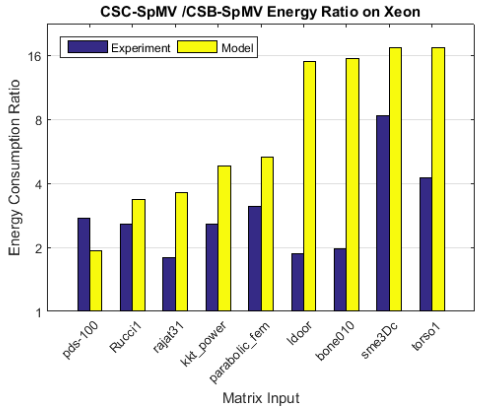


Figure 4. Energy consumption comparison between CSC-SpMV and CSB-SpMV on the Intel Xeon platform, computed by $\frac{E_{CSC}}{E_{CSB}}$. Both the ICE model estimation and experimental measurement on Intel Xeon platform show the consistent results that $\frac{E_{CSC}}{E_{CSB}}$ is greater than 1, meaning CSC SpMV algorithm consumes more energy than the CSB SpMV algorithm on different input matrices.

from Florida sparse matrix collection [9]. The details of experimental setup, how to obtain the platform parameters and input parameters are described in the long version of this study [24].

A. Validating ICE Using Different SpMV Algorithms

The model aims to compare energy consumption of two algorithm. Therefore, we validate the ICE model by showing the comparison using the ratio of energy consumption of two algorithms. From the model-estimated data, CSB SpMV consumes less energy than CSC SpMV on both platforms. Even though CSB has higher work complexity than CSC, CSB SpMV has less I/O complexity than CSC SpMV. Firstly, the dynamic energy cost of one I/O is much greater than the energy cost of one operation (i.e., $\varepsilon_{I/O} >> \varepsilon_{op}$) on both platforms. Secondly, CSB has better parallelism than CSC, computed by $\frac{Work}{Span}$, which results in shorter execution time. Both reasons contribute to the less energy consumption of CSB SpMV. The measurement data confirms that CSB SpMV

$$E_{CO} = O(\varepsilon_{op} \times 2nmp + \varepsilon_{I/O} \times (n+m+p+\frac{nm+mp+np}{B} + \frac{nmp}{B\sqrt[2]{Z}}) + \pi_{op} \times \frac{2nmp}{N}) \quad (18)$$

Table V
MATMUL COMPLEXITY ANALYSIS

Complexity	Cache-oblivious Algorithm	Basic Algorithm
Work	$\Theta(2nmp)$ [10]	$\Theta(2nmp)$ [29]
I/O	$\Theta(n+m+p+\frac{nm+mp+np}{B} + \frac{nmp}{B\sqrt[2]{Z}})$ [10]	$\Theta(\frac{nm+nmp+np}{B})$ [this study]
Span	$\Theta(\frac{2nmp}{N})$ [this study]	$\Theta(\frac{2nmp}{N})$ [this study]

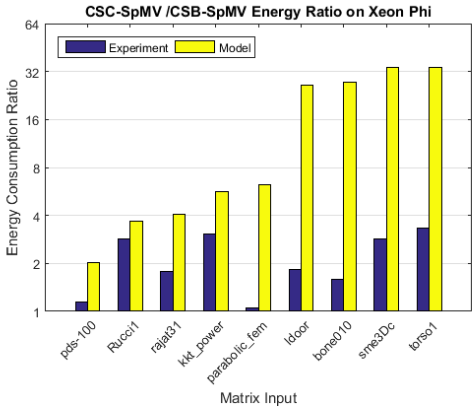


Figure 5. Energy consumption comparison between CSC-SpMV and CSB-SpMV on the Intel Xeon Phi platform, computed by $\frac{E_{CSC}}{E_{CSB}}$. Both the ICE model estimation and experimental measurement on Intel Xeon Phi platform show the consistent results that $\frac{E_{CSC}}{E_{CSB}}$ is greater than 1, meaning CSC SpMV algorithm consumes more energy than the CSB SpMV algorithm on different input matrices.

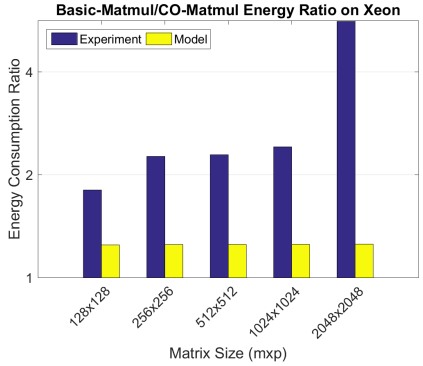


Figure 6. Energy consumption comparison between Basic-Matmul and CO-Matmul on the Intel Xeon platform, computed by $\frac{E_{Basic}}{E_{CO}}$. Both the ICE model estimation and experimental measurement on Intel Xeon platform show that $\frac{E_{Basic}}{E_{CO}}$ is greater than 1, meaning Basic-Matmul algorithm consumes more energy than the CO-Matmul algorithm.

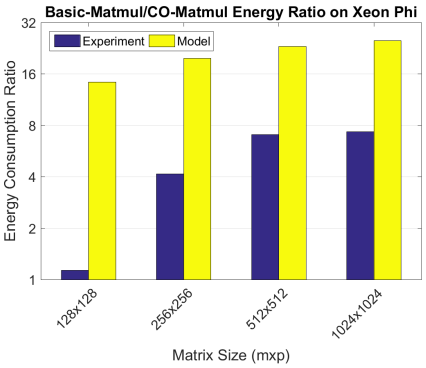


Figure 7. Energy consumption comparison between Basic-Matmul and CO-Matmul on the Intel Xeon Phi platform, computed by $\frac{E_{Basic}}{E_{CO}}$. Both the ICE model estimation and experimental measurement on Intel Xeon Phi platform show that $\frac{E_{Basic}}{E_{CO}}$ is greater than 1, meaning Basic-Matmul algorithm consumes more energy than the CO-Matmul algorithm.

algorithm consumes less energy than CSC SpMV algorithm, shown by the energy consumption ratio between CSC-SpMV and CSB-SpMV greater than 1 in the Figure 4 and 5. For all input matrices, the ICE model has confirmed that CSB SpMV consumes less energy than CSC SpMV algorithm. Because the model has abstracted possible platform by only 4 parameters (i.e., ε_{op}, $\varepsilon_{I/O}$, π_{op}, and $\pi_{I/O}$), there are the differences between the model and experiment ratios shown in the Figure 4 and 5. For accurate models that provide the precise energy estimation, the platform parameters need to be highly detailed such as RTHpower model for embedded platforms [22, 23].

B. Validating ICE With Matmul Algorithms

From the model-estimated data, Basic-Matmul consumes more energy than CO-Matmul on both platforms. Even though both algorithms have the same work and span complexity, Basic-Matmul has more I/O complexity than CO-Matmul, which results in greater energy consumption of Basic-Matmul compared to CO-Matmul algorithm. The measurement data confirms that Basic-Matmul algorithm consumes more energy than CO-Matmul algorithm, shown by the energy consumption ratio between Basic-Matmul and CO-Matmul greater than 1 in the Figure 6 and 7. For all input

matrices, the ICE model has confirmed that Basic-Matmul consumes more energy than CO-Matmul algorithm.

VII. Conclusion

In this study, we have devised a new general model for analyzing the energy complexity of multithreaded algorithms. The energy complexity of an algorithm is derived from its *work*, *span* and *I/O* complexity. Moreover, two case studies are conducted to demonstrate how to use the model to analyze the energy complexity of SpMV algorithms and matmul algorithms. The energy complexity analyses are validated for two SpMV algorithms and two matmul algorithm on two HPC platforms with different input matrices. The experimental results confirm the theoretical analysis with respect to which algorithm consumes more energy. The ICE energy complexity model gives algorithm-developers the insight into which algorithm is analytically more energy-efficient. Improving the ICE model by considering the numbers of platform cores is a part of our future work.

Acknowledgment

This research work has received funding from the European Union Seventh Framework Programme (EXCESS project, grant number 611183) and the Research Council of Norway (PREAPP project, grant number 231746/F20).

References

[1] P. Alonso, M. F. Dolz, R. Mayo, and E. S. Quintana-Orti, "Modeling power and energy consumption of dense matrix factorizations on multicore processors," *Concurrency Computat.*, 2014.

[2] L. Arge, M. T. Goodrich, M. Nelson, and N. Sitchinava, "Fundamental parallel algorithms for private-cache chip multiprocessors," in *Procs of the Twentieth Annual Symp on Parallelism in Algorithms and Architectures*, ser. SPAA '08, 2008, pp. 197–206.

[3] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick, "The landscape of parallel computing research: A view from berkeley," *Technical Report No. UCB/EECS-2006-183, University of California, Berkeley*, 2006.

[4] M. Bender, G. Brodal, R. Fagerberg, R. Jacob, and E. Vicari, "Optimal sparse matrix dense vector multiplication in the i/o model," *Theory of Computing Systems*, vol. 47, no. 4, pp. 934–962, 2010.

[5] A. Buluç, J. T. Fineman, M. Frigo, J. R. Gilbert, and C. E. Leiserson, "Parallel sparse matrix-vector and matrix-transpose-vector multiplication using compressed sparse blocks," in *Procs of the Twenty-first Annual Symp on Parallelism in Algorithms and Architectures*, ser. SPAA '09, 2009.

[6] J. Choi, M. Dukhan, X. Liu, and R. Vuduc, "Algorithmic time, energy, and power on candidate hpc compute building blocks," in *Procs of the 2014 IEEE 28th Int Parallel and Distributed Processing Symp*, ser. IPDPS '14, 2014, pp. 447–457.

[7] J. W. Choi, D. Bedard, R. Fowler, and R. Vuduc, "A roofline model of energy," in *Procs of the 2013 IEEE 27th Int Symp on Parallel and Distributed Processing*, ser. IPDPS '13, 2013, pp. 661–672.

[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*. The MIT Press, 2009.

[9] T. A. Davis and Y. Hu, "The university of florida sparse matrix collection," *ACM Trans. Math. Softw.*, vol. 38, no. 1, pp. 1:1–1:25, 2011.

[10] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran, "Cache-oblivious algorithms," in *Procs of the 40th Annual Symp on Foundations of Computer Science*, ser. FOCS, 1999.

[11] M. Frigo and V. Strumpen, "The cache complexity of multithreaded cache oblivious algorithms," in *Procs of the Eighteenth Annual ACM Symp on Parallelism in Algorithms and Architectures*, ser. SPAA '06, 2006, pp. 271–280.

[12] P. Ha, V. Tran, I. Umar, P. Tsigas, A. Gidenstam, P. Renaud-Goud, I. Walulya, and A. Atalar, "Models for energy consumption of data structures and algorithms," EU FP7 project EXCESS deliverable D2.1 (http://www.excess-project.eu), Tech. Rep., 2014.

[13] C. Imes, D. Kim, M. Maggio, and H. Hoffmann, "Poet: a portable approach to minimizing energy under soft real-time constraints," in *Real-Time and Embedded Technology and Applications Symp (RTAS), 2015 IEEE*, 2015, pp. 75–86.

[14] V. Korthikanti and G. Agha, "Analysis of parallel algorithms for energy conservation in scalable multicore architectures," in *Int Conf on Parallel Processing, 2009. ICPP '09.*, 2009, pp. 212–219.

[15] V. A. Korthikanti and G. Agha, "Towards optimizing energy costs of algorithms for shared memory architectures," in *Procs of the Twenty-second Annual ACM Symp on Parallelism in Algorithms and Architectures*, ser. SPAA '10, 2010, pp. 157–165.

[16] P. Kumar, A. Gurtov, and P. H. Ha, "An efficient authentication model in smart grid networks," in *Procs of the 15th Int Conf on Information Processing in Sensor Networks*, 2016, pp. 65:1–65:2.

[17] J. Lagravire, J. Langguth, M. Sourouri, P. H. Ha, and X. Cai, "On the performance and energy efficiency of the pgas programming model on multicore architectures," in *Procs of the Int Workshop on Optimization of Energy Efficient HPC and Distributed Systems (OPTIM), as part of the Int Conf on High Performance Computing and Simulation (HPCS)*, 2016, pp. 800–807.

[18] A. C. I. Malossi, Y. Ineichen, C. Bekas, A. Curioni, and E. S. Quintana-Orti, "Systematic derivation of time and power models for linear algebra kernels on multicore architectures," *Sustainable Computing: Informatics and Systems*, vol. 7, pp. 24 – 40, 2015.

[19] N. Mishra, H. Zhang, J. D. Lafferty, and H. Hoffmann, "A probabilistic graphical model-based approach for minimizing energy under performance constraints," in *Procs of the Twentieth Int Conf on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '15, 2015, pp. 267–281.

[20] S. Roy, A. Rudra, and A. Verma, "An energy complexity model for algorithms," in *Procs of the 4th Conf on Innovations in Theoretical Computer Science*, ser. ITCS '13, 2013, pp. 283–304.

[21] D. C. Snowdon, E. Le Sueur, S. M. Petters, and G. Heiser, "Koala: A platform for os-level power management," in *Procs of the 4th ACM European Conference on Computer Systems*, ser. EuroSys '09, 2009.

[22] V. N. N. Tran, B. Barry, and P. H. Ha, "Power models supporting energy-efficient co-design on ultra-low power embedded systems," in *Procs of the Intl Conf on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XVI), in press.*, 2016.

[23] ——, "Rthpower: Accurate fine-grained power models for predicting race-to-halt effect on ultra-low power embedded systems," in *Procs of the 17th IEEE Int Symp on Performance Analysis of Systems and Software*, 2016, pp. 155–156.

[24] V. N. Tran and P. H. Ha, "ICE: A general and validated energy complexity model for multithreaded algorithms," *CoRR*, 2016.

[25] I. Umar, O. J. Anshus, and P. H. Ha, "Deltatree: A locality-aware concurrent search tree," in *Procs of the 2015 ACM SIGMETRICS Int Conf on Measurement and Modeling of Computer Systems*, 2015, pp. 457–458.

[26] ——, "Effect of portable fine-grained locality on energy efficiency and performance in concurrent search trees," in *Procs of the 21st ACM SIGPLAN Symp on Principles and Practice of Parallel Programming*, 2016, pp. 36:1–36:2.

[27] ——, "Greenbst: An energy-efficient concurrent search tree," in *Proc. of the 22nd Intl. European Conf. on Parallel and Distributed Computing (Euro-Par 16)*, 2016, pp. 502–517.

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

[29] K. Yelick. (2004, Sept) Cs 267 parallel matrix multiplication. [Online]. Available: http://www.cs.berkeley.edu/ yelick/cs267

Appendix C

Paper III

2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)

REOH: Using Probabilistic Network for Runtime Energy Optimization of Heterogeneous Systems

Vi Ngoc-Nha Tran*, Tommy Oines†, Alexander Horsch‡, Phuong Hoai Ha§
Department of Computer Science
UiT The Arctic University of Norway
Email: *vi.tran@uit.no, †tommy.s.oines@uit.no, ‡alexander.horsch@uit.no, §phuong.hoai.ha@uit.no

***Abstract*—Significant efforts have been devoted to choosing the best configuration of a computing system to run an application energy efficiently. However, available tuning approaches mainly focus on homogeneous systems and are inextensible for heterogeneous systems which include several components (e.g., CPUs, GPUs) with different architectures.**

This study proposes a holistic tuning approach called REOH using probabilistic network to predict the most energy-efficient configuration (i.e., which platform and its setting) of a *heterogeneous* system for running a given application. Based on the computation and communication patterns from Berkeley dwarfs, we conduct experiments to devise the training set including 7074 data samples covering varying application patterns and characteristics. Validating the REOH approach on heterogeneous systems including CPUs and GPUs shows that the energy consumption by the REOH approach is close to the optimal energy consumption by the Brute Force approach while saving 17% of sampling runs compared to the previous (homogeneous) approach using probabilistic network. Based on the REOH approach, we develop an open-source energy-optimizing runtime framework for selecting an energy efficient configuration of a heterogeneous system for a given application at runtime.

I. Introduction

Improving the energy efficiency and reducing energy consumption are ones of the most important requirements of computing systems. The factors that have impacts on the application performance and its optimization strategies are algorithm design and implementation (i.e., control flow, memory types, memory access pattern and instruction count) and its execution configuration [1]. When an application runs on a heterogeneous system, one of the strategies to reduce energy consumption is to run the application with an appropriate system configuration.

Several attempts [2]–[12] have been made to find the best configurations to run an application to achieve energy efficiency. However, available tuning approaches are mostly conducted for homogeneous systems while little research considers heterogeneous systems including several platform components (e.g., CPUs and GPUs) with different types of processing units and different architectures.

Table I summarizes the related work to this study according to the four aspects: the optimization goal (i.e, Optimization), whether the optimization object is configuration or code variant (i.e., Object), whether the targeted system is homogeneous or heterogeneous (i.e., System), and whether the approach is applicable for general or specific applications (i.e., Application). Table I shows how our study is different from its related work. The goal is to optimize energy efficiency by choosing an appropriate configuration of heterogeneous systems for a given application. The details of related works can be found in the full report of this study [13].

The main goal of existing tuning approaches is to improve energy-efficiency. However, the existing models are mostly built for homogeneous systems, which has only one type of devices such as GPU [6]–[10], [12] or CPU [3]–[5]. There are also a set of studies [18]–[20] for a specific type of heterogeneous systems (i.e., APUs) but they are mainly focus on improving performance instead of energy-efficiency.

The existing heterogeneous approaches in the Table I are either for specific applications (i.e., iterative applications that can be divided to several iterations where execution time of the next iteration can be predicted based on the current iteration) [11], [16] or for finding a heterogeneous balance of datacenter [17] where the configuration at datacenter level is a mix of CPUs and microprocessors.

Among the available tuning approaches, probabilistic model-based approaches have their advantages of not requiring prior knowledge on the targeted application or the throughout understanding of system components like other approaches [8], [10]. By finding the similarity between the targeted application from sampling data and previous observed applications from training data, it can quickly provide the accurate estimation of energy consumption for the targeted application.

The previous probabilistic model based approaches only applicable for homogeneous systems (i.e., CPUs). Heterogeneous systems have complex structures containing different platform architectures (e.g., CPUs, GPUs, FPGAs, ASICs) where each platform has its own sets of settings and methods to change its configurations. Applying the probabilistic model based approach [5] on each individual platform of a heterogeneous system requires the analysis of the available settings and a new configuration data for each platform. In the other words, it requires separated sets of training and sampling data, and separated runs of prediction for each platform. This results in more sampling runs than doing one prediction for a heterogeneous system with only one whole set of training and sampling data. Therefore, the probabilistic model based approaches for heterogeneous systems requires the analysis of the available settings of all included platforms within a heterogeneous system and finding the setting equivalence of one platform to another platform.

978-1-5386-7308-9/18/$31.00 ©2018 IEEE
DOI 10.1109/ICPADS.2018.00058

TABLE I
AUTO-TUNING FRAMEWORK

Study	Optimization	Object	System	Application
OSKI [14]	Time	Code variant	Homogeneous (i.e., CPU)	Specific (i.e., Sparse kernels)
Nitro [15]	Time	Code variant	Homogeneous (i.e., GPU)	General
PowerCap [3]	Timeliness Energy-efficiency	Configuration	Homogeneous (i.e., CPU)	General
POET [4]	Energy-efficiency	Configuration	Homogeneous (i.e., CPU)	General
LEO [5]	Time Energy-efficiency	Configuration	Homogeneous (i.e., CPU)	General
HPC runtime framework [6]	Energy-efficiency	Configuration	Homogeneous (i.e., CPU)	General
GPU models [7]	Power	Configuration	Homogeneous (i.e., GPU)	General
CRISP [8]	Energy	Configuration	Homogeneous (i.e., GPGPU)	General
MPC [9]	Energy-efficiency	Configuration	Homogeneous (e.g., GPGPU)	General
GreenGPU [11], [16]	Energy-efficiency	Workload division Frequency	Heterogeneous (e.g., CPU and GPU)	Specific (i.e., Iterative applications)
GPGPU DVFS models [10]	Energy-efficiency	Configuration	Homogeneous (i.e., GPGPU)	General
GPGPU SVR models [12]	Energy-efficiency	Configuration	Homogeneous (i.e., GPGPU)	General
Market mechanism [17]	Service quality Energy-efficiency	High-level configurations (i.e., Datacenters)	Heterogeneous (e.g., CPUs and microprocessors)	General
REOH (this study)	Energy-efficiency	Configuration	Heterogeneous (e.g., CPU and GPU)	General

In this study, we propose a way to unify the configurations of different platforms on a heterogeneous system and do the prediction only once. This way we save energy of the sampling runs. Even though we evaluate the probabilistic model-based approach (i.e., REOH) on a system containing CPU and GPU only, REOH is general for heterogeneous systems which contain any architectures (e.g., CPUs, GPUs, FPGAs, ASICS) where we can identify and change their configurations (i.e., the combination of number of cores, memory and frequency) in runtime.

The proposed approach aim to address the following research question: *"Given executable files of an application and a heterogeneous system containing platforms with different architecture, which system configuration (i.e., platform and its setting) to run the application most energy-efficiently?"*

This study propose holistic tuning approach based on probabilistic model to predict the most energy-efficient configuration of heterogeneous systems for a given application. Based on the application communication and computation patterns (i.e., Berkeley dwarfs [21], we choose the Rodinia benchmarks [22] for the experiments and devise a training data set. The objectives when choosing the benchmarks are to devise a training data set that cover a wide range of application patterns and characteristics.

We also provide an open-source energy-optimizing runtime framework to choose which configuration of a heterogeneous system to run a given application at runtime. Even though the open-source is for the experimented system including only one CPU and one GPU, the code is available and can be adjusted to heterogeneous systems containing other types of platforms as long as changing platform configurations during runtime is supported.

This study is for applications that runs on one platform (e.g., CPU or GPU) at a time. The application has different executable files for different platforms (e.g., CPU or GPU) that can be chosen during runtime. For example, Rodinia benchmarks suite [22] supports programming models such as OpenCL which can provide different executable files of the same benchmark. This approach, however, can also apply to applications that can be divided to several phases. Each phase is wrapped in an executable file and can be considered as one application in REOH approach. Therefore, each phase of such applications only runs on one platform but the whole execution with different phases runs on several platforms.

In this work, the following contributions have been made.

- Devise a new holistic tuning approach for heterogeneous systems using probabilistic network, which is called REOH. In this study, we propose a method to unify the configurations of different platform types (e.g., CPU and GPU), consider the total energy of both static and dynamic energy and devise a training data set containing 7074 samples by running a selected set of 18 applications based on the knowledge of application patterns from Berkeley dwarfs on a total of 393 system configurations.
- Validate the REOH approach on a heterogeneous system consisting of CPU and GPU, showing that REOH approach achieves the close energy consumption (i.e., within 5% different) to the optimal energy consumption by the brute-force approach when choosing the most energy-efficient system configuration for the applications while saving 17% number of sampling runs than the existing probabilistic network approaches [5].
- Develop an open-source energy-optimizing runtime framework for selecting an energy efficient configuration of a heterogeneous system for a given application at runtime. The framework takes as the input the executable files that the users want to run on a targeted heterogeneous system. Then the framework will choose an appropriate configuration of the targeted heterogeneous system to run the executable files energy-efficiently. This tool is provided as an open source for scientific research purposes.

The content of the paper is organized as follows. Section II describes REOH, the energy optimization approach for heterogeneous systems. In Section III, we validate the approach on a heterogeneous system consisting og CPUs and GPUs. Based on the proposed energy optimization approach, Section IV describe the energy-optimizing runtime framework and its implementation. Section V concludes the study.

II. A Holistic Tuning Approach for Heterogeneous Systems

This section describes REOH, the holistic tuning approach enhanced for heterogeneous systems using the probabilistic graphical model-based approach [5].

A. Unifying platform configurations

Unlike the previous (homogeneous) probabilistic graphical models approach [5], the REOH approach proposed in this study is for heterogeneous systems including different platforms with different architectures. The probabilistic modeling approach requires experimental data from a set of configurations that can be tuned during runtime.

The configurations must be pre-defined and provided in training data. For REOH, the configurations are the combination of the number of cores, the core frequency and the number of memory controllers. An example of CPU configuration is 24 cores running at frequency 1.7 GHz with two memory channels. Each platform architecture has its own hardware specification with different numbers of cores, the core frequencies or memory controllers [5]. For heterogeneous systems including several platforms with different architectures, in order to apply the probabilistic approach, finding the equivalence of configurations from different platforms is essential.

In this section, we propose a methodology to convert the configurations of different platforms. We consider the peak compute flops and peak memory bandwidth when finding the equivalence of the configurations of different platforms. The study by Lee et.al. [23] provided a comparison of CPU and GPU performance on 14 kernels considering architectural differences such as processing element (or PE) and bandwidth differences. The average performance (in flops) of each processing element is computed by dividing the platform computing flops by the total number of processing elements in the platform: $Flops_{PE} = \frac{PeakFlops}{TotalPE}$. In the context of this study, the total processing elements are the number of cores available in the platform. E.g., $Flops_{CPUcore} = \frac{PeakFlops_{CPU}}{TotalCores_{CPU}}$ and $Flops_{GPUcore} = \frac{PeakFlops_{GPU}}{TotalCores_{GPU}}$.

Therefore, to unify the number of cores in GPU (or $nGPUcore$) with a equivalent number of cores in CPU (or $nCPUcore$), we compare performance of CPU cores and GPU cores as in Equation 1:

$$\begin{aligned} nGPUcore &= \frac{Flops_{GPUcore}}{Flops_{CPUcore}} \times nCPUcore \\ &= \frac{PeakFlops_{GPU}}{TotalCores_{GPU}} \times \frac{TotalCores_{CPU}}{PeakFlops_{CPU}} \times nCPUcore \end{aligned} \quad (1)$$

In our heterogeneous system, there are two platforms: CPU Xeon E5-2650Lv3 has 24 cores and peak performance as 115.2 GFlops while GPU Nvidia Quadro K620 has 384 cores with peak performance as 860 GFlops. The average performance for a CPU core is $\frac{115.2}{24} = 4.8$ GFlops while the average performance for a GPU core is $\frac{860}{384} = 2.24$ Gflops. One GPU core is equivalent to $\frac{24}{115.2} * \frac{860}{384} = 0.47$ CPU core, which is approximately half of the performance of one CPU core. Therefore, one GPU core is approximately equivalent to 0.5 CPU core.

Similarly, we convert the number of memory controllers of GPU (or $nGPUmem$) to the number of memory controllers in CPU (or $nCPUmem$) based on peak memory bandwidth of CPU and GPU as in Equation 2. CPU Xeon E5-2650L and

GPU Nvidia Quadro K620 has a peak bandwidth 68 GB/s and 28.8 GB/s respectively. Both CPU and GPU platforms have two memory controllers. The bandwidth of one memory controller of GPU ($GB_{GPUcore}$) is equivalent to $\frac{28.8}{68}$ CPU counterpart, which is approximately half of the bandwidth of a CPU memory controller.

$$nGPUmem = \frac{GB_{GPUcore}}{GB_{CPUcore}} \times nCPUmem \quad (2)$$

The frequencies in REOH approach are represented by integer numbers as indexes. The increasing order of frequency indexes reflects the increasing oder of frequency values. For example, the experimented CPU has 8 frequencies (i.e., 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.81GHz) represented by the numbers (i.e., 0, 1, 2, 3, 4, 5, 7, 8, respectively). The experimented GPU has one frequency (i.e., 1.73 GHz) represented by the number 6.

B. Total energy consumption of heterogeneous systems

In the REOH holistic approach, we target to optimize the total energy consumption of heterogeneous systems, including both static (idle) and dynamic energy of every platform in the system while the existing (homogeneous) approaches only consider the energy consumption of individual platform in isolation.

Unlike the homogeneous approach that considers CPU energy and GPU energy in isolation, the holistic approach considers CPU energy and GPU energy together. It is because although the application runs on GPU (resp. CPU), idle CPU (resp. GPU) consumes energy as well (i.e., static energy). This is one of the reasons that makes the most energy efficient configurations from homogeneous approaches not always the most energy efficient configurations in heterogeneous systems. Figure 1 shows the optimal dynamic energy of CPU and GPU while 2 shows the optimal total energy including static energy of the idle platform and dynamic energy of the running platform. The optimal configurations for each application from the two sets of data (i.e., the dynamic energy data and the total energy data) are not always the same. For example, from the dynamic energy data, running application 17 on GPU consumes less energy than running it on CPU while from the total energy data, running application 17 on CPU is more energy-efficient than on GPU.

The research question that the REOH approach wants to address is: which platform (CPU or GPU), together with its configuration, in a heterogeneous system is the most energy efficient for executing a given application. In our research context, when an application is executed by ones of the platforms (e.g., active platforms), the other platforms are in idle mode. The energy consumption of the active platforms includes their static and dynamic energy while the energy consumption of the idle platforms includes only their static energy. The total energy consumption of a whole heterogeneous system includes not only the energy of active platforms but also the energy of idle platforms as Equation 3. The energy consumption of active platforms includes static and dynamic energy while the energy consumption of idle platforms is the static energy. In Equation 3, the heterogeneous system has m platforms. The active platforms are platforms (1, 2, .., n) and the idle platforms are platforms (n+1, n+2, .., m).

$$E^{total} = \sum_{i=1}^{n}(E_i^{static} + E_i^{dynamic}) + \sum_{j=n+1}^{m} E_j^{static} \quad (3)$$

In our heterogeneous system used for validating the REOH approach, there are two platforms CPU and GPU. If an application is run on CPU while GPU is idle, the total energy is computed as $E_{CPU}^{total} = E_{CPU}^{static} + E_{CPU}^{dynamic} + E_{GPU}^{static}$. If an application is run on GPU while CPU is idle, the total energy is computed as $E_{GPU}^{total} = E_{GPU}^{static} + E_{GPU}^{dynamic} + E_{CPU}^{static}$. This is one of the improvements of REOH holistic approach compared to the existing (homogeneous) approaches.

TABLE II
APPLICATION CATEGORIES BASED ON DWARF LIST

Dwarf	**Performance Limit [21]**	**Benchmark [22]**
Graph Traversal	Memory Latency	B+Tree BFS
Structured Grid	Memory Bandwidth	HeartWall Particle Filter
Unstructured Grid	Memory Latency	CFD Solver Back Propagation
Dense linear algebra	Computation	LUD kmeans
Sparse Matrix	50%Computation 50% Memory Bandwidth (cf. Figure 9 in [21])	
Dynamic Programming	Memory Latency	Path Finder Needleman-Wunsch
N-body	Computation	LAVAMD
Spectral	Memory Latency	GPUDWT

C. Application categories

We propose a selected set of applications for experimenting and devising a general training data set which can cover a wide range of communication and computation patterns. A training data set obtained offline is required by the probabilistic network approach. The main objectives of the training data set is to represent the wide range of computation and communication patterns and characteristics. In order to identify such varied set of patterns, we consider the pattern categories based on Berkeley dwarfs [21] and its corresponding benchmarks in the Rodinia benchmark suite [22].

We summarize the dwarf list and their corresponded benchmarks based on their categories and characteristics as in Table II. Each of the dwarfs has performance limit due to computation, memory bandwidth or memory latency as shown in the second column (e.g., Performance Limit). The third column shows the benchmarks belonging to the dwarf.

There are several impact factors that affect the application performance and its optimization strategies such as algorithm

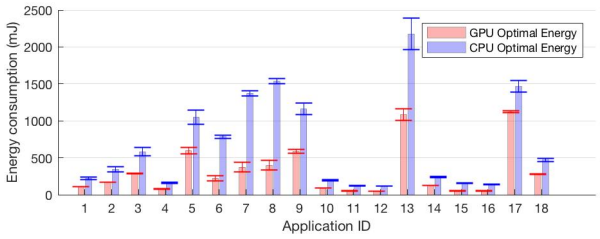


Fig. 1. Optimized energy consumption of CPU and GPU from homogeneous approach

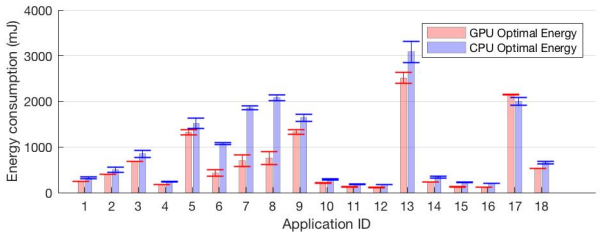


Fig. 2. Optimized energy consumption of CPU and GPU from the heterogeneous approach, which considers both static and dynamic energy of each platform

design, execution configuration, control flow, memory types, memory access pattern and instruction count [1]. These factors are represented by three categories of performance limits: computation, memory bandwidth and memory latency [21]. In order to select the benchmarks that represent a wide range of application behaviors, we choose a set of benchmarks that cover all three categories of the performance limits such as Kmeans, BFS, Particle Filter and CFD. The four benchmarks belong to the first four dwarfs in Table II.

We chose Rodinia [22] benchmarks to validate our approach because it provides implementations for a variety of platforms (e.g., CPU and GPU) and programming models (e.g., OpenCL, CUDA, OpenMP). Among the supported programming models of Rodinia, OpenCL implementations are selected since OpenCL library is supported on a various architectures such as CPU, GPU and accelerators.

Moreover, the problem size can also impact the benchmark performance and its optimization strategy [1], [24]. For each chosen benchmarks, we also select a set of input that covers a varying range of benchmark patterns.

The selected input was generated using the data generators from Rodinia, in which the sample sizes were chosen to grow exponentially to cover various range of input sizes. BFS has input graphs with sizes varying from 512kB to 8MB. CFD experiments are conducted with only three input sizes due to the unavailability of input generator and limited input provided by Rodinia. Kmeans has the input generating from two parameters: the number of objects and the number of features. For instance, in Table III, the input name 1000_34 means there are 1000 objects and each object has 34 features [25]. Particle Filter has the input generating from three parameters as its three dimensions. For instance, the input name 128_10_1000_dp means that the input dimensions is 128x128x10 with 1000 particles and particles are double type [26]. For each input size and configuration, each benchmark is performed five times and the measurement of average and deviation values are stored in training data set.

TABLE III
APPLICATION DETAILS

Application ID	Benchmark	Input
1	BFS	graph1M
2	BFS	graph2M
3	BFS	graph4M
4	BFS	graph512k
5	BFS	graph8M
6	CFD	fvcorr.domn.097K
7	CFD	fvcorr.domn.193K
8	CFD	missile.domn.0.2M
9	Kmeans	1000000_34
10	Kmeans	100000_34
11	Kmeans	10000_34
12	Kmeans	1000_34
13	Kmeans	3000000_34
14	ParticleFilter	128_10_100000_dp
15	ParticleFilter	128_10_10000_dp
16	ParticleFilter	128_10_1000_dp
17	ParticleFilter	128_2500_10000_dp
18	ParticleFilter	128_500_10000_dp

III. Energy Saving - Experimental Results

In this section, we validate the REOH approach by experimental study: how close to the optimal configuration (by the brute-force approach) the configuration by the REOH approach is. The optimal configuration means the best platform and its best setting in term of energy consumption. The REOH approach predicts the best configurations (i.e., the best

platform and its best setting in term of energy consumption) based on the training data and sampling data.

A. Devise training data and sampling data

The training data was devised by conducting the experiments to measures energy consumption of 18 applications (each application is a combination of a benchmark and an input) on all available configurations of two platforms (i.e., 384 configurations of CPU and 9 configurations of GPU) in the targeted heterogeneous system (cf. Table III). The 384 configurations of CPU are the combination of 24 cores, 8 frequencies and 2 memory controllers. The CPU configurations (i.e., the combinations of cores, frequencies, memory controllers) are set by using *cpufrequtils* package and *numactl* library. The 9 configurations of GPU are the workgroup sizes assigned to applications, such as 1, 2, 4, 8, 16, 32, 64, 128, 256 work units, which affect the occupancy and the number of active multiprocessors of GPU.Time and energy measurement were performed with MeterPU [27] library using Intel PCM for CPU and Nvidia NVML for GPU. Each application was run five times for each configuration and the mean and standard deviation values of measured performance and consumed energy are stored. Note that the minimum number of cores (respectively memory controller) is one in order to ensure that the application always completes in a finite amount of time.

The sampling data is obtained by running a given application with sample configurations and measuring its performance time and energy consumption. In our validation, sample configurations are chosen randomly.

B. Approach validation

Based on the training data and sampling data, the probabilistic model is applied to estimate the energy consumption of the remaining configurations (namely, all possible configurations except for sample configurations). Noted that when sampling an application A, A's data is removed from the training data set. From the estimated energy consumption of all configurations, the best configuration which consumes the least energy is selected.

We compares the result of the REOH approach with the LEO approach [5], the state-of-the-art (homogeneous) approach based on a similar probabilistic model. REOH approach is applied on a heterogeneous system with both CPU and GPU data while LEO approach is applied on homogeneous system (i.e., either on CPU platform with CPU data or GPU platform with GPU data). The details (i.e., data from which platform and data size) of training and sampling set for each approach are summarized in Table IV.

The probabilistic approach uses regression diagnostics (i.e., regstats function) [28] with full quadratic [29] as an input model. For REOH and LEO-CPU prediction, the regstats function has 3 predictors (i.e., the number of cores, the frequency index and the number of memory controllers) which creates 10 (i.e., $\frac{(3+1)\times(3+2)}{2}$) predictor variables [29]. The model for REOH and LEO-CPU, therefore, requires at least 10 observations (i.e., the number of sampling data). Since the considered GPU has less than 10 configurations, we only use one predictor (i.e., workgroup size) for the regression function when applying the probabilistic approach for GPU platform with GPU data only. The model for LEO-GPU requires at least 3 sampling data.

TABLE IV
TRAINING AND SAMPLING DATA FOR EACH APPROACH

	Training Data		Sampling Data	
Approach	**Platform**	**Size**	**Platform**	**Size**
LEO-CPU	CPU	384x18	CPU	15x1
LEO-GPU	GPU	9x18	GPU	3x1
REOH	CPU+GPU	393x18	CPU	15x1

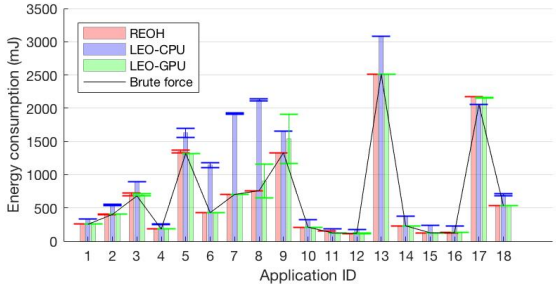


Fig. 3. Energy comparison of the four approaches: REOH, LEO-CPU, LEO-GPU and Brute Force

The prediction was performed with the total number of samples varying from 10 (the minimum samples requirement) to 50 samples. The accuracy of the model increases when the number of sample increases to 15. After reaching 15 samples, the accuracy of the model does not significantly changed when taking more samples. Therefore, we choose to sampling 15 data on 15 configurations when performing model prediction with REOH and LEO-CPU approach. For LEO-GPU, we choose the number of sampling data as 3.

In this validation, we compare the most energy-efficient configuration by the REOH approach for a heterogeneous system containing a CPU and a GPU to the most energy-efficient configurations by the LEO approach for a homogeneous system with a CPU platform and the most energy-efficient configurations by the LEO approach for a homogeneous system with a GPU platform. Moreover, we also compare the REOH results with the optimal results by the brute-force approach that has all measured data of all platforms (i.e., CPU and GPU) available. The brute-force approach always choose the optimal configuration. Figure 3 shows the energy consumption (in mJ) of the configurations selected by the four approaches for 18 applications and Figure 4 shows the energy consumption difference between the three approaches (LEO-CPU, LEO-GPU [5] and REOH) and the Brute-force approach. The list of applications and their ID are summarized in Table III.

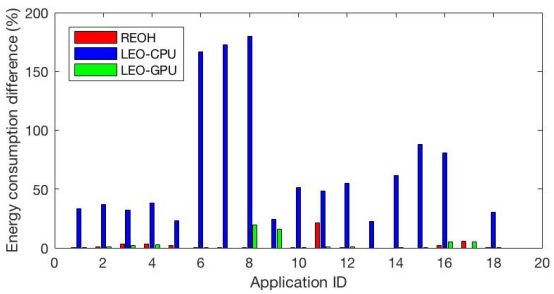


Fig. 4. Percentage of the differences on energy consumption of REOH, LEO-CPU and LEO-GPU approach compared to Brute Force approach

The results shows that for 17 out of 18 applications, the the REOH approach predicts the close results to LEO-GPU approach and the Brute Force approach (up to 0.9% more energy consumption to LEO-GPU and within 5.7% deviation to Brute Force) except application 11. Unlike other applications where the performance increases when the number of cores increases, application 11 has the performance increased in the first 12 cores and decreased in the second 12 cores as shown in its experimental data (note that the platform has two 12-core CPUs). Application 11 has a different performance pattern than other applications which leads to the less precise prediction of REOH on application 11.

REOH also predicts better results than LEO-CPU except application 17. LEO-CPU approach has better prediction only on the application 17: 5.7% less energy consumption than the REOH approach. Application 17 has the best configuration on the CPU platform and the LEO-CPU approach, which considers only CPU data, is expected to be more accurate. However, its energy difference on the CPU platform between LEO-CPU and REOH approaches is marginal. Even though REOH approach predicts a configuration with higher energy consumption than LEO-CPU approach at application 17, its energy consumption is also within 5.7% of the optimal energy consumption by the brute-force approach (cf. Figure 4).

The results have confirmed that the REOH approach can use the training set from selected applications to predict competitive configurations (within 5.7% of the optimal in 17 applications) in term of energy consumption. Moreover, the REOH approach only needs 15 samples from CPU data to predict the most energy-efficient configuration while LEO requires two predictions on data from two separate platforms, either CPU or GPU data. The total number of samples when using LEO approach is $15 + 3 = 18$, which is 20% more sampling numbers as compared to REOH approach. By using REOH approach, the system is beneficial in two ways: not sampling GPU data and save 17% (i.e., $\frac{3}{15+3}$) the number of sampling runs.

IV. Energy-optimizing Runtime Framework

Based on the new REOH approach, an open-source runtime framework has been developed to provide users with an energy-efficient system configuration for a given executable running on a heterogeneous system. The framework is publicly available at: https://github.com/uit-agc/REOH.

A. Framework design

Figure 5 shows an overview of our framework. The implementation details can be found in the full report of this study [13].

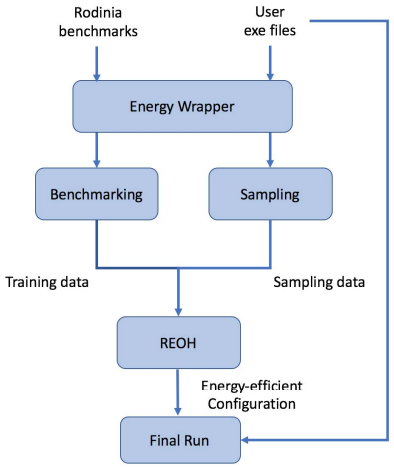


Fig. 5. Prototype Overview

a) Energy Wrapper: The energy wrapper consists of an executable that is responsible for setting platform configurations and measuring energy and execution time of a given application. Each application is provided with two executables: one for the CPU platform and one for the GPU platform, assuming the the underlying heterogeneous system consists of CPU and GPU platforms. Time and energy measurement were performed using MeterPU [27], instantiated with Intel PCM for CPU and Nvidia NVML for GPU. The executables are executed using the POSIX `system()` command.

b) Benchmarking: The module is to obtain the training data for a given heterogeneous system by executing the energy wrapper module over all 18 applications (cf. Table III) for all system configurations. This step only needs to perform once for different workloads.

c) Sampling: The sampling is performed by executing the energy-wrapper for user executables on sample configurations. This module is to provide the sampling data in order to estimate the energy consumption of the executables on all configurations. This step is performed for every given application and its executables from users.

The output data of both the Benchmarking and Sampling module is converted to the appropriate format using the scripts provided in this framework. During transformation, we also add static energy consumed by CPU and GPU. The static energy were measured by recording the energy measurements over 20 seconds for each platform using MeterPU [27]. This was done once to measure the the static power of each platform in the heterogeneous system. The static power are stored for later use.

d) REOH: The energy-optimizing module estimates the energy consumption of all configurations of the heterogeneous system based on the training data set and sampling data set. Then it provides a appropriate energy-efficient configuration to run the given application.

e) Final Run: From the configuration provided by REOH module, the Final Run module runs the appropriate executable file (e.g., executable file for CPU or GPU) on the provided configuration and measure its energy consumption.

V. Conclusion

This study has proposed and validated REOH, a new holistic approach using probabilistic model to predict and select the optimal configurations in term of energy consumption of heterogeneous systems for a given application. This study has demonstrated that REOH can achieve almost optimal energy consumption (within 5.7% of the optimal energy consumption by the brute-force approach) while saving the energy consumption of 17% less sample runs. Based on the REOH approach, a runtime framework for executing given executables energy-efficiently is developed and provided as open source software for scientific purposes.

References

[1] W.-c. Feng, H. Lin, T. Scogland, and J. Zhang, "Opencl and the 13 dwarfs: A work in progress," in *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, ICPE '12, pp. 291–294, ACM, 2012.

[2] A. Magni, C. Dubach, and M. F. P. O'Boyle, "A large-scale cross-architecture evaluation of thread-coarsening," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, pp. 11:1–11:11, 2013.

[3] H. Zhang and H. Hoffmann, "Maximizing performance under a power cap: A comparison of hardware, software, and hybrid techniques," in *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '16, pp. 545–559, 2016.

[4] C. Imes, D. H. K. Kim, M. Maggio, and H. Hoffmann, "Poet: a portable approach to minimizing energy under soft real-time constraints," in *21st IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 75–86, April 2015.

[5] N. Mishra, H. Zhang, J. D. Lafferty, and H. Hoffmann, "A probabilistic graphical model-based approach for minimizing energy under performance constraints," in *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2015.

[6] G. L. T. Chetsa, L. Lefrvre, J. M. Pierson, P. Stolf, and G. D. Costa, "A runtime framework for energy efficient hpc systems without a priori knowledge of applications," in *2012 IEEE 18th International Conference on Parallel and Distributed Systems*, pp. 660–667, Dec 2012.

[7] Y. Abe, H. Sasaki, S. Kato, K. Inoue, M. Edahiro, and M. Peres, "Power and performance characterization and modeling of gpu-accelerated systems," in *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, pp. 113–122, May 2014.

[8] R. Nath and D. Tullsen, "The crisp performance model for dynamic voltage and frequency scaling in a gpgpu," in *Proceedings of the 48th International Symposium on Microarchitecture*, MICRO-48, pp. 281–293, ACM, 2015.

[9] A. Majumdar, L. Piga, I. Paul, J. L. Greathouse, W. Huang, and D. H. Albonesi, "Dynamic gpgpu power management using adaptive model predictive control," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 613–624, Feb 2017.

[10] J. Guerreiro, A. Ilic, N. Roma, and P. Tomas, "Gpgpu power modeling for multi-domain voltage-frequency scaling," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 789–800, Feb 2018.

[11] K. Ma, X. Li, W. Chen, C. Zhang, and X. Wang, "Greengpu: A holistic approach to energy efficiency in gpu-cpu heterogeneous architectures," in *2012 41st International Conference on Parallel Processing*, pp. 48–57, Sept 2012.

[12] Q. Wang and X. Chu, "Gpgpu power estimation with core and memory frequency scaling," *SIGMETRICS Perform. Eval. Rev.*, vol. 45, pp. 73–78, Oct. 2017.

[13] V. N. Tran, T. Oines, A. Horsch, and P. H. Ha, "REOH: using probabilistic network for runtime energy optimization of heterogeneous systems," *CoRR*, vol. abs/1801.10263, 2018.

[14] R. Vuduc, J. W. Demmel, and K. A. Yelick, "Oski: A library of automatically tuned sparse matrix kernels," in *Institute of Physics Publishing*, 2005.

[15] S. Muralidharan, M. Shantharam, M. Hall, M. Garland, and B. Catanzaro, "Nitro: A framework for adaptive code variant tuning," in *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, pp. 501–512, May 2014.

[16] K. Ma, Y. Bai, X. Wang, W. Chen, and X. Li, "Energy conservation for gpucpu architectures with dynamic workload division and frequency scaling," *Sustainable Computing: Informatics and Systems*, vol. 12, pp. 21 – 33, 2016.

[17] M. Guevara, B. Lubin, and B. C. Lee, "Market mechanisms for managing datacenters with heterogeneous microarchitectures," *ACM Trans. Comput. Syst.*, vol. 32, pp. 3:1–3:31, Feb. 2014.

[18] J. Shen, A. L. Varbanescu, Y. Lu, P. Zou, and H. Sips, "Workload partitioning for accelerating applications on heterogeneous platforms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, pp. 2766–2780, Sept 2016.

[19] F. Zhang, J. Zhai, B. He, S. Zhang, and W. Chen, "Understanding co-running behaviors on integrated cpu/gpu architectures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, pp. 905–918, March 2017.

[20] F. Zhang, B. Wu, J. Zhai, B. He, and W. Chen, "Finepar: Irregularity-aware fine-grained workload partitioning on integrated architectures," in *2017 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pp. 27–38, Feb 2017.

[21] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick, "The landscape of parallel computing research: A view from berkeley," *Technical Report No. UCB/EECS-2006-183, University of California, Berkeley*, 2006.

[22] "Rodinia:accelerating compute-intensive applications with accelerators - http://www.cs.virginia.edu/ skadron/wiki/rodinia/index.php/ rodinia:accelerating_compute-intensive_applications_with_accelerators."

[23] V. W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. D. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. Hammarlund, R. Singhal, and P. Dubey, "Debunking the 100x gpu vs. cpu myth: An evaluation of throughput computing on cpu and gpu," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ISCA '10, pp. 451–460, ACM, 2010.

[24] V. N. N. Tran and P. H. Ha, "Ice: A general and validated energy complexity model for multithreaded algorithms," in *2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 1041–1048, Dec 2016.

[25] N. Ailon, R. Jaiswal, and C. Monteleoni, "Streaming k-means approximation," in *Advances in Neural Information Processing Systems 22* (Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, eds.), pp. 10–18, Curran Associates, Inc., 2009.

[26] A. Corrigan, F. Camelli, R. Löhner, and J. Wallin, "Running unstructured grid cfd solvers on modern graphics hardware," in *19th AIAA Computational Fluid Dynamics Conference*, no. AIAA 2009-4001, June 2009.

[27] Lu Li, Christoph Kessler, "MeterPU: A Generic Measurement Abstraction API Enabling Energy-tuned Skeleton Backend Selection.," in *Proc. International Workshop on Reengineering for Parallelism in Heterogeneous Parallel Platforms (REPARA-2015) at ISPA-2015*, vol. 3, pp. 154–159, IEEE, 2015.

[28] "Regression diagnostics - https://www.mathworks.com/help/stats/regstats.html."

[29] "x2fx - convert predictor matrix to design matrix - https://www.mathworks.com/help/stats/x2fx.html."

Bibliography

[1] Movidius vision processing unit. http://www.movidius.com/solutions/vision-processing-unit (retrieved on Sept. 23, 2015).

[2] Poski: Parallel optimized sparse kernel interface. http://bebop.cs.berkeley.edu/poski (retrieved on Nov. 17, 2015).

[3] Regression diagnostics - https://www.mathworks.com/help/stats/regstats.html.

[4] Rodinia:accelerating compute-intensive applications with accelerators - http://www.cs.virginia.edu/ skadron/wiki/rodinia/index.php/ rodinia:accelerating_compute-intensive_applications_with_accelerators.

[5] x2fx - convert predictor matrix to design matrix - https://www.mathworks.com/help/stats/x2fx.html.

[6] Y. Abe, H. Sasaki, S. Kato, K. Inoue, M. Edahiro, and M. Peres. Power and performance characterization and modeling of gpu-accelerated systems. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, pages 113–122, May 2014.

[7] Nir Ailon, Ragesh Jaiswal, and Claire Monteleoni. Streaming k-means approximation. In Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 10–18. Curran Associates, Inc., 2009.

[8] Jack J. Dongarra Alexey L. Lastovetsky. *HighPerformance Heterogeneous Computing*. Wiley Online Library, 2009.

[9] M. Alioto. Ultra-low power vlsi circuit design demystified and explained: A tutorial. *IEEE Transactions on Circuits and Systems I: Regular Papers, vol.59, no.1, pp.3-29, Jan. 2012*, 59(1):3–29, Jan 2012.

[10] P Alonso, M F Dolz, R Mayo, and E S Quintana-Orti. Modeling power and energy consumption of dense matrix factorizations on multicore processors. *Concurrency Computat.*, 2014.

[11] Lars Arge, Michael T. Goodrich, Michael Nelson, and Nodari Sitchinava. Fundamental parallel algorithms for private-cache chip multiprocessors. In *Procs of the Twentieth Annual Symp on Parallelism in Algorithms and Architectures*, SPAA '08, pages 197–206, 2008.

[12] Krste Asanovic, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A. Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams, and Katherine A. Yelick. The landscape of parallel computing research: A view from berkeley. *Technical Report No. UCB/EECS-2006-183, University of California, Berkeley*, 2006.

[13] Muhammad Ali Awan and Stefan M. Petters. Race-to-halt energy saving strategies. *Journal of Systems Architecture*, 60(10):796 – 815, 2014.

[14] MichaelA. Bender, GerthStoelting Brodal, Rolf Fagerberg, Riko Jacob, and Elias Vicari. Optimal sparse matrix dense vector multiplication in the i/o model. *Theory of Computing Systems*, 47(4):934–962, 2010.

[15] Keren Bergman, Shekhar Borkar, Dan Campbell, William Carlson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Jon Hiller, Sherman Karp, Stephen Keckler, Dean Klein, Robert Lucas, Mark Richards, Al Scarpelli, Steven Scott, Allan Snavely, Thomas Sterling, R. Stanley Williams, Katherine Yelick, Keren Bergman, Shekhar Borkar, Dan Campbell, William Carlson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Jon Hiller, Stephen Keckler, Dean Klein, Peter Kogge, R. Stanley Williams, and Katherine Yelick. Exascale computing study: Technology challenges in achieving exascale systems, 2008.

[16] Aydin Buluç, Jeremy T. Fineman, Matteo Frigo, John R. Gilbert, and Charles E. Leiserson. Parallel sparse matrix-vector and matrix-transpose-vector multiplication using compressed sparse blocks. In *Procs of the Twenty-first Annual Symp on Parallelism in Algorithms and Architectures*, SPAA '09, 2009.

[17] G. L. T. Chetsa, L. Lefrvre, J. M. Pierson, P. Stolf, and G. Da Costa. A runtime framework for energy efficient hpc systems without a priori knowledge of applications. In *2012 IEEE 18th International Conference on Parallel and Distributed Systems*, pages 660–667, Dec 2012.

[18] Jee Choi, Marat Dukhan, Xing Liu, and Richard Vuduc. Algorithmic time, energy, and power on candidate hpc compute building blocks. In *Proceedings of the 2014 IEEE 28th International Parallel and Distributed Processing Symposium*, IPDPS '14, pages 447–457, Washington, DC, USA, 2014.

[19] Jee Whan Choi, Daniel Bedard, Robert Fowler, and Richard Vuduc. A roofline model of energy. In *Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, IPDPS '13, pages 661–672, Washington, DC, USA, 2013.

[20] Henry Cook, Miquel Moreto, Sarah Bird, Khanh Dao, David A. Patterson, and Krste Asanovic. A hardware evaluation of cache partitioning to improve utilization and energy-efficiency while preserving responsiveness. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ISCA '13, 2013.

[21] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 2009.

[22] Andrew Corrigan, Fernando Camelli, Rainald Löhner, and John Wallin. Running unstructured grid cfd solvers on modern graphics hardware. In *19th AIAA Computational Fluid Dynamics Conference*, number AIAA 2009-4001, June 2009.

[23] Timothy A. Davis and Yifan Hu. The university of florida sparse matrix collection. *ACM Trans. Math. Softw.*, 38(1):1:1–1:25, 2011.

[24] Wu-chun Feng, Heshan Lin, Thomas Scogland, and Jing Zhang. Opencl and the 13 dwarfs: A work in progress. In *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, ICPE '12, pages 291–294, New York, NY, USA, 2012. ACM.

[25] Malte Forster and Jiri Kraus. Scalable parallel amg on ccnuma machines with openmp. *Computer Science - Research and Development*, 26(3-4):221–228, 2011.

[26] Matteo Frigo, Charles E. Leiserson, Harald Prokop, and Sridhar Ramachandran. Cache-oblivious algorithms. In *Procs of the 40th Annual Symp on Foundations of Computer Science*, FOCS, 1999.

[27] Matteo Frigo and Volker Strumpen. The cache complexity of multithreaded cache oblivious algorithms. In *Procs of the Eighteenth Annual ACM Symp on Parallelism in Algorithms and Architectures*, SPAA '06, pages 271–280, 2006.

[28] John R. Gilbert, Cleve Moler, and Robert Schreiber. Sparse matrices in matlab: Design and implementation. *SIAM J. Matrix Anal. Appl.*, 13(1):333–356, January 1992.

[29] J. Guerreiro, A. Ilic, N. Roma, and P. Tomas. Gpgpu power modeling for multi-domain voltage-frequency scaling. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 789–800, Feb 2018.

[30] Marisabel Guevara, Benjamin Lubin, and Benjamin C. Lee. Market mechanisms for managing datacenters with heterogeneous microarchitectures. *ACM Trans. Comput. Syst.*, 32(1):3:1–3:31, February 2014.

[31] P. Ha, V. Tran, I. Umar, P. Tsigas, A. Gidenstam, P. Renaud-Goud, I. Walulya, and A. Atalar. Models for energy consumption of data structures and algorithms. Technical report, EU FP7 project EXCESS deliverable D2.1 (http://www.excess-project.eu), 2014.

[32] Phuong Ha, Vi Tran, Ibrahim Umar, Aras Atalar, Anders Gidenstam, Paul Renaud-Goud, and Philippas Tsigas. D2.2 White-box methodologies, programming abstractions and libraries. Technical Report FP7-611183 D2.2, EU FP7 Project EXCESS, February 2015.

[33] Phuong Ha, Vi Tran, Ibrahim Umar, Aras Atalar, Anders Gidenstam, Paul Renaud-Goud, Philippas Tsigas, and Ivan Walulya. D2.3 d2.4 report on the final prototype of programming abstractions for energy-efficient inter-process communication. Technical Report FP7-611183 D2.4, EU FP7 Project EXCESS, August 2016.

[34] Phuong Ha, Vi Tran, Ibrahim Umar, Aras Atalar, Anders Gidenstam, Paul Renaud-Goud, Philippas Tsigas, and Ivan Walulya. D2.3 power models, energy models and libraries for energy-efficient concurrent data structures and algorithms. Technical Report FP7-611183 D2.3, EU FP7 Project EXCESS, February 2016.

[35] Phuong Ha, Vi Tran, Ibrahim Umar, Philippas Tsigas, Anders Gidenstam, Paul Renaud-Goud, Ivan Walulya, and Aras Atalar. D2.1 Models for energy consumption of data structures and algorithms. Technical Report FP7-611183 D2.1, EU FP7 Project EXCESS, August 2014.

[36] M.D. Hill and M.R. Marty. Amdahl's law in the multicore era. *Computer*, 41(7):33–38, 2008.

[37] P. Hoai Ha, N.-N. Tran, Vi, I. Umar, P. Tsigas, A. Gidenstam, P. Renaud-Goud, I. Walulya, and A. Atalar. D2.1 Models for energy consumption of data structures and algorithms. *ArXiv e-prints*, January 2018.

[38] C. Imes, D. H. K. Kim, M. Maggio, and H. Hoffmann. Poet: a portable approach to minimizing energy under soft real-time constraints. In *21st IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 75–86, April 2015.

[39] C. Imes, D.H.K. Kim, M. Maggio, and H. Hoffmann. Poet: a portable approach to minimizing energy under soft real-time constraints. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2015 IEEE*, pages 75–86, April 2015.

[40] Mircea Horea Ionica and David Gregg. The movidius myriad architecture's potential for scientific computing. *Micro, IEEE*, 35(1):6–14, Jan 2015.

[41] H. Jacobson, A. Buyuktosunoglu, P. Bose, E. Acar, and R. Eickemeyer. Abstraction and microarchitecture scaling in early-stage power modeling. In *IEEE 17th International Symposium on High Performance Computer Architecture (HPCA), 2011*, pages 394–405, Feb 2011.

[42] Chao Jin, Bronis R de Supinski, David Abramson, Heidi Poxon, Luiz DeRose, Minh Ngoc Dinh, Mark Endrei, and Elizabeth R Jessup. A survey on software methods to improve the energy efficiency of parallel computing. *The International Journal of High Performance Computing Applications*, 31(6):517–549, 2017.

[43] Christoph Kessler, Lu Li, Usman Dastgeer, Rosandra Cuello, Oskar Sjöström, Phuong Ha Hoai, and Vi Tran. D1.3 Energy-tuneable domain-specific language/library for linear system solving. Technical Report FP7-611183 D1.3, EU FP7 Project EXCESS, February 2015.

[44] Christoph Kessler, Lu Li, Usman Dastgeer, Philippas Tsigas, Anders Gidenstam, Paul Renaud-Goud, Ivan Walulya, Aras Atalar, David Moloney, Phuong Ha Hoai, and Vi Tran. D1.1 Early validation of system-wide energy compositionality and affecting factors on the EXCESS platforms. Technical Report FP7-611183 D1.1, EU FP7 Project EXCESS, April 2014.

[45] Dmitry Khabi, Vi Tran, and Ivan Walulya. D5.4 Report on the first evaluation results and discussion. Technical Report FP7-611183 D5.4, EU FP7 Project EXCESS, August 2015.

[46] V.A. Korthikanti and Gul Agha. Analysis of parallel algorithms for energy conservation in scalable multicore architectures. In *International Conference on Parallel Processing, 2009. ICPP '09.*, pages 212–219, Sept 2009.

[47] Vijay Anand Korthikanti and Gul Agha. Towards optimizing energy costs of algorithms for shared memory architectures. In *Proceedings of the 22nd Annual ACM Symposium on Parallelism in Algorithms and Architectures*, 2010.

[48] Vladimir Kotlyar, Keshav Pingali, and Paul Stodghill. A relational approach to the compilation of sparse matrix programs. Technical report, 1997.

[49] Pardeep Kumar, Andrei Gurtov, and Phuong H. Ha. An efficient authentication model in smart grid networks. In *Procs of the 15th Int Conf on Information Processing in Sensor Networks*, pages 65:1–65:2, 2016.

[50] J. Lagravire, J. Langguth, M. Sourouri, P. H. Ha, and X. Cai. On the performance and energy efficiency of the pgas programming model on multicore architectures. In *Procs of the Int Workshop on Optimization of Energy Efficient HPC & Distributed Systems (OPTIM-HPCS)*, pages 800–807, 2016.

[51] Etienne Le Sueur and Gernot Heiser. Dynamic voltage and frequency scaling: The laws of diminishing returns. In *Proceedings of the 2010 International Conference on Power Aware Computing and Systems*, HotPower'10, pages 1–8, Berkeley, CA, USA, 2010. USENIX Association.

[52] Victor W. Lee, Changkyu Kim, Jatin Chhugani, Michael Deisher, Daehyun Kim, Anthony D. Nguyen, Nadathur Satish, Mikhail Smelyanskiy, Srinivas Chennupaty, Per Hammarlund, Ronak Singhal, and Pradeep Dubey. Debunking the 100x gpu vs. cpu myth: An evaluation of throughput computing on cpu and gpu. In *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ISCA '10, pages 451–460, New York, NY, USA, 2010. ACM.

[53] Jingwen Leng, Tayler Hetherington, Ahmed ElTantawy, Syed Gilani, Nam Sung Kim, Tor M. Aamodt, and Vijay Janapa Reddi. Gpuwattch: Enabling energy optimizations in gpgpus. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ISCA '13, pages 487–498, New York, NY, USA, 2013. ACM.

[54] Lu Li and Christoph Kessler. Validating energy compositionality of GPU computations. In *HIPEAC Workshop on Energy Efficiency with Heterogeneous Computing (EEHCO)*, 2015.

[55] Sheng Li, Jung Ho 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*, 2009.

[56] Xu Liu and John Mellor-Crummey. A tool to analyze the performance of multithreaded programs on numa architectures. In *Proceedings of the 19th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '14, 2014.

[57] Lu Li, Christoph Kessler. MeterPU: A Generic Measurement Abstraction API Enabling Energy-tuned Skeleton Backend Selection. In *Proc. International Workshop on Reengineering for Parallelism in Heterogeneous Parallel Platforms (REPARA-2015) at ISPA-2015*, volume 3, pages 154–159. IEEE, 2015.

[58] K. Ma, X. Li, W. Chen, C. Zhang, and X. Wang. Greengpu: A holistic approach to energy efficiency in gpu-cpu heterogeneous architectures. In *2012 41st International Conference on Parallel Processing*, pages 48–57, Sept 2012.

[59] Kai Ma, Yunhao Bai, Xiaorui Wang, Wei Chen, and Xue Li. Energy conservation for gpucpu architectures with dynamic workload division and frequency scaling. *Sustainable Computing: Informatics and Systems*, 12:21 – 33, 2016.

[60] Alberto Magni, Christophe Dubach, and Michael F. P. O'Boyle. A large-scale cross-architecture evaluation of thread-coarsening. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, pages 11:1–11:11, New York, NY, USA, 2013. ACM.

[61] A. Majumdar, L. Piga, I. Paul, J. L. Greathouse, W. Huang, and D. H. Albonesi. Dynamic gpgpu power management using adaptive model predictive control. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 613–624, Feb 2017.

[62] A. Cristiano I. Malossi, Yves Ineichen, Costas Bekas, Alessandro Curioni, and Enrique S. Quintana-Orti. Systematic derivation of time and power models for linear algebra kernels on multicore architectures. *Sustainable Computing: Informatics and Systems*, 7:24 – 40, 2015.

[63] Nikita Mishra, Huazhe Zhang, John D. Lafferty, and Henry Hoffmann. A probabilistic graphical model-based approach for minimizing energy under performance constraints. In *Proceedings of the 20th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '15. ACM, 2015.

[64] S. Muralidharan, M. Shantharam, M. Hall, M. Garland, and B. Catanzaro. Nitro: A framework for adaptive code variant tuning. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, pages 501–512, May 2014.

[65] Rajib Nath and Dean Tullsen. The crisp performance model for dynamic voltage and frequency scaling in a gpgpu. In *Proceedings of the 48th International Symposium on Microarchitecture*, MICRO-48, pages 281–293, New York, NY, USA, 2015. ACM.

[66] G. Ofenbeck, R. Steinmann, V. Caparros, D.G. Spampinato, and M. Puschel. Applying the roofline model. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), 2014*, pages 76–85, March 2014.

[67] Anne-Cecile Orgerie, Marcos Dias de Assuncao, and Laurent Lefevre. A survey on techniques for improving the energy efficiency of large-scale distributed systems. *ACM Comput. Surv.*, 46(4):47:1–47:31, 2014.

[68] Franz Pernkopf, Robert Peharz, and Sebastian Tschiatschek. *Introduction to Probabilistic Graphical Models*, volume 1, chapter 18, pages 989–1064. Elsevier, 2014.

[69] Phitchaya Mangpo Phothilimthana, Tikhon Jelvis, Rohin Shah, Nishant Totla, Sarah Chasins, and Rastislav Bodik. Chlorophyll: Synthesis-aided compiler for low-power spatial architectures. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '14, 2014.

[70] Swapnoneel Roy, Atri Rudra, and Akshat Verma. An energy complexity model for algorithms. In *Procs of the 4th Conf on Innovations in Theoretical Computer Science*, ITCS '13, pages 283–304, 2013.

[71] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd edition, 2003.

[72] J. Shen, A. L. Varbanescu, Y. Lu, P. Zou, and H. Sips. Workload partitioning for accelerating applications on heterogeneous platforms. *IEEE Transactions on Parallel and Distributed Systems*, 27(9):2766–2780, Sept 2016.

[73] David C. Snowdon, Etienne Le Sueur, Stefan M. Petters, and Gernot Heiser. Koala: A platform for os-level power management. In *Proceedings of the 4th ACM European Conference on Computer Systems*, EuroSys '09, pages 289–302, New York, NY, USA, 2009. ACM.

[74] T. Suzumura, K. Ueno, H. Sato, K. Fujisawa, and S. Matsuoka. Performance characteristics of graph500 on large-scale distributed environment. In *IEEE International Symposium on Workload Characterization (IISWC), 2011*, pages 149–158, 2011.

[75] T. Takagi and M. Sugeno. Fuzzy identification of systems and its applications to modeling and control. *Systems, Man and Cybernetics, IEEE Transactions on*, SMC-15(1):116–132, Jan 1985.

[76] SYMPRAXIS TEAM. Assessing the employment and social impact of energy efficiency. Technical report, Cambridge Econometrics, 2015.

[77] V. N. N. Tran, B. Barry, and P. H. Ha. Power models supporting energy-efficient co-design on ultra-low power embedded systems. In *2016 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS)*, pages 39–46, July 2016.

[78] V. N. N. Tran and P. H. Ha. Ice: A general and validated energy complexity model for multithreaded algorithms. In *2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS)*, pages 1041–1048, Dec 2016.

[79] V. N.N. Tran, T. Oines, A. Horsch, and P. H. Ha. Reoh: Using probabilistic network for runtime energy optimization of heterogeneous systems. In *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*, 2018.

[80] Vi Ngoc-Nha Tran, Brendan Barry, and Ha. Rthpower: Accurate fine-grained power models for predicting race-to-halt effect on ultra-low power embedded systems. In *Proceedings of the 17th IEEE International Symposium on Performance Analysis of Systems and Software*, ISPASS 16, 2016.

[81] Ibrahim Umar, Otto J. Anshus, and Phuong H. Ha. Effect of portable fine-grained locality on energy efficiency and performance in concurrent search trees. In *Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '16, 2016.

[82] Ibrahim Umar, Otto J. Anshus, and Phuong H. Ha. Greenbst: An energy-efficient concurrent search tree. In *Proc. of the 22nd Intl. European Conf. on Parallel and Distributed Computing (Euro-Par 16)*, page pages to appear., 2016.

[83] Ibrahim Umar, Otto Johan Anshus, and Phuong Hoai Ha. Deltatree: A locality-aware concurrent search tree. In *Proceedings of the 2015 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '15, 2015.

[84] Richard Vuduc, James W Demmel, and Katherine A Yelick. Oski: A library of automatically tuned sparse matrix kernels. In *Institute of Physics Publishing*, 2005.

[85] Qiang Wang and Xiaowen Chu. Gpgpu power estimation with core and memory frequency scaling. *SIGMETRICS Perform. Eval. Rev.*, 45(2):73–78, October 2017.

[86] S. Williams, L. Oliker, R. Vuduc, J. Shalf, K. Yelick, and J. Demmel. Optimization of sparse matrix-vector multiplication on emerging multicore platforms. In *Proceedings of the 2007 ACM/IEEE Conference on Supercomputing, 2007. SC '07.*, pages 1–12, Nov 2007.

[87] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: An insightful visual performance model for multicore architectures. *Commun. ACM*, 52(4):65–76, 2009.

[88] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *Proceedings of IEEE 36th Annual Foundations of Computer Science*, pages 374–382, Oct 1995.

[89] Kathy Yelick. Cs 267 parallel matrix multiplication, Sept 2004.

[90] F. Zhang, B. Wu, J. Zhai, B. He, and W. Chen. Finepar: Irregularity-aware fine-grained workload partitioning on integrated architectures. In *2017 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pages 27–38, Feb 2017.

[91] F. Zhang, J. Zhai, B. He, S. Zhang, and W. Chen. Understanding co-running behaviors on integrated cpu/gpu architectures. *IEEE Transactions on Parallel and Distributed Systems*, 28(3):905–918, March 2017.

[92] Huazhe Zhang and Henry Hoffmann. Maximizing performance under a power cap: A comparison of hardware, software, and hybrid techniques. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '16, pages 545–559, New York, NY, USA, 2016. ACM.

[93] Albert Y. Zomaya and Young Choon Lee. *Energy Efficient Distributed Computing Systems.* Wiley-IEEE Computer Society Pr, 1st edition, 2012.