Towards Reliable and Accurate Energy Predictive Modelling using Performance Events on Modern Computing Platforms

Arsalan Shahid

UCD student number: 16203221

The thesis is submitted to University College Dublin
in fulfilment of the requirements for the degree of
Doctor of Philosophy in Computer Science

School of Computer Science

Head of School: Assoc. Professor Chris Bleakley

Research Supervisor: Assoc. Prof. Alexey Lastovetsky

May 2020

Acknowledgements

First and foremost, I would like to thank Almighty Allah for giving me the opportunity, determination and strength to do my research. His continuous grace and mercy enabled me to be as yielding, useful and benevolent to His universe as possible. I would like to express my special thanks to my supervisor Prof. Alexey Lastovetsky for acceptance in the Heterogeneous Computing Laboratory (HCL). His support, guidance and overall insights in the field of high-performance heterogeneous computing have made this an inspiring experience for me.

I would like to extend my gratitude to Dr Ravi Reddy Manumachu for his guidance, support, and encouragement throughout my Ph.D. I would also like to pay a special respect to Muhammad Fahad for appreciation, support, and collaboration. Finally, from the bottom of my heart I would like to say big thank you to all my colleagues in UCD's HCL for their fruitful collaborations: Emin Nuriyev, Semen Khokhriakov, Hamidreza Khaleghzadeh and Tania Malik.

The last four years of my life have been truly rewarding for becoming a member of the highly prestigious University College Dublin. I would like to extend my respect and regards to Prof. Chris Bleakley, the head of school of computer science, for his trust, wise guidance, and support for providing me the appointment as a lecturer for computer systems module in the school. I also thank Prof. Barry Smyth and all the professors from the school who provided me the opportunities to demonstrate the world-class and structured modules. From the UCD School of Computer Science, thank you to both Prof Chris Bleakley and Prof Michela Bertolotto, my DSP members. I specially thank the examiners of my Ph.D. thesis, Prof. Francisco Carmelo Almeida Rodriguez and Prof. Mel Ó Cinnéide, and the Chair of Examination Committee Prof. Tahar Kechadi for their insightful feedback for polishing this thesis. Thank you also to the school's support staff: Paul Martin, Anthony O'Gara, Lorraine

McHugh, and D'Arcey Jackson, for their consistent helpfulness over the years.

This research has emanated from research conducted with the financial support of Science Foundation Ireland (SFI) under Grant Number 14/IA/2474. I would like to thank SFI and University College Dublin for their financial support in the form of scholarship awards. I am also grateful to the financial support of COST Action IC0805 "Open European Network for High-Performance Computing on Complex Environment" for the insightful summer schools and workshops held in University of Calabria (Italy), University College Dublin (Ireland) and Ruđer Bošković Institute (Croatia). I would also like to thank the organizing team of the International HPC summer school held in TU Ostrava (Czech Republic) for accepting and supporting my participation.

I am grateful to Huawei Technologies Ireland Research Center (IRC) for providing me a research opportunity to contribute towards the state-of-the-art innovations in cloud-computing domains such as resource mapping and optimization and anomaly detection based on innovative machine learning algorithms. I am thankful to my manager Owen O'Brien, for his insightful comments on my work and his guidance to keep me inline with the organization objectives. I further thank my team at Huawei IRC for their collaborations: NiZhenWei Ni, Jaroslaw Diuwe, Rogerio Robetti, and Lorenzo Cipriani.

I would like to thank the many amazing friends I met in Ireland who made this journey full of pleasures.

A heartfelt thanks to my parents and my brother Ahtsham for their love and support throughout my life. Thank you both for giving me strength to reach for the stars and chase my dreams.

To my family.

Abstract

Information and Communication Technologies (Information and Communication Technologies (ICT)) systems and devices are forecast to consume up to 50% of global electricity in 2030. Considering the unsustainable future predicted, energy efficiency in ICT is becoming a grand technological challenge and is now a first-class design constraint in all computing settings. Energy efficiency in ICT can be achieved at the hardware level (or system-level) and software level (or application-level). While the mainstream approach is to minimize the energy of the operating environment and is extensively researched, application-level energy optimization is comparatively understudied and forms the focus of this work.

The fundamental building block for energy minimization at the application level is an accurate measurement of energy consumption during application execution. There are three popular approaches to providing it: (a) System-level physical measurements using external power meters, (b) Measurements using on-chip power sensors, and (c) Energy predictive models. While the first approach is considered to be the ground truth, it is cost-prohibitive for the energy optimization of applications. The second approach using energy measurements by state-of-the-art on-chip sensors is not recommended for energy optimization of applications due to several issues related to the lack of accurate information in the vendor manuals and poorly reported experimental

accuracy. The third approach of energy predictive modelling based on performance monitoring counters (PMCs) as model variables is now the leading method for prediction of energy consumption during application execution. In this thesis, we focus on application-level energy measurement, modelling, and optimization using PMCs and high-level application metrics.

A vast majority of research works propose models where the employed model variables (or PMCs) are selected solely on the basis of their high positive correlation with energy consumption and report prediction accuracies ranging from poor to excellent. There are a few pieces of research that critically examine the inaccuracy of PMC-based models. We present a study to identify the causes of inaccuracies. We introduce *additivity* as a property of PMCs that appears to have a significant impact on the accuracy of energy predictive models. It is based on an experimental observation that energy consumption of serial execution of two applications is equal to the sum of the energy consumption of those applications when they are run separately. Our initial study shows that a vast majority of PMCs on modern multicore CPUs are non-additive. We demonstrate how the accuracy of energy predictive models based on linear regression can be improved by selecting PMCs based on a property of additivity.

We formulate a sound theoretical framework to understand the fundamental significance of the model variables with respect to the energy consumption and the causes of inaccuracy or the reported wide variance of the accuracy of the models. We use a model-theoretic approach to formulate the assumed properties of existing energy predictive models in a mathematical form. We extend the formalism by adding properties, heretofore unconsidered, that account for a limited form of energy conservation law. The extended formalism defines our theory of *energy predictive models for computing*. By applying the

basic practical implications of the theory, we improve the prediction accuracy of state-of-the-art linear regression models from 31.2% to 18.01%.

We show that high positive correlation of the model variables with dynamic energy consumption alone is not sufficient to provide good prediction accuracy but the model variables must satisfy the properties of *consistency* test that takes into account the physical significance of the model variables originating from the conservation of energy of computing, the manifestation of the fundamental physical law of energy conservation. We demonstrate that use of the state-of-the-art measurement tools for energy optimization may lead to significant losses of energy (up to 84% for applications used in experiments) since they do not take into account the properties of the theory of energy predictive models for computing.

Finally, we present the first comprehensive experimental study to compare the energy predictive modelling techniques employing PMCs. We show that the selection of model variables using techniques such as additivity that incorporate their physical significance with energy consumption produce performs better in terms of average prediction accuracies than statistical methods. We discover that the PMCs that record the activities of CPU cores during the execution of the applications must be used used to train the energy predictive models. This is because they are representative of dynamic energy consumption activities. We further demonstrate that a platform-level and application-level linear regression-based model employing the additive PMCs, irrespective of the applications used for training and testing, performs better in terms of average prediction accuracies.

Contents

List of Figures

List of Tables

Statement of Original Authorship

I hereby certify that the submitted work is my own work, was completed while registered as a candidate for the degree stated on the Title Page, and I have not obtained a degree elsewhere on the basis of the research presented in this submitted work.

Chapter 1

Introduction

Since the emergence of data-driven products aiming towards a fully automated and intelligent connected world, the high-quality computing resource is considered as a top-notch strategic component for innovations and economic boosts. The high-performance computing (HPC) platforms, dedicated data centers, and cloud-based servers provide the computational power to the industry and small and medium enterprises to realize the objectives of a smart future for mankind with improved living standards.

For decades, achieving performance gains from computing resources is the major concern of both the hardware architects and the application developers. The implications of Moore's law [2] translate in the doubling of the number of transistors in a dense integrated circuit every 18 to 24 months. According to Dennard scaling [3], a postulation of Moore's law guides that the power use of the transistors is directly proportional to their area. Moore's law and Dennard's scaling have been providing a road-map for researchers for decades to achieve exponential improvement in the performance of processors by increasing their clock frequencies without any significant additional power consumption. However, around 2006, the break of Dennard's scaling put to a permanent stop on getting further benefits and improving performance. As a result, the processor manufacturers evolved the architectures from single-core to multi-core and many-core platforms. However, the end of Moore's law has practical implications to stop this type of performance scaling.

The rest of this chapter is divided into four main sections. In Section 1.1, we present the evolution of computing platforms for performance and briefly summarize the recent research and development in the area of performance modelling and optimization for the latest high-performance computing platforms. Section 1.2 presents the motivation of research and highlight energy consumption as a leading concern for HPC platforms. We also present the main energy measurement approaches of computing and discuss how energy predictive models evolved as the dominant method for energy measurement during the execution of applications. We conclude this section with a summary of the main challenges of energy measurement methods. In Section 1.3, we list the main contributions of this research, and finally Section 1.4 details the structure of the thesis.

1.1 High-Performance Computing Platforms & Challenges

Since the origin, HPC community has always been concerned with the objective of increasing the platform performance for executing an application or set of applications. Figure 1.1 shows the top supercomputing platforms and their increased performance (in petaflops) over a period of time. It can be seen that *U.S. summit* is the fastest supercomputing platform after 2018. However, the next most ambitious objective for an HPC platform is to achieve an exascale performance. China plans to release first exascale machine by the end of year 2020. It should be noted that latest HPC platforms have become highly heterogeneous owing to tight integration of multicore CPUs and accelerators (such as GPUs, Intel Xeon Phi (Xeon Phi)s, or Field Programmable Gate Array (FPGA)s) to maximize the dominant objectives of performance and energy efficiency.

Modern computing platforms are evolving with increased complexities such as high resource contention and non-uniform memory access (NUMA). The tight integration of tens of cores organized in multiple sockets with multi-level cache hierarchy and contending for shared on-chip resources such as last

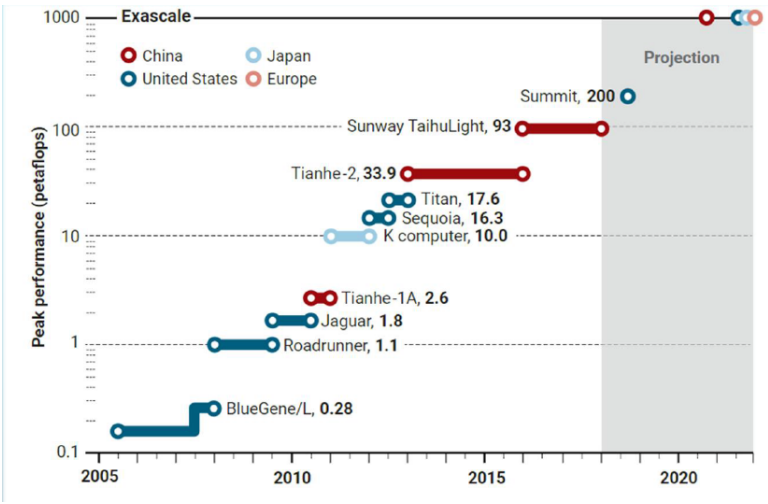


Figure 1.1: Evolution of HPC Platforms for Performance.

level cache and DRAM controllers causes severe resource contention. The NUMA is because the time for memory access between a core and main memory is not uniform as the main memory is distributed between locality domains or groups called NUMA nodes.

1.1.1 Evolution of Performance Modelling for HPC Platforms: A Bird's-Eye View

For over three decades before the mid-2000s, computing users came to expect processor performance doubling every 18 months because of Moore's law and Dennard scaling. Both clock rate and power increased rapidly. From the HPC point of view, this was the era of homogeneous and heterogeneous clusters of single-core processors. However, by late 2003, system designers hit the power wall caused by issues because of ever-increasing power consumption and power density (that is, the amount of power dissipated per unit area, which represents the heat dissipation). The power problem was caused mainly by the breakdown of Dennard scaling. It is a scaling model that suggests the power density of a transistor-based processor of a unit area remains constant due to voltage and current scaling down with the length of the transistor. Up until 2004, efforts towards reducing the size of the tran-

sistor meant that frequency could be increased for no significant increase in heat dissipation. In other words, the breakdown of Dennard scaling implies that frequency scaling was no longer an economical option. Therefore, the chip fabrication industry turned to multicore CPU architectures to address this problem of increased power consumption and power density. Frequency scaling was abandoned in favor of multiple processors per chip. In addition to this, around 2001, the use of GPUs for general-purpose computing became practical due to the appearance of programmable shaders and floating-point operations support. Therefore, the performance efficiency of the HPC system has been researched widely by the community. In order to be able to optimize the performance of a computing platform, the modelling is a bigger objective. There are system-level and application-level performance optimization methods that take as input these models.

We briefly study the evolution of performance models and researches that have attempted to realistically capture the real-life behavior of applications executing on these platforms for performance maximization.

The simplest performance models used positive constant numbers and various terms such as normalized processor speed, normalized cycle time, task computation time, average execution time, etc., to characterize the speed of an application [4], [5], [6]. The common aspect of these models is that the performance of a processor is assumed to have no dependence on the size of the workload. We call them constant performance models (CPMs).

The most advanced load balancing algorithms use functional performance models (FPMs), that are application-specific. The FPMs represent the speed of a processor by a continuous function of problem size but satisfies some assumptions on its shape [7],[8]. The assumptions require them to be smooth enough in order to guarantee that optimal solutions minimizing the computation time always load balanced. The FPMs capture accurately the real-life behavior of applications executing on nodes consisting of uniprocessors (single-core CPUs).

However, modern HPC platforms have complex nodal architectures with the highly hierarchical arrangement and tight integration of processors where resource contention and Non-Uniform Memory Access (NUMA) are inherent.

On such platforms, the performance profiles of real-life scientific applications are not smooth and deviate significantly from the shapes that allowed traditional and state-of-the-art load balancing algorithms to find optimal solutions.

Lastovetsky et al. [9] study the drastic deviations in the performance profiles for a real-life scientific application. The authors propose an optimization technique reusing an advanced performance model of computation (FPM) but using novel load distribution to minimize the computation time of the application. Ravi et al. [10] illustrate in-depth these variations in performance and energy profiles of two widely known and highly optimized scientific routines, OpenBLAS DGEMM [11] and FFTW [12] on a modern multicore Intel Haswell CPU platform. They explain the limitations of the FPM-based load balancing algorithms proposed in [13], [14], [15], [16], [17], [18], [19], [20], [21]. They propose novel model-based methods and algorithms for minimization of time and energy of computations for the most general performance and energy profiles of data-parallel applications executing on homogeneous multicore clusters. Unlike load balancing algorithms, optimal solutions found by these algorithms may not load-balance an application. The new model-based methods proposed in [9], [10], however, can not be used for the optimization of data-parallel applications on HPC platforms with hybrid nodes for maximization of performance since they are designed for homogeneous clusters, i.e., cluster of identical processors. There have also been attempts to profile the performance of a computing system by using high-level parameters such as DRAM activity, data stream, etc. such as the roofline model [22].

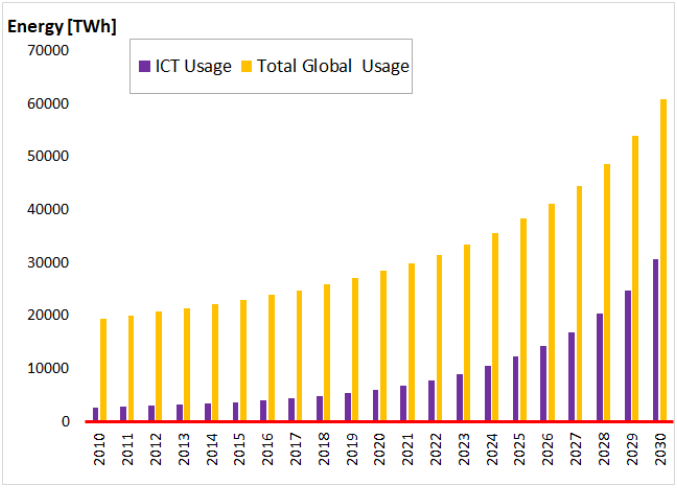
1.2 Motivation of This Research

1.2.1 *Energy*: A Recent & Big Challenging Area

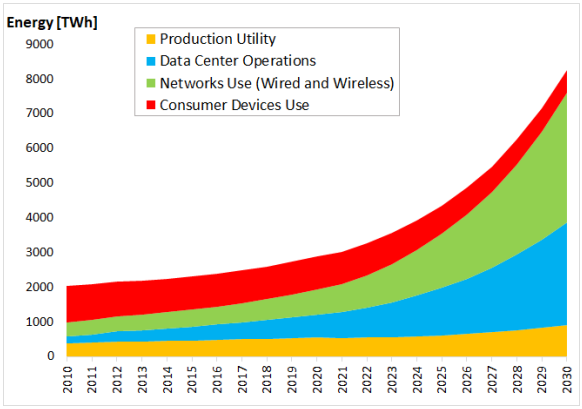
The energy consumption of Information and Communication Technologies (ICT) systems and devices is reported to be about 5300 terawatt-hours (TWh) in the year 2019. It accounts for 20% of the global electricity demand [23]. An alarming picture is the rapidly increasing trend of the energy usage of ICT and its share of global usage (Figure 1.2(a)). If the trend continues, ICT will

consume up to 50% of global electricity in 2030 with a contribution to greenhouse gas emissions of 23% [1]. Figure 1.2(b) shows the breakdown of the main energy consumption sectors in ICT. It can be seen that by 2030 the major contributors to ICT energy would be operational data centers and the network operations in the form of autonomous driving networks (ADN). Figure 1.2(c) clearly forecasts the rise and fall in the ICT sectors towards energy consumptions. To summarize, the forecast for ICT energy consumption portends an unsustainable future. Therefore, energy efficiency in ICT is becoming a grand technological challenge and is now a first-class design constraint in all computing settings [24, 25].

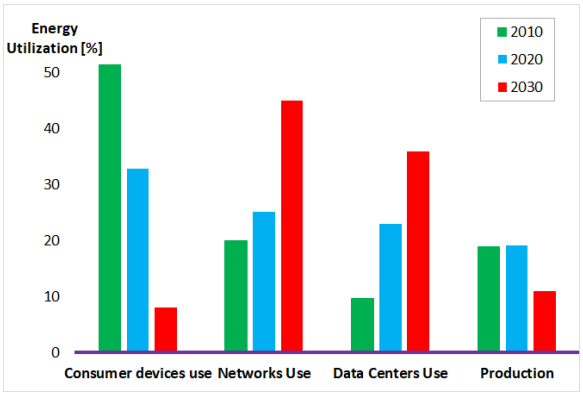
Increased performance of HPC systems comes at the cost of energy consumption and with such a tremendous boost in the performance of these modern systems, tons of watts of power is being consumed by HPC and supercomputing centers. Tianhe-2, the 33.9-petaflop, 3.12-million processor machine by China supercomputing center consumes around 17.8 MW of power and it is equivalent to powering 13501 households, approximately. Intelligently controlling energy consumption is a significant aspect in massive High Performance Computing (HPC) units as modern data-centers can eat-up as much of electricity as a city. According to a DOE Office of Science report [25], excessive energy consumption in HPC system design is now a principal and mainstream challenge in the scientific community. While microprocessor architects and engineers have always been producing energy-efficient computing hardware, it is now a validated fact that the energy consumption of a platform during the execution of the application is significant. Consider, for example, a modern heterogeneous and multicore server (having a CPU, Graphics Processing Unit (GPU), and a Xeon Phi) in its idle mode consuming approximately 200 watts when running with an application using all of its resources can consume up to 1.2 KiloWatts. Furthermore, in today's data-center facility, there are tons and thousands of such servers in operation. This gives the application developers and optimization experts an opportunity to put efforts in optimizing the execution of applications in terms of their energy consumption.



(a) Contribution of the ICT towards the global energy consumption.



(b) Breakdown of energy utilization with in ICT sectors



(c) Energy consumption forecast for ICT sectors

Figure 1.2: Energy consumption distributions for ICT sectors. Adapted from [1].

1.2.2 Challenges in State-of-the-art Energy Efficiency Approaches in Computing

Following are two main approaches to enhance energy efficiency in computing:

1. Hardware approach
2. Software approach

The first approach deals with producing energy-efficient hardware devices at a transistor (or gate) level to boost the efficiency of power electronic devices. The goal is to make these components use as little energy as possible to avoid unnecessarily generated heat and avoid the use of complex cooling systems.

The second approach deals with developing energy-efficient software. Based on the scope and coverage, it can be further subdivided into the following two approaches:

1. System level
2. Application level

The system-level optimization approaches are largely driven by hardware innovations in manufacturing energy-efficient devices. The techniques include clock and power gating, dynamic voltage and frequency scaling (Dynamic Voltage and Frequency Scaling (DVFS)), and dynamic power management (DPM). A component executing an application has two types of power consumption: (a). static power and (b). dynamic power. Static or idle power is the power consumed when the component is not running an application. However, the switching activity in the component's circuits is responsible for dynamic power consumption. Clock gating and power gating are the two main stream techniques that reduce the power consumption of the CMOS circuits. Clock gating technique is used in many synchronous circuits for reducing dynamic power dissipation, by removing the clock signal once the circuit is idle. However, power gating technique is used in integrated circuit designs design to reduce

idle power consumption by turning off the current flowing in the blocks of the idle circuit. Clock gating saves the dynamic power consumption of the components by disabling portions of the circuitry to restrict the flip-flops in them for switching between states. On the other hand, power gating reduces the static power dissipation by turning off portions of inactive circuitry. DPM enables the computing components to move to a low power state when they are not executing the applications. Another approach to reduce the power consumption at a system or platform-level is dynamic frequency and voltage scaling (DVFS) that takes advantage of the modern processors that allow programmers to set different clock frequencies and reduces dynamic power consumption.

The application-levels solution methods include optimization of applications by using application-level decision variables and predictive models for performance and energy consumption of applications. The main decision variables are the number of threads executing the application and the workload distribution among the computing components such as CPU cores, memory banks, etc employed by an application during the execution. The other decision variables can be operating frequency, number of threads, number of processes, etc. This approach is comparatively understudied but recent breakthroughs in application-level optimization methods such as [10, 26, 27, 28, 29], pave the way to excel research and development in this domain. These approaches address the following three main challenges posed by inherent complexities in modern multicore CPUs:

- Severe resource contention due to tight integration of tens of cores organized in multiple sockets with multi-level cache hierarchy and contending for shared on-chip resources such as last level cache (Last Level Cache (LLC)), interconnect (For example Intel's Quick Path Interconnect, AMD's Hyper Transport), and DRAM controllers.

- Non-uniform memory access (NUMA) where the time for memory access between a core and main memory is not uniform and where main memory is distributed between locality domains or groups called NUMA nodes.

- Dynamic power management (DPM) of multiple power domains (CPU sockets, DRAM).

In this thesis, we will focus exclusively on the application-level approach. Accurate measurement of energy consumption during an application execution is key to energy minimization at the application level. There are three popular approaches to providing it are as follows:

1. System-level physical power measurements using external power meters.
2. Measurements using on-chip power sensors.
3. Energy predictive models.

1.2.3 System Level Physical Power Measurements

Using the system-level physical power measurements provided by external power meters in the first approach, the dynamic energy consumption during an application execution is determined by applying the following formula (5.1).

$$E_D = E_T - (P_S \times T_E) \tag{1.1}$$

The total energy consumption E_T is the area under the discrete function of the power samples provided by the power meter versus the time intervals between the samples. Well-known numerical approaches such as trapezoidal rule can be used to calculate this area approximately. The trapezoidal rule works by approximating the area under a function using trapezoids rather than rectangles to get better approximations. The execution time T_E of the application execution can be determined accurately using the processor clocks. The accuracy of obtaining the total energy consumption E_T and the static power consumption P_S is equal to the accuracy provided in the specification of the power meter. Therefore, we consider this approach to be the ground truth. However, the accuracy of measured energy consumption is effected by the accuracy limits of the employed power meters. For example, the variance in the accuracy of power readings for WattsUp Pro power meter is 2.5%.

However, the first approach provides the physical power measurement at a computer level only and therefore lacks the ability to furnish fine-grained decomposition of the energy consumption of an application executing on several independent computing devices on a computer. This component-level decomposition of energy consumption is significant in order to optimize the application by distributing the workload. A naïve approach to optimize the application for dynamic energy consumption will have exponential complexity. The approach must explore all possible workload distributions. For each workload distribution, it determines the total dynamic energy consumption during the parallel execution of the workload by applying the formula (5.1). It, then, returns the workload distribution with the minimum total dynamic energy consumption.

1.2.4 On-chip Sensor Based Power Measurements

The second approach is based on on-chip power sensors now available in mainstream processors such as Intel and AMD Multicore CPUs, Nvidia GPUs, and Intel Xeon Phis. There are vendor-specific libraries to acquire the power data from these sensors. For example, Running Average Power Limit (RAPL) [30] can be used to monitor power and control frequency (and voltage) of Intel CPUs, and Nvidia Management Library (NVML) [31] and Intel System Management Controller chip (SMC) [32] provide the power consumption by Nvidia GPUs and Intel Xeon Phi respectively. The dynamic energy consumption during an application executing on a computing device equipped with on-chip sensors is also calculated using the Formula (5.1). The execution time T_E of the application execution can be determined accurately using the timers provided in the computing device. The base power consumption P_S is obtained using the on-chip sensors when the component is idle. The total energy consumption E_T is calculated from the power samples using the trapezoidal rule.

While the accuracy of Nvidia GPU on-chip sensors is reported in the NVML manual ($\pm 5\%$) [31], the accuracies of the other sensors are not known. For the GPU and Xeon Phi on-chip sensors, there is no information about how a power reading is determined that would allow one to determine its accuracy.

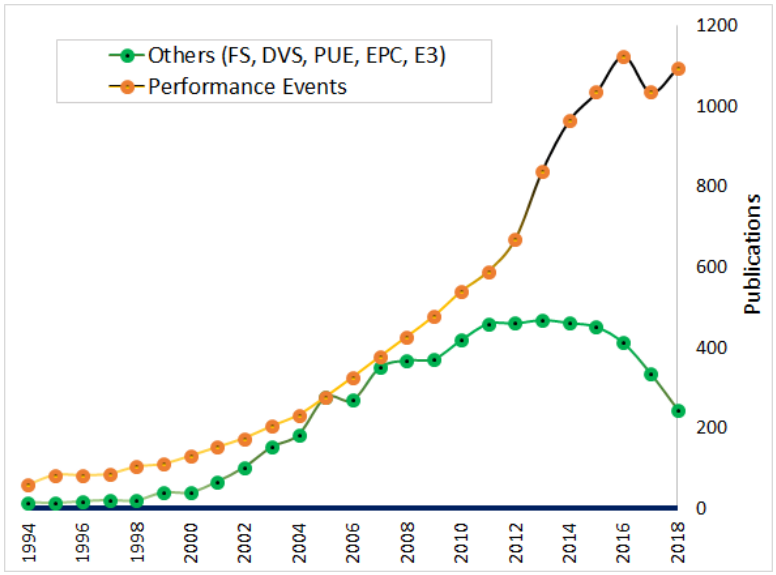
For the CPU on-chip sensors, RAPL uses separate voltage regulators (VR IMON) for both CPU and DRAM. VR IMON is an analog circuit within voltage regulator (VR), which keeps track of an estimate of the current [33]. RAPL is shown to exhibit good prediction accuracy for applications employing decision variables such as dynamic voltage and frequency scaling (DVFS) [34] and the number of application-level threads [35] but keeping the workload size fixed.

However, In [36], the authors demonstrate that RAPL shows poor correlation with real measurements if the workload size is varied and all the other parameters are fixed. They present the first comprehensive comparative study of the accuracy of state-of-the-art on-chip power sensors against system-level physical power measurements using external power meters (the ground truth). They show that energy measurements provided by the state-of-the-art on-chip sensors significantly deviate from the ground truth. Moreover, owing to the nature of the deviations, calibration can not improve the accuracy of the on-chip sensors to the extent that they could favor their use in the optimization of applications for dynamic energy. We define calibration as a constant adjustment (positive or negative value) made to the data points in a dynamic energy profile of an application obtained using a measurement approach (on-chip sensors or energy predictive models) with the aim to reduce its error against the ground truth.

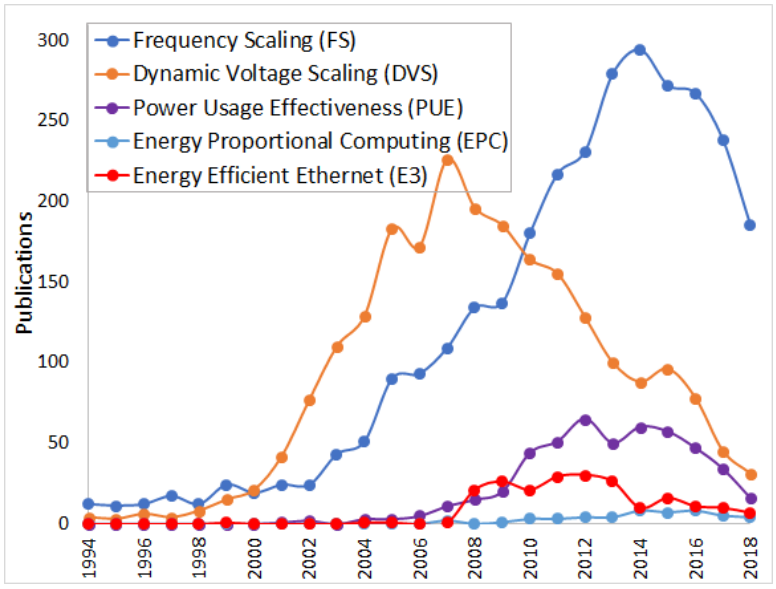
1.2.5 Energy Predictive Modelling

The third approach based on software energy predictive models emerged as a popular alternative to determine the energy consumption of an application. Figures 1.3(a), 1.3(b) illustrate that this approach has been most dominant in recent years as evidenced by the number of academic publications.

A vast majority of such models are linear and employ performance monitoring counters (PMCs) as predictor variables. Performance events or performance monitoring counters (PMCs) are special-purpose registers provided in modern microprocessors to store the counts of software and hardware activities. We will use the acronym PMCs to refer to software events, which are pure kernel-level counters such as *page-faults*, *context-switches*, etc. as



(a)



(b)

Figure 1.3: a). Number of research publications on computing energy and PMCs. b). Research publications in other areas of energy in computing. These statistics have been collected from Google Scholar and Microsoft Academic.

well as micro-architectural events originating from the processor and its performance monitoring unit called the hardware events such as *cache-misses*, *branch-instructions*, etc. They have been developed primarily to aid low-level performance analysis and tuning. Remarkably while PMCs have not been used for performance modeling (Section 1.1.1), over the years, they have become dominant predictor variables for energy predictive modeling.

Modern hardware processors provide a large set of PMCs. Consider the Intel Haswell multicore server CPU. On this server, the PAPI tool [37] provides 53 hardware performance events. The Likwid tool [38], [39] provides 167 PMCs. This includes events for uncore and micro-operations (μops) of CPU cores specific to Haswell architecture that are not provided by PAPI. However, all the PMCs can not be determined using a single application run since only a limited number of registers are dedicated to collecting them. For example, to collect all the Likwid PMCs for a single runtime configuration of an application on the server, the application must be executed 53 times. It must be also pointed out that energy predictive models based on PMCs are not portable across a wide range of architectures. While a model based on either Likwid PMCs or PAPI PMCs may be portable across Intel and AMD architectures, it will be unsuitable for GPU architectures.

Therefore, there are three serious constraints that pose difficult challenges to employing PMCs as predictor variables for energy predictive modeling.

1. There is a large number of PMCs to consider.

2. A lot of programming effort and time are required to automate and collect all the PMCs. This is because all the PMCs can not be collected in one single application run. There are a number of PMCs that can not be collected with each other in the same application run. Therefore, identifying those PMCs also adds to the time for the collection of PMCs.

3. A model purely based on PMCs lacks portability. This means that a model build using PMCs of an Intel processor may not be used for ARM processors because of the non-availability of the same PMCs.

We now focus mainly on techniques employed to select a subset of PMCs

to be used as predictor variables for energy predictive modeling. We present a brief survey of them.

O'Brien et al. [40] survey the state-of-the-art energy predictive models in HPC and present a case study demonstrating the ineffectiveness of the dominant PMC-based modeling approach for accurate energy predictions. In the case study, they use 35 carefully selected PMCs (out of a total of 390 available in the platform) in their linear regression model for predicting dynamic energy consumption. [41], [42], [43] select PMCs manually, based on in-depth study of architecture and empirical analysis. [44], [45], [46], [47], [48], [49], [50] select PMCs that are highly correlated with energy consumption using Spearman's rank correlation coefficient (or Pearson's correlation coefficient) and principal component analysis (Principal Component Analysis (PCA)). [41], [49], [51] use variants of linear regression to remove PMCs that do not improve the average model prediction error.

From the survey, we can classify the existing techniques into three categories.

1. The first category contains techniques that consider all the PMCs with the goal to capture all possible contributors to energy consumption. To the best of our knowledge, we found no research works that adopt this approach.

2. The second category consists of techniques that are based on a statistical methodology such as a correlation and principal component analysis for the selection of PMCs.

3. The last category contains techniques that use expert advice or intuition to pick a subset (that may not necessarily be determined in one application run) and that, in experts' opinion, is a dominant contributor to energy consumption.

However, the existing techniques have not considered one fundamental property of predictor variables that should have been considered in the first place to remove PMCs unfit for modeling energy. We address this oversight in this thesis in Chapter 4.

A pervasive approach is to determine the energy consumption of a hardware component based on linear regression of the PMC counts in the component during an application run. The total energy consumption is then calculated as the sum of these individual consumptions. While the models allow determination of fine-grained decomposition of energy consumption during the execution of an application, there are research works highlighting their poor accuracy [52, 53, 40, 54, 36]. In this thesis, we study the causes of inaccuracies in state-of-the-art energy models.

1.2.6 Summary of Challenges in Energy Measurement Approaches

To summarize, using system-level physical power measurements provided by power meters is an expensive approach.

Energy measurements by state-of-the-art on-chip sensors (RAPL for multicore CPUs, NVML for GPUs, MPSS for Xeon Phis) are not recommended for energy optimization of applications. The fundamental issue with this measurement approach is the lack of information about how a power reading for a component is determined during the execution of an application utilizing the component. While the accuracy of this information is reported in the case of NVML, experimental results demonstrate that practical accuracy is worse [36]. Moreover, the dynamic energy profile patterns against application-level decision variables (such as workload) of the on-chip sensors differ significantly from the patterns obtained using the ground truth, which suggests that the measurements using on-chip sensors do not capture the holistic picture of the dynamic energy consumption during application execution. At the same time, we observed that the energy measurements reported by the on-chip sensors are deterministic and reproducible and, therefore can be used as predictor variables in energy predictive models.

Software energy predictive models emerged as a popular alternative to determine the energy consumption of an application. A vast majority of the models employ PMCs as the model variables. While the main advantage of a software energy predictive model is the determination of fine-grained de-

composition of energy consumption during the execution of an application at less cost compared to the ground truth approach using power meters, this approach suffers from following serious drawbacks.

- The selection of the best set of PMCs as predictor variables is crucial.
- The complexity of model construction and lack of consensus among the research works, which report prediction accuracies ranging from poor to excellent.
- A vast majority of research works select PMCs solely on the basis of their high positive correlation with energy consumption without any deep understanding of the physical significance of the model variables. However, a sound theoretical framework to understand the fundamental significance of the model variables with respect to the energy consumption, and the causes of inaccuracy or the reported wide variance of the accuracy of the models is lacking.

In this thesis, we address these major drawbacks.

1.3 Contributions of This Research

To summarize, the main contributions of this thesis are:

1. A comprehensive study of energy measurement approaches for multicore CPUs. We show that energy predictive models based on PMCs are plagued by poor accuracy.
2. A novel theory of energy predictive models for computing and its practical implications.
3. A novel property for PMCs called *additivity*, to determine the subset of PMCs that can potentially be used for reliable energy predictive modeling.

4. Experimental studies and methodologies demonstrating the improvements in the accuracy of linear energy predictive models using the theory of energy predictive models for computing.

5. Techniques to improve the prediction accuracy energy predictive models for data-parallel applications executing on independently powered processor components. We show that models employing socket-level PMCs, which represent the resource utilization of individually powered components, yield a more accurate energy predictive model.

6. A study demonstrating significant energy losses incurred due to the employment of inaccurate energy measuring tools in energy optimization methods.

7. The first comprehensive experimental study to compare the energy predictive modelling techniques employing PMCs.

1.4 Thesis Structure

The structure of this thesis is as follows. In Chapter 2, we review the evolution of HPC machines, efforts on performance modelling and optimization for modern multicore and heterogeneous platforms, and the existing works on energy measurement, modelling, and optimization methods. We also describe the shortcomings and challenges of energy predictive modelling approaches. In Chapter 3, we present a comprehensive study on the state-of-the-art energy measurement methods including system-level energy measurements, on-chip sensor readings, and predictive models based on performance monitoring counters. In Chapter 4, we first present *additivity* as a selection criterion for PMCs to be used as model variables in energy predictive models. We present the results of the preliminary experimental study of the additivity of PMCs on an Intel Haswell multicore server and show that most of the PMCs are not additive. We then present a novel theory of energy predictive models for computing that lay the basis of a sound theoretical framework to understand the fundamental significance of model variables with respect to energy consumption.

We incorporate the implications of the theory in the state-of-the-art energy models and study their prediction accuracy using strict experimental methodology on multicore CPU platforms. In Chapter 5, we present a comparative study of techniques for platform-level and application-level energy predictive modelling using PMCs. Finally, Chapter 6 concludes the thesis.

Chapter 2

Background and Related Work

We now take a brief tour through the beginnings and consequent firm establishment of performance events as the dominant parameters in energy predictive models (Section 2.1, 2.2, 2.3). While presenting this brief history, we also review system-level power measurements (Section 2.3.3), on-chip sensors (Section 2.3.4), notable models that predict energy consumption based on utilization or activity factors estimated from performance events, and other energy models using additive approaches (Section 2.3.5). This is followed by a survey of research works on system-level energy optimization employing hardware parameters as decision variables and application-level energy optimization using application-level parameters (Section 2.4).

2.1 Evolution of Multicore CPU Platforms

Prior to the emergence of multicore processors and during the era of unicore processors, the computing system users came to expect a doubling in performance every 18 months because of Moore's law and Dennard scaling. However, the goal to achieve a higher performance of applications on multicore CPUs is left as trivial. Some complex issues that surround the migration forward to multicore architecture which affect the performance are as follows:

1. Severe resource contention due to tight integration of tens of cores organized in multiple sockets with multi-level cache hierarchy and contending

for shared on-chip resources such as:

- last level cache (LLC)
- Interconnect (For example Intel's Quick Path Interconnect (Quick Path Interconnect (QPI)), AMD's Hyper Transport)
- DRAM controllers

2. Non-uniform memory access (NUMA) where the time for memory access between a core and main memory is not uniform and where main memory is distributed between locality domains or groups called NUMA nodes

3. Dynamic power management (DPM) of multiple power domains (CPU sockets, DRAM).

These inherent complexities in modern multicore processors pose critical challenges to system modelling experts and algorithm designers.

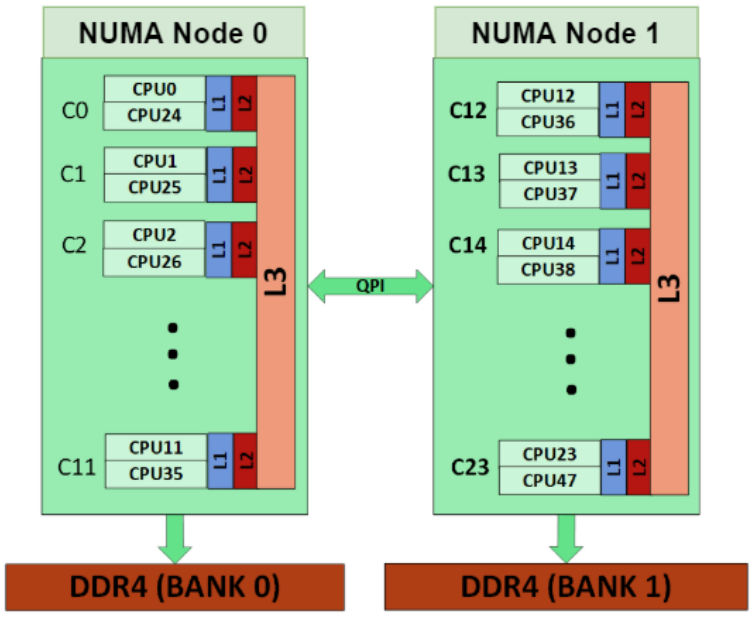


Figure 2.1: A typical processor architecture of a modern multicore CPU

Figure 2.1 depicts a representative architecture of modern multicore CPU processors. It comprises two NUMA nodes or sockets (NUMA node 0 and

NUMA node 1) having 12 physical cores each. Each core has two threads and a private L1 and L2 caches. All the cores in a NUMA node share the L3 cache. Furthermore, each socket has a DDR4 based RAM. The time it takes to access a data block depends on its location in the cache hierarchy and/or the main memory. The closer the data in the memory to the core, the less time it takes to access it. For example, the time it takes to access data in the L1 cache is less than it takes for L2 or L3 caches. Because all the cores in a socket share the same L3 cache, there is a severe resource contention for it between threads that should be tackled by the algorithm designers with an aim to optimize the energy consumption of such systems.

2.2 Terminology for Power and Energy in Computing

This section presents the terminology for power and energy in computing. A component executing an application has two types of power consumption:

- Static power
- Dynamic power

Static or idle power is the power consumed when the component is not running an application. However, the switching activity in the component's circuits is responsible for dynamic power consumption.

Using the aforementioned definitions of power, we define static energy consumption as the energy of a platform without the execution of an application. Whereas, the dynamic energy consumption is calculated by subtracting the static energy consumption from the total energy consumed by the platform during the application run. In other words, If P_S is the static power consumption of the platform, E_T is the total energy consumption of the platform during the execution of an application, which takes T_E seconds, then the dynamic energy E_D can be calculated as,

$$E_D = E_T - (P_S \times T_E) \tag{2.1}$$

In this thesis, we focus purely on the minimization of the dynamic energy consumption for reasons below:

1. Static energy consumption is a constant (or an inherent property) of a platform that can not be optimized. It does not depend on the application configuration.

2. Although the static energy consumption is a major concern in embedded systems, it is becoming less compared to the dynamic energy consumption due to advancements in hardware architecture design in HPC systems.

3. We target applications and platforms where dynamic energy consumption is the dominating energy dissipator.

4. Finally, we believe its inclusion can underestimate the true worth of an optimization technique that minimizes the dynamic energy consumption. We elucidate using two examples from published results.

    - In our first example, consider a model that reports predicted and measured the total energy consumption of a system to be 16500J and 18000J. It would report the prediction error to be 8.3%. If it is known that the static energy consumption of the system is 9000J, then the actual prediction error (based on dynamic energy consumption only) would be 16.6% instead.
    - In our second example, consider two different energy prediction models (M_A and M_B) with the same prediction errors of 5% for application execution on two different machines (A and B) with same total energy consumption of 10000J. One would consider both the models to be equally accurate. But supposing it is known that the dynamic energy proportions for the machines are 30% and 60%. Now, the true prediction errors (using dynamic energy consumptions only) for the models would be 16.6% and 8.3%. Therefore, the second model M_B should be considered more accurate than the first.

2.3 Methods for Energy Consumption Measurement in Computing

We now take a brief tour through the beginnings and consequent firm establishment of performance events as the dominant parameters in energy predictive models. While presenting this brief history, we also review popular energy consumption simulators, system-level energy measurement, on-chip sensors, notable models that predict energy consumption based on utilization or activity factors estimated from performance events, and other energy models using additive approaches. Finally, we provide research works that critically examine the accuracy of PMC based energy predictive models.

2.3.1 Power model in CMOS circuits

Power consumption in CMOS circuits is proportional to the voltage supplied to the processor, V_{dd}^2, and the respective frequency, f. The model can be represented as:

$$P = C_{eff} * V_{dd}^2 * f \tag{2.2}$$

where C_{eff} is the effective capacitance or coefficient of proportionality.

This model has been used in many earlier studies implementing dynamic voltage and frequency techniques to correlate power dissipation. A simpler model is found in the literature is:

$$P = C_{eff} * f^3 \tag{2.3}$$

This model considers frequency as proportional to the supply voltage. Energy savings from memory voltage and frequency scaling techniques have been widely investigated. We will highlight some of the works in the upcoming sections.

2.3.2 Energy and Power Modelling Using Simulators

In the late 1990s, architects began studying in earnest architecture-level power models in simulators similar to how performance is studied in cycle-level architecture simulators.

The Cacti tool (that is an open-source tool for cache access and cycle time modelling) [55] originally written to study latencies of caches in detail, subsequently provided their dynamic power and leakage power models. It was presented by HP Laboratories Inc. and provides thorough, near accurate memory access time and energy estimates. However it is not a trace-driven simulator, so energy consumption resulting in the number of hits or misses is not accounted for a particular application.

In 2000, whole-processor power simulators, SimplePower [56] and Wattch [57], appeared. SimplePower [56] provided detailed dynamic power models of integer ALU and other architectural units in an in-order pipelined processor. The authors analyzed the energy consumption of applications from embedded video and signal processing systems. The experimental results suggested that the simulator helps to point-out the key power-consuming components for the compiler and system designers to optimize.

The Wattch [56] tool focused on an out-of-order super-scalar pipeline. It was claimed to be 1000$\times$ faster than other competitive tools with a compromise of 10% on overall accuracy. The authors further explained the use of Wattch to explore the possibility of dynamic thermal management techniques to reduce power leakages.

While both the aforementioned simulators used analytical methods for modeling power, IBM's PowerTimer [58] used empirical techniques. It predicted the power consumption of an architectural unit based on the measured power consumption of a similar unit in an existing microprocessor and scaling it appropriately; taking into account variations in size and design. Furthermore, the authors also conducted a power-performance tradeoff study.

Li et al. presented McPAT [59] as an integrated power, area, and timing modeling framework for multithreaded, multicore, and manycore architectures. It supports the estimation of power consumption for various compo-

nents in a multiprocessor, which includes shared caches, integrated memory controllers, in-order and out-of-order processor cores, and networks-on-chip. At the microarchitectural level, McPAT includes models for the fundamental components of a chip multiprocessor and at the circuit and technology level, it supports critical-path timing, area, dynamic, short-circuits, and leakage power modeling. McPAT help architects to use new standards combining performance with both area and power. The authors show as the die cost increases with the area, hence the area is a critical design constraint. However, McPAT has known limitations in power estimation, which were reported in [60].

The research and development towards simulators for power and energy analysis had the following main drawbacks:

- *Speed of exploration:* This is a serious limitation of simulators because simulations can take a lot of time (sometimes in hours). The quick and reliable method to model the energy consumptions of the platforms is desired for real-time systems for online energy and performance analysis.
- *Lack of rapid adaptability and sustainability to fast-changing hardware architecture landscape:* As explained before the inherent complexities in modern computing platforms because of resource contention and NUMA, it is difficult to have an accurate simulator that can tackle and take into account the behavior of applications in terms of energy on such platforms.

Because of the limitations of simulators, three mainstream alternatives were:

1. System-level measurements using physical power meters.
2. On-chip power sensors.
3. Energy Predictive Models.

We now discuss these approaches in detail.

2.3.3 System-level measurements using physical power meters

Initially, an appealing alternative route to simulators was allowed by direct physical measurements of power consumption using power meters. Although power meters provided the total power consumption of a system, the major challenges to be addressed are the following:

- *Accurate decomposability of power consumption at the fine-grained component level of granularity:* The decomposition of energy is important to understand the hotspot components which should be optimized for energy. This component-level decomposition of energy consumption is also significant in order to optimize the application by distributing the workload. A naïve approach to optimize the application for dynamic energy consumption will have exponential complexity. The approach must explore all possible workload distributions. For each workload distribution, it determines the total dynamic energy consumption during the parallel execution of the workload by applying the Formula 2.1. It, then, returns the workload distribution with the minimum total dynamic energy consumption.

- *Reduce the time complexity of system-level measurements:* The reliable dynamic energy consumption measurements of the application during its execution is time consuming activity and contain atleast three stages. First, without the execution of the application, one must measure the base power of the system. Second, with the execution of application the total energy consumption of the applications should be measured. In the third stage, the base power should be multiplied with the execution time of the application and then should be subtracted from the total energy consumption to obtain the dynamic energy consumption. Furthermore, the measurements have to be repeated several times until they fall in a desired statistical confidence as the production environments such as data centers are not stable, i.e., the workload vary drastically, which may result in difference of energy consumption measurements for multiple

executions of same application.

To the best of our knowledge, Fahad et al. [61] present the first solution method called *A*dditive e*n*ergy *Mo*delling of *H*ybrid *A*pplications (AnMoHA) to accurately determine the application component level energy consumption employing system-level power measurements using power meters, which we consider to be the ground truth. Authors experimentally validate AnMoHA on a cluster of two hybrid heterogeneous computing nodes for two well-known and highly optimized parallel applications, matrix-matrix multiplication and 2D fast Fourier transform for a diverse range of problem sizes. The error between the combined profile of additive energy models determined by using the summation of the application component energy profiles and parallel profiles determined experimentally ranges between 2% and 5%. There are two shortcomings of this work 1) the component-level decomposition has been shown to be accurate up to socket-levels and the accuracy for core-level energy consumption decomposition is not known and 2) due of complexity of the method, it can not be applied in real-time situations, in general, for modern data-center and cloud computing platforms executing hundreds of applications.

We do not find any other work using system-level measurements to provide the component-level energy consumption decomposition of the applications.

2.3.4 On-chip power sensors

As an alternative approach to system-level measurements, on-chip power sensors have been used to provide the decomposition of energy consumption at a component level [62]. We now review some of the notable research works in this area.

Burtsher et. al [63] examined the power profiles of three different Nvidia GPUs (Tesla K20c, K20m, and K20x) when executing an n-body simulation benchmark using integrated sensors. The authors find that accurate power profiling of an application running on GPU is not straightforward, and there are multiple anomalies when using the on-board sensors on K20 GPUs. They find inaccurate power readings on K20c and K20m, which lag behind the expected profile based on a software model, which they consider to be the ground truth.

Furthermore, the authors observe that the power sampling frequency on K20 GPUs varies greatly and the GPU sensor does not periodically sample power readings.

Intel CPUs offer Running Average Power Limit (RAPL) [30] to monitor power and control frequency (and voltage). RAPL was based on a software model using performance monitoring counters (PMCs) as predictor variables to measure energy consumption for CPUs and DRAM for processor generations preceding Haswell such as Sandybridge and Ivybridge E5. For latest generation processors such as Haswell and Skylake, however, RAPL uses separate voltage regulators (VR IMON) for both CPU and DRAM. VR IMON is an analog circuit within voltage regulator (VR), which keeps track of an estimate of the current. It, however, adds some latency because the measured current-sense signal has a delay from the actual current signal to CPU. This latency may affect the accuracy of the readings. The CPU samples this reading periodically for calculating the power [33]. The accuracy of VR IMON for different input current ranges is not known. According to [33], DRAM and CPU IMON report higher errors when the system is idle and DRAM VR inaccuracy can be large if the system is allocated memory capacity much lower than its capability.

Hackenberg et al. [64] studied the RAPL accuracy on Haswell generation processors by running different micro-benchmarks. They compare the RAPL readings with total system (AC) power consumption using power meters and find the RAPL readings in a strong correlation with AC measurements.

Fahad et al. [36] present the first detailed study on the accuracy of on-chip power sensors and show that deviations of the energy measurements provided by on-chip sensors including Intel RAPL from the system-level power measurements (considered as ground truth) do not motivate their use in the optimization of applications for dynamic energy.

Let us now compare the work of [64] and [36].

- The authors [64] compare the total power consumption by the system with AC power and power consumption by the micro-benchmarks with RAPL. However, [36] uses HPC applications and compare their dynamic

energies calculated using system-level power measurements with the ones calculated using Intel RAPL. Therefore, both use different reference domains.

- In [64], authors run micro-benchmarks in different threading configurations, whereas in [36], authors build the energy profiles of scientific applications representing real-world workloads using different configurations (problem size, CPU threads, CPU cores).
- [64] run their micro-benchmarks on the Haswell platform only whereas for [36], the experimental testbed is more diverse and includes advance generations of Intel CPU micro-architecture.
- [64] find a correlation between the measurements with both tools (power meters and RAPL) on Haswell. However, they could not confirm RAPL can be correlated owing to different reference domain. [36] further extended the knowledge-base by showing that we can not calibrate the measurements with both tools because of their qualitative differences and interlacing behavior.

Alberto et al. [65] [66] presented an energy measurement library (EML) to provide a flexibility in between the hardware and software portability for the instrumented code between several processor architectures. EML addressed the issues such as the lack of abstraction for the users to be able to seamlessly collect energy measurements from low-level APIs and measurement tools with in a desired precision. The authors demonstrate that EML introduces low energy consumption overheads and therefore can be used as a light-weight standard interface.

Intel Xeon Phi co-processors are equipped with on-board Intel System Management Controller chip (SMC) [32] providing energy consumption that can be programmatically obtained using Intel manycore platform software stack (Intel MPSS) [67]. The accuracy of Intel MPSS is not available. AMD starting from Bulldozer micro-architecture equip their processors with an estimation of average power over a certain interval through the Application Power

Management (APM) [68] capability. [35] reports that APM provides highly inaccurate data, particularly during the processor sleep states. In this work, we will not cover tools for AMD processors.

Nvidia Management Library *NVML* [31] provides programmatic interfaces to obtain the energy consumption of an Nvidia GPU from its on-chip power sensors. There are, however, some issues with the energy measurements provided by Nvidia on-chip sensors [63]. One important issue is how to relate the energy consumption of an application and the energy consumption of the computing elements that are involved in the execution of the application and containing the sensors. While sensors may provide the power consumption of a component within sufficient accuracy, they may not determine the energy consumed by an application when executing on the same component within the same accuracy window. For example: while the accuracy of a power reading is reported by NVML for an Nvidia GPU to be 5%, researchers found that when an application is executed on the GPU, the accuracy is often less.

To address these challenges of the inaccuracies and shortcomings in system-level measurements and on-chip sensor-based readings, the performance events based approaches estimating power consumption of architectural units based on their activity factors were invented and eventually became the core of current energy predictive models. The accelerated adoption of this approach was made possible by the simultaneous provision of hardware performance counters (almost akin to standardization) by all the major hardware vendors, and also the availability of lightweight tools providing portable APIs to determine the PMCs.

2.3.5 Notable Energy Predictive Models on Modern Computing Platforms

Energy Predictive Models for CPUs

Energy Predictive Modelling using Additive Approaches:

In this section, we review some of the popular models that use an additive approach to measure or model the energy consumption on a computing plat-

form. This approach sums the contributions of individual energy-consuming elements in a processor and memory.

One of the simplest additive models was presented by Roy et al. [69] which represents an algorithm energy consumption as a weighted sum of the energy consumption by CPU and memory. The authors contribute by providing a simple model constructed from a more complicated model that better models the energy obtained from the ground truth.

Lewis et al. [70] proposed a system-wide energy model (based on hardware performance counters) as a summation of power models of processor, memory (DRAM), fans, motherboard (chipset) peripherals, and hard-disk drive. The model uses statistical estimation methods and build a linear regression model for run-time power analysis. The authors evaluated their model using SPEC CPU benchmarks and showed the accuracy of the energy model within an error of 4%.

Basmadjian et al. [71] presented a similar model including more components such as network interface card and power supply unit to the equation for constructing a similar aggregated power model of the server as a function of resource utilization by its sub-components. The authors evaluated their model for tower and blade server and demonstrate the error rate of 2% and 10% for the best-case and worst-case scenarios, respectively.

Energy Predictive Modelling Using Utilization Parameters:

The early energy modelling efforts include works where researchers use the resource utilization parameters such as CPU, memory, network, and I/O utilization statistics with the motivation to capture all the possible energy-consuming activities.

Heath et al. [72] proposed a linear model that is based on the utilization of CPU, disk, and the network. The proposed power model for a compute node can be represented by the following formula:

$$P = C_{base} + C_1 \times U_{CPU} + C_2 \times U_{Disk} + C_3 \times U_{Net} \tag{2.4}$$

where C_{base} is the base power of a node. C_1, C_2, and C_3 are the coefficients in the linear model for utilization of CPU, disk, and network. The experimental

setup consists up of a server with four Pentium III processor-based PCs and four Celeron based blade servers. The total power consumed is calculated as the sum of power consumption of the hardware resources. The authors use micro-benchmarks (also known as synthetic applications) to stress each component individually with different workload configurations. The results showed an average and maximum errors of 1.3% and 2.7%, respectively.

A more complex power model (Mantis) is proposed in [52] relying on the utilization metrics of CPU, disk, and network components and PMCs for memory. The proposed model can be represented using the following formula:

$$P = C_{base} + C_1 \times U_{CPU} + C_2 \times U_{Mem} + C_3 \times U_{Disk} + C_4 \times U_{Net} \quad (2.5)$$

where C_{base} is the base power of a computing node. U_{CPU}, U_{Mem}, U_{Disk}, and U_{Net} represent the CPU, memory, disk, and network utilizations respectively. The authors performed the experiments on two servers using diverse applications from SPEC and Stream benchmarking suites with varying resource needs. The models were calibrated for blade servers using idle runs. Several benchmarks including SPECcpu2000, SPECjbb2000, and SPECweb2005 benchmarks suites and STREAM benchmarks were used in the experiments.

Fan et al. [73] propose a simple linear model that correlates the power consumption of a single-core processor with its utilization. Their proposed model can be represented as follows:

$$P_{CPU} = P_{base} + (P_{max} - P_{base}) \times (U/100) \quad (2.6)$$

where P_{base} represents the base power of a processor. P_{max} is the power at maximum utilization and U signifies the utilization of the processor. The authors obtained the tuning parameter r during the calibration process using a similar model to [52].

Energy profiling for applications using CPU and disk activity is the subject in [74]. The author proposes an automated energy profiling tool to help a programmer select between different energy and performance trade-offs.

Feng et al. [75] conducted a similar study using the PowerPack tool. Their work focused on building actual power consumption profiles in a cluster of homogeneous nodes. This is done by isolating power by component and measuring the power consumed in the CPU, memory, disk, and network interface components. Each node consists of 933 MHz Intel Pentium III processor, four 256 M SDRAM, 15.3GB IBM DTLA-307015 DeskStar disk, and an Intel 82559 Ethernet Pro 100 onboard Ethernet controller and is connected to a group of sensor resistors on a circuit board. A digital multimeter measures the voltage on each resistor taking 4 measurements per second. The data measured by the meters are further logged and processed.

Li et al. [76] attributed power consumption to CPU, memory, and disk utilization. OS routine power characterization is provided based on the observation that power is strongly correlated with the instructions per cycles (IPC) and a linear regression model is proposed. The processor, 0.18-micron processing technology, runs at 900 MHz frequency and 2.0 V supply voltage. The authors used SPECjvm98, SPECint95, and PostgreSQL benchmarks in the experiments. The estimation model profiles power consumption at the routine level with less than 10% error.

Rivoire et al. [77], [78] study and compare five full-system real-time power models using a variety of machines and benchmarks. Four of these models are utilization-based whereas the fifth includes CPU PMCs in the model variable set along with the utilization of CPU and disk. They report that the PMC-based model is the best overall in terms of accuracy since it accounted for the majority of the contributors to the system's dynamic power (especially the memory activity). They also question the generality of their PMC-based model since the PMCs used in their model variable set may not have the same essence and hence not portable across different architectures (Intel, AMD, etc).

A linear model that takes into account CPU utilization and I/O bandwidth is described in [79] to predict the power consumption of a server. Their model can be represented by the following equation:

$$P = C_{base} + C_1 \times U_{CPU} + C_2 \times U_{I/O} \tag{2.7}$$

where C_{base} is the base power of a compute node. U_{CPU} the CPU utilization, $U_{I/O}$ is the I/O bandwidth in MBPS. C_1 and C_2 are the coefficients of the model. The authors use a WattsUp power-meter to obtain external system-level power consumption to calibrate their model.

Dargie et al. [80] use the statistics of CPU utilization to model the relationship between the power consumption of the multicore processor and workload quantitatively. They demonstrate that the relationship is quadratic for a single-core processor and linear for multicore processors.

Another similar non-linear model based on CPU utilization is presented by Jung et al. [81]. The authors proposed Mistral that provides a trade-off between power consumption, performance, and transient costs. The power consolidation manager takes adaptation actions based on the estimation of power consumption returned by the power model. The parameters of the estimation model are tuned during the offline model calibration phase to fit actual power usage measured using a power meter. The experiments were performed on Pentium 4 hosts running at 1.8 GHz. The evaluated benchmarks run on a three-tier servlet version of RUBiS. The authors showed that their models provide an estimation error of approximately 5%.

Tools to Determine PMCs:

Performance events or performance monitoring counters (PMCs) are special-purpose registers provided in modern microprocessors to store the counts of software and hardware activities. We will use the acronym PMCs to refer to software events, which are pure kernel-level counters such as *page-faults*, *context-switches*, etc. as well as micro-architectural events originating from the processor and its performance monitoring unit called the hardware events such as *cache-misses*, *branch-instructions*, etc. They have been developed primarily to aid low-level performance analysis and tuning. Remarkably while PMCs have not been used for performance modeling, over the years, they have become dominant predictor variables for energy predictive modeling. Modern hardware processors provide a large set of PMCs. Consider the Intel Haswell multicore server CPU. On this server, there are a total of 167 PMCs.

Before we review in detail the PMC based models using different ap-

proaches for various processor architectures such as CPUs, GPUs, and other accelerators, we present the popular tools to collect the PMCs.

PAPI [37] provides a standard API for accessing PMCs available on most modern microprocessors. It provides two types of events, *native events* and *present events*. *Native events* correspond to PMCs native to a platform. They form the building blocks for *present events*. A *preset event* is mapped onto one or more native events on each hardware platform. While *native events* are specific to each platform, *preset events* obtained on different platforms can not be compared.

Likwid [38] provides command-line tools and an API to obtain PMCs for both Intel, POWER8, and AMD processors on the Linux OS. It contains a variety of performance measurement and application tunning tools such as likwid-pin and likwid-bench. Furthermore, Likwid is light-weight, which means the performance overheads of Likwid is less than 6000 cycles. The recent stable released version is Likwid 5.0 with support to extract PMCs of accelerators such as GPUs.

For Nvidia GPUs, CUDA Profiling Tools Interface (*CUPTI*) [82] can be used for obtaining the PMCs for CUDA applications. CUPTI provides the following APIs: Activity API, Callback API, Event API, Metric API, and Profiler API. Sample PMCs that can be obtained using these APIs are total instruction count, data rate, memory load, and store counts, cache hits and misses, number of branches instructions and etc.

Intel PCM [83] is used for reading PMCs of core and uncore (which includes the QPI) components of an Intel processor. It is exposed to programmers as a C++ API and is also able to provide energy measurements from Intel on-chip sensors. It can further support the statistical analysis of core frequencies, QPI power, and DRAM activities.

Linux Perf [84] also called *perf_events* can be used to gather the PMCs for CPUs in Linux. It also comes as a performance profiling tool suite including *perf stat*, *perf record*, *perf report*, *perf annotate*, *perf top* and *perf bench*.

Techniques for Selection of PMCs for Linear Energy Predictive Modeling:

A vast majority of PMC based models are linear. We now survey the works using PMCs as model variables in linear power and energy predictive models.

Singh et al. [44] use PMCs provided by AMD Phenom processor. They divide the PMCs into four categories and rank them in the increasing order of correlation with power using the Spearman's rank correlation. Then they select the top PMC in each category (four in total) for their energy prediction model. The authors implemented a power-aware scheduler and evaluated their model's accuracy using OpenMP versions of NAS parallel and SPEC benchmarking suite and showed a median error of 5.8% and 3.9%, respectively.

Goel et al. [45] divide PMCs into event categories that they believe capture different kinds of microarchitectural activity. The PMCs in each category are then ordered based on their correlation to power consumption using the Spearman's rank correlation. The PMCs with less correlation are then investigated by analyzing the accuracy of several models that employ them.

Kadayif et al. [85] present a PMC-based model for predicting the energy consumption of programs on an UltraSPARC platform. The platform provides 30 different PMCs. However, they use only eight and do not specify how they have selected them. The authors use a number of clock cycles to determine the performance of the programs. A further limitation of their methodology is the restriction to read only two events at a time.

Lively et al. [46] employ 40 PMCs in their predictive model. They use an elaborate statistical methodology to select PMCs. They compute the Spearman's rank correlation for each PMC and remove those below a threshold. They compute the principal components (PCA) of the remaining PMCs and select those with the highest PCA coefficients. Bircher et al. [41] employ an iterative linear regression modeling process where they add a PMC at each step and stop until the desired average prediction error is achieved.

Song et al. [47] select a group of PMCs (for their energy model of Nvidia Fermi C2075 GPU) that are strongly correlated to power consumption based on the Pearson correlation coefficient.

Witkowski et al. [48] use PMCs provided by the *Perf* tool for their model. They use the correlation (Pearson correlation coefficient) between a PMC and the measured power consumption and select those PMCs, which have high correlation coefficients. Although they find that the PMCs related to DRAM

have a low correlation with power consumption, they still use them since these variables signify the intensity of DRAM operations, which contributes significantly to power consumption.

Gschwandtner et al. [42] deal with the problem of selecting the best subset of PMCs on the IBM POWER7 processor, which offers over 500 different PMCs. The first manually select a medium number of hardware counters that they believe are prominent contributors to energy consumption. Then they empirically select a subset from their initial selection. Jarus et al. [49] use PMCs provided by the *Perf* tool for their models. The PMCs employed differ for different models and are selected using the two-stage process. In the first stage, PMCs that are correlated 90% or above are selected. In the second stage, stepwise regression with forwarding selection is used to decide the final set of PMCs.

Haj-Yihia et al. [43] start with a set of 23 PMCs (offered by Likwid) based on expert knowledge of the Intel architecture. Then they perform linear regression iteratively where they drop PMCs (one by one) that do not impact the average prediction error of their model. The authors used RAPL power measurements as ground truth and evaluated their models using SPEC CPU benchmarks.

Wu et al. [50] use the Spearman correlation coefficient and PCA to select the subset of PMCs, that are highly correlated with power consumption. Chadha et al. [51] select a particular PMC from the list of PAPI PMCs available for their platform and check if it fits well with the linear regression model. If it does, they select it as a key parameter for their modeling and experimental study. Otherwise, they skip it.

From the literature, we can divide the techniques to select the PMCs into the following three main categories:

1. Techniques that consider all the PMCs with the goal to capture all possible contributors to energy consumption. To the best of our knowledge, we found no research works that adopt this approach. This could be due to several reasons:

    - Gathering all PMCs requires a lot of programming effort and time.

- Interpretation (for example, visual) of the relationship between energy consumption and PMCs is difficult especially when there is a large number of PMCs.
- Dynamic or runtime models must choose PMCs that can be gathered in just one application run.
- Typically, simple models (those with fewer parameters) are preferred over complex models not because they are accurate but because simplicity is considered a desirable virtue.

2. Techniques that are based on a statistical methodology such a correlation and principal component analysis for the selection of PMCs.

3. Techniques that use expert advice or intuition to pick a subset (that may not necessarily be determined in one application run) and that, in experts' opinion, is a dominant contributor to energy consumption.

Energy predictive modelling using PMCs:

One of the first models correlating PMCs to energy values was developed by Bellosa et al. [86]. Their model is based on events such as integer operations, floating-point operations, memory requests due to cache misses, etc. that they believed to strongly correlate with power consumption. To trigger these events, special micro-benchmarks are written and executed for several seconds.

Icsi et al. [87] propose an elaborate methodology to determine component-level power estimates from the access rates of the components, which are based on PMCs. The authors selected a total of 22 strictly collocated energy consuming components. The constructed power model can be represented by following equation:

$$TotalPower = \sum_{i=1}^{22} Power(C_i) + B_{Power} \tag{2.8}$$

Where B_{Power} is the base power of their evaluation platform.

Li et al. [76] propose power models for the operating system (OS) based on their observations of a strong correlation between instructions per cycle

(IPC) and OS routine power. Their results show the power estimation errors of less than 6%.

Lee et al. [88] adopt a statistically rigorous approach to derive regression models using performance events to predict power. The authors first derived a model baseline using statistical methods. The initial baseline is further refined using different sampled observations. Their results showed a median and maximum error rate of 4.3% and 24.5%, respectively.

Powell et al. [89] use a linear regression model to estimate activity factors and power for a large number of micro-architectural structures using a small number of PMCs that represent utilization statistics such as IPC and load rate. The evaluation against SPEC CPU benchmarks for their power model showed an average error of 8%.

Goel et al. [45] derive per-core power models using PMC values and temperature readings. They evaluated their models on a four and eight-core machine using SPEC OMP, NAS, and SPEC 2006 benchmarking suites and showed that the prediction error lies in between 1.2% to 4.4%.

Bertran et al. [90] present a power model that provides a per-component power breakdown of a multicore CPU. Their model is based on activity factors obtained from PMCs for various components in a multicore CPU.

Bircher et al. [41] propose an iterative modeling procedure to predict power using PMCs. They use PMCs that trickle down from the processor to other subsystems such as CPU, disk, GPU, etc and PMCs that flow inward into the processor such as Direct Memory Access (DMA) and I/O interrupts.

Rotem et al. [30] present a software power model, which eventually became *RAPL*, in Intel Sandybridge. This model predicts the energy consumption of core and uncore components (QPI, LLC) based on some PMCs (which are not disclosed).

Lastovetsky et al. [10] present an application-level energy model where the dynamic energy consumption of a processor is represented by a function of problem size. Unlike PMC-based models that contain hardware-related PMCs and do not consider problem size as a variable, this model takes into account the highly non-linear and non-convex nature of the relationship between energy consumption and problem size for solving optimization problems

of data-parallel applications on homogeneous multicore clusters for energy.

Notable Energy Predictive Models for Software and Hardware Accelerators

We now survey a few notable research works that have proposed energy predictive models for accelerators such as GPU, Xeon Phi, and FPGA.

Hong et al. [91] propose an energy prediction model for an Nvidia GPU similar to the PMC-based unit power prediction approach of [87]. The authors calculated the power consumption as a sum of individual power consumption of all the components that compose a Streaming Multiprocessor (SM) and GDDR memory. They evaluated the accuracy of their model on NVIDIA GTX280 GPU platform. The results show the power prediction errors of 8.94% and energy savings up to 10.99%. The main limitation of their model is the lack of portability because it takes as an input the detailed architectural specifications.

Nagasaka et al. [92] present a statistical approach that uses GPU performance counters exposed for CUDA applications to predict the power consumption of GPU kernels. Their models are evaluated using NVIDIA GeForce GTX 285 GPU and resulted in an average error and maximum error of 4.7% and 23% to predict the total power consumption, respectively.

Song et al. [47] propose power and energy prediction models that employ a configurable, back-propagation, artificial neural network (BP-ANN). The parameters of the BP-ANN model are ten carefully selected PMCs of a GPU. The values of these PMCs are obtained using the CUDA Profiling Tools Interface (CUPTI) [82] during the application execution.

Shao et al. [93] construct an instruction-level energy model of a Xeon Phi processor. They studied the scalability of processor cores considering the energy per instruction. The authors evaluated their model using the Linpack benchmark suite and show an energy gain of 10%.

Khatib et al. [94] propose a linear instruction-level model to predict dynamic energy consumption for soft processors in FPGA. The model considers both inter-instruction effects and the operand values of the instructions. The authors

evaluate the prediction accuracy of their model using different benchmarks and show that it has an average error of 4.7%.

Energy Predictive Models for HPC Applications

In [95], authors presented a power model for parallel scientific applications by using PMCs. Their work extends [87] for Intel-based processors. The authors modelled the power of all the components using a linear function composed of access rates of various processor blocks. The sum of all the power-consuming blocks gives the total power.

Dongarra et al. [96] evaluated the energy consumptions of HPC applications from LAPACK benchmark suite using Intel RAPL. The authors argue that the RAPL API serves as a competitive alternative to power meters.

In [48] and [49], authors propose system-wide power prediction models for HPC servers based on performance counters. They cluster real-life HPC applications into groups and create specialized power models for them.

Gschwandtner et al. [42] present linear regression models based on hardware counters for prediction of energy consumption of HPC applications executing on IBM POWER7 processor. They pick a small subset from 500 different hardware counters offered by the POWER7 processor. The authors report a maximum prediction error of 15%.

Wu et al. [50] present a PMC-based energy predictive model for HPC application workloads.

In [97], authors develop CPU and DIMM power and energy models using artificial neural networks. They study three important HPC kernels, matrix multiplication, stencil computation, and LU factorization. To derive component-level power measurements, they use the PowerMon2 apparatus. They report an absolute error rate of 5.5% for the total power consumption and energy usage predictions for the three kernels.

Cabrera et al. [98] presented an analytical energy model for high-performance Linpack benchmarks to provide the energy consumption estimation of a Linpack application before its execution. Because of the ability to predict the energy consumption of an application beforehand, the authors demon-

strate that their model can be integrated in the operating systems schedulers.

Important Surveys on Energy Predictive Models

Mobius et al. [99] present a survey of power consumption models for single-core and multicore processors, virtual machines, and servers. They conclude that linear regression-based approaches dominate and that one prominent shortcoming of these models is that they use static instead of variable workloads for training the models.

Dayarathna et al. [100] present an in-depth survey on data center power modeling. They organize power models based on two classifications:

- Hardware-centric
- Software-centric

Bridges et al. [101] present a survey of techniques to monitor and model the energy consumption of GPUs. They cover in-depth PMC-based modeling of GPUs. They also state that the accuracy of results from internal power meters must be thoroughly verified using external power meters.

O'Brien et al. [40] survey predictive power and energy models focusing on the highly heterogeneous and hierarchical node architecture in modern HPC computing platforms. The authors studied the accuracy of linear regression-based energy predictive models using applications from Intel math kernel libraries such as DGEMM and FFT.

Francisco et al. [102] presented a detailed survey on the energy measurements tools for high-performance ultrascale computing at hardware and software levels. The authors highlighted that the energy measurements capabilities differ on a number of platforms where available and therefore the community still lacks the availability of a standard.

Critiques of PMCs for Energy Predictive Modelling

However, there are research works that have critically examined and highlighted the poor prediction accuracy of PMCs for energy predictive modeling.

Economou et al. [52] highlight the practical limitation, which is the inability to obtain all the PMCs simultaneously or in one application run. They also mention the lack of PMCs to model the energy consumption of disk I/O and network I/O.

McCullough et al. [53] evaluate the competence of predictive power models for modern node architectures and show that linear regression models show prediction errors as high as 150%. They suggest that direct physical measurement of power consumption should be the preferred approach to tackle the inherent complexities posed by modern node architectures.

Hackenberg et al. [35] present a study of various power measurement strategies, which includes *Intel RAPL* [30]. They report that the accuracy of *RAPL* depends on the type of workload and is quite poor for workloads that use the hyper-threading feature. They also report that the accuracy is poor for applications with small execution times and becomes better only for applications with longer execution times since the predictions are energy averages.

O'Brien et al. [40] survey predictive power and energy models focusing on the highly heterogeneous and hierarchical node architecture in modern HPC computing platforms. Using a case study of PMCs, they highlight the poor prediction accuracy and ineffectiveness of models to accurately predict the dynamic power consumption of modern nodes due to the inherent complexities (contention for shared resources such as Last Level Cache (LLC), NUMA, and dynamic power management). Their results show the poor prediction accuracy of PMC based models for HPC applications with an average error of up to 64%.

2.4 Energy Consumption Optimization Approaches in Computing

In this section, we review some of the state-of-the-art energy minimization techniques on modern computing platforms. There are two broad categories of energy optimization techniques in computing:

1. **System-level and component-level optimization techniques:** Techniques to tweak and tune different operating parameters such as frequency, voltage, number of cores, cache size with an aim to reduce the energy consumption of a computing system at a system level.

2. **Application-level optimization techniques:** While microarchitectural and chip-design advancements have been the leading providers of energy savings, application-level energy optimization strategies emerged as a promising approach. This approach takes into account the application-level parameters such as data partitioning, algorithmic cost, communication cost, etc to reduce the energy consumption for the execution of applications on a given platform.

2.4.1 System-level and Component-level Optimization

Following are the three main approaches to optimize energy consumption at a system level:

1. **Dynamic voltage and frequency scaling (DVFS):** This technique is based on setting the operating frequency of a system or a component such as a socket according to the workload to obtain the optimal performance and energy-saving configurations [103, 104, 105, 106, 107, 108].

2. **Dynamic power management (DPM):** Techniques that use processor reconfiguration in terms of the number of cores, cache and memory sizes, operating frequency, and voltage to obtain the power saving setting [109, 110, 111, 112, 113, 114, 115].

3. **Component-level management (CPM):** Techniques that deal with optimizing the energy of the sub-components of the processor such as like main memory, caches, and I/O, etc [116, 117, 118, 119, 120, 121, 122].

2.4.2 Application-level Optimization

In this thesis, we focus on application-level energy optimization techniques. We now present some of the works in this area.

Demmel et al. [123] present the energy optimization method at the algorithm level using a dense matrix-multiplication and n-body simulation problem. The conducted the experiments on an Intel Atom CPU and a GPU platform. Their results show the existence of a strong scaling in energy. They show that the GPU workloads execute faster but consume more energy consumption so there always exists a trade-off between power and computation intensity.

Alberto et al. [124] presented a generic heuristic based approach to improve the energy consumption in iterative parallel algorithms with the dynamic load balancing. The authors demonstrated using detailed experimentation that the proposed solution method attain less energy consumption than the homogeneous workload balance.

Choi et al. [125] present an extension of the roofline model for energy. Their model represents the energy consumption of the algorithms as a function of the number of operations, level of concurrency, and data flow from the memory. The authors show that a 28 nm GPU consumes less static power than a 40 nm GPU. The results show that floating-point operations are a useful metric to represent the energy-consuming activities on a system.

Alessi et al. [126] present an extension of OpenMP called OpenMPE to tackle the power management of the shared memory processors. The authors demonstrate the use of OpenMPE to achieve multi-objective optimizations for applications using popular techniques such as dynamic voltage and frequency scaling.

Silva at al. [127] study the energy-aware frequency tuning and an optimal number of active cores for inter-node high-performance computing applications. The authors use an application-agnostic power model for their workloads. The power model is build using Complementary Metal-oxide-semiconductor (CMOS) logic designs as a function of the operating frequency.

Wang et al. [128] present an energy optimization method for single-chip heterogeneous processors (SCHP). The authors show that the overall optimization of applications and their partitioning between the CPU and accelerators such as GPU outperforms the optimization of workload partitioning by 13%.

The works reviewed above do not consider workload distribution as a de-

cision variable. We now survey some of the works that consider workload partitioning using problem size as a decision variable to obtain energy savings.

Lastovetsky et al. [129], [9] discovered that the performance profiles of the applications on a Xeon Phi processor are not smooth. The authors study the variations in performance profiles for a real-life data-parallel scientific application, that is, Multidimensional Positive Definite Advection Transport Algorithm (MPDATA). The authors discovered that load imbalancing can be used to optimize the workload partitioning and minimizing the computation time of its parallel execution.

To further enhance the work in this area, Lastovetsky et al. [130], Reddy et al. [27], and Khaleghzadeh et al. [131] presented theoretical works that provide the data partitioning algorithms by keeping time and energy of computations as the objective parameters. The authors demonstrated that in the multicore era, load balancing is no longer synonymous with optimization. Furthermore, they outline recent methods and algorithms for the optimization of parallel applications for performance and energy on modern computing platforms, which do not rely on load balancing and often return imbalanced but optimal solutions.

2.5 Summary

Energy consumption is a leading design constraint along with performance in all computing settings. Two main approaches to enhance energy efficiency in computing are: (1). hardware approach, and (2). software approach. Software-based energy efficiency is achieved at two levels: (1). system-level and (2). application-level. While the system-level optimization approaches are largely driven by hardware innovations in manufacturing energy-efficient devices, the application-levels solution methods include optimization of applications by using application-level decision variables and predictive models for performance and energy consumption of applications. In our work, we focused exclusively on the application-level approach.

Accurate and reliable measurement of energy consumption for an application execution is significant to energy optimization at an application level. The three main approaches to provide it are: (a). System-level physical measurements by the use of power meters, (b). Measurements of current and voltage by using on-chip power sensors and (c). Energy predictive models. Physical power-meter based measurements are accurate but they cannot provide fine-grained component level energy decomposition which is a crucial input for application-level optimization methods. On-chip sensor-based measurement approaches can provide the decomposition of energy consumption at a component level but are inaccurate for application-level energy measurements. However, we found that the sensor readings are deterministic and reproducible and therefore can be used as model variables in energy predictive models.

Energy predictive models have emerged as a top-notch alternative to estimate the energy consumption of an application. Initial energy predictive models use utilization based parameters such as model variables to capture all the energy-consuming activities in a platform. However, previous researches show that these pure utilization models were inaccurate. A vast majority of such models are linear and employ performance monitoring counters (PMCs) as predictor variables. PMCs are special-purpose registers provided in modern microprocessors to store the counts of software and hardware activities. A pervasive approach is to determine the energy consumption of a hardware component based on the linear regression of the PMC counts in the component during an application run. The total energy consumption is then calculated as the sum of these individual consumptions. While the models allow determination of fine-grained decomposition of energy consumption during the execution of an application, there are research works highlighting their poor accuracy [52, 53, 40, 36].

While the main advantage of a software energy predictive model is a determination of fine-grained decomposition of energy consumption during the execution of an application at less cost compared to the ground truth approach using power meters, this approach suffers from following serious drawback:

- Complexity of model construction and lack of consensus among the re-

search works, which report prediction accuracies ranging from poor to excellent.

- A vast majority of research works select PMCs solely on the basis of their high positive correlation with energy consumption without any deep understanding of the physical significance of the model variables.
- A sound theoretical framework to understand the fundamental significance of the model variables with respect to the energy consumption and the causes of inaccuracy or the reported wide variance of the accuracy of the models is lacking.

Chapter 3

A Comprehensive Study on the Accuracy of State-of-the-art Energy Predictive Models on Multicore CPUs

Accurate measurement of energy consumption during an application execution is key to energy minimization techniques at the software level. There are three popular approaches to providing it: (a) System-level physical measurements using external power meters, (b) Measurements using on-chip power sensors, and (c) Energy predictive models.

In this chapter, we present a comparative study on the prediction accuracy of linear energy predictive models employing performance monitoring counters (PMCs) as predictor variables with system-level energy measurements and on-chip sensors (Intel RAPL) for multicore CPUs [1]. The rest of this chapter is divided into three main sections. In Section 3.1, we explain our experimental setup with details on the experimental platforms, system-level energy measurement methods, list of applications in the test suite, and PMC gathering methodology. In Section 3.2, we present our experimental results. Finally Section 3.3 present the summary of results, learned lessons and recommen-

[1] This chapter is chiefly based on [36].

dations for future research directions.

3.1 Experimental Setup

In this section, we explain our experimental setup and energy measurement methodology.

3.1.1 Experimental platforms

We employ two nodes for our comparative study:

- HCLServer1 has an Intel Haswell multicore CPU having 24 physical cores with 64 GB main memory and
- HCLServer2 has an Intel Skylake multicore CPU consisting of 22 cores and 96 GB main memory.

Detailed specifications for both servers are given in Table 3.1. These nodes are representative of computers used in cloud infrastructures, supercomputers, and heterogeneous computing clusters.

Each node has a power meter installed between its input power sockets and the wall A/C outlets.

3.1.2 System-Level Physical Measurements Using Power Meters

HCLServer1 and HCLServer2 are connected with a *Watts Up Pro* power meter. *Watts Up Pro* power meters are periodically calibrated using the ANSI C12.20 revenue-grade power meter, Yokogawa WT310. We present our detailed methodology on calibration in Appendix D.

The maximum sampling speed of *Watts Up Pro* power meters is one sample every second. The accuracy specified in the data-sheets is $\pm 3\%$. The minimum measurable power is 0.5 watts. The accuracy at 0.5 watts is ± 0.3 watts.

Table 3.1: Specification of the Intel Haswell (HCLServer1) and Intel Skylake (HCLServer2) multicore CPU Server

Hardware Specifications	**Intel Haswell Server (HCLServer1)**	**Intel Skylake Server (HCLServer2)**
Processor	Intel E5-2670 v3 @2.30GHz	Intel(R) Xeon(R) Gold 6152
Micro-architecture	Haswell	Skylake
Thread(s) per core	2	2
Cores per socket	12	N/A
Socket(s)	2	1
NUMA node(s)	2	1
L1d cache	32 KB	32 KB
L11 cache	32 KB	32 KB
L2 cache	256 KB	1024 KB
L3 cache	30720 KB	30976 KB
Main memory	64 GB DDR4	96 GB
TDP	240 W	140 W
Idle Power	58 W	32 W
Software Specifications		
OS release	CentOS 7	Ubuntu 16.04 LTS
Linux kernel	3.10	3.10
OpenMP version	3.1	3.1
MPI version	3.2.1	N/A
Compiler	gcc 4.8.5	gcc 4.8.5
Python version	3.4.3	3.6.8
Likwid version	4.1	4.3.2
Intel MKL Version	2017.0.2	2017.0.2

We use the HCLWattsUp interface [132] to obtain the power measurements from the WattsUp Pro power meters. The interface and the methodology used to obtain a data point are explained in Appendix A.2.

We follow a statistical methodology (Appendix A.3) to ensure the reliability of our experimental results. The methodology determines a sample mean (execution time or dynamic energy or PMC) by executing the application repeatedly until the sample mean meets the statistical confidence criteria (95% confidence interval, a precision of 0.025 (2.5%)) Student's *t*-test is used to determine the sample mean. The test assumes that the individual observations are independent and their population follows the normal distribution. We use Pearson's chi-squared test to ensure that the observations follow the normal distribution.

We follow a detailed experimental methodology to obtain reliable component level energy consumption measurements using HCLWattsUp API and Intel RAPL presented by Fahad et al. [36]. We present it in Appendix A.3.1 and A.3.2.

3.1.3 Methodology to Obtain PMCs on HCLServers

We compare in this section the prediction accuracy of linear energy predictive models employing performance monitoring counters (PMCs) as predictor variables with HCLWattsUp and Intel RAPL.

Modern computing platforms such as multicore CPUs provide a large set of PMCs. The most popular tools that can be used to gather the values of the PMCs for a platform include Likwid [38], PAPI [37], Intel PCM [83], and Linux *perf* [84]. The programmers, however, can obtain only a small number of PMCs (typically 3-4) during an application run due to the limited number of hardware registers dedicated to storing them. Consider, for example, the Intel Haswell server whose specification is shown in Table 3.1. *Likwid* tool provides 167 PMCs for this platform. To obtain the values of the PMCs for an application, the application must be executed about 53 times since only a limited number of PMCs can be obtained in a single application run.

There are three main restrictions that make it difficult for the process of

employing PMCs as a predictor variable in models. First, there is a large number of PMCs to consider. In a typical Intel Haswell architecture (see Table 3.1), there are 167 PMCs offered by Likwid tool. Second, a lot of programming effort and time are required to automate and collect all the PMCs. This is because of the limited number of hardware registers available on platforms for storing the PMCs. In a single run of an application, only 3-4 PMCs can be collected. Third, a model purely based on PMCs lacks portability. The reason is that all the PMCs available on a CPU platform may not necessarily be available on a GPU platform. This makes the process of collecting a suitable subset of PMCs critical. The main techniques used to select PMCs for modeling can be divided into the following four categories:

- Techniques that consider all the PMCs offered for a computing platform with the goal to capture all possible contributors to energy consumption. To the best of our knowledge, we found no research works that adopt this approach because of the models' complexities.
- Techniques using a statistical methodology such as correlation, principal component analysis (PCA), and so forth. to choose a suitable subset [133, 134].
- Techniques that use expert advice or intuition to pick a subset of PMCs and that, in experts' opinion, are dominant contributors to energy consumption [43].

Table 3.2 shows the list of applications employed in our experimental suite. The application suite contains highly optimized memory bound and compute-bound scientific routines such as DGEMM and FFT from Intel Math Kernel Library (MKL), benchmarks from NASA Application Suite (NAS), Intel HPCG, *stress*, *naive* matrix-matrix multiplication and *naive* matrix-vector multiplication. The reason to select a diverse set of applications is to avoid bias in our models and to have a range of PMCs for different executions of diverse applications.

For a given application, we measure three quantities during its execution on our platforms. First, is the dynamic energy consumption provided

Table 3.2: List of Applications

Application	Description
MKL FFT	Intel optimized 2-dimensional Fast Fourier Transform
MKL DGEMM	Intel optimized 3-dimensional Dense Matrix Multiplication
HPCG	Intel optimized High Performance Conjugate Gradient. 3-dimensional regular 27-point discretization of an elliptic partial differential equation
NPB IS	Integer Sort, Kernel for random memory access that sort small integers using the bucket sort technique
NPB LU	Lower-Upper Gauss-Seidel solver
NPB EP	Embarrassingly Parallel random number generator
NPB BT	Solve synthetic system of nonlinear partial differential equations using Block Tri-diagonal solver
NPB MG	Approximate 3-dimensional discrete Poisson equation using the V-cycle Multi Grid on a sequence of meshes
NPB FT	Discrete 3-dimensional fast Fourier Transform
NPB DC	Data Cube
NPB UA	Unstructured Adaptive mesh solving heat equation with convection and diffusion from moving ball.
NPB CG	Conjugate Gradient
NPB SP	Scalar Penta-diagonal solver
NPB DT	Data traffic
stress	CPU, disk and I/O stress
Naive MM	Naive Matrix-matrix multiplication
Naive MV	Naive Matrix-vector multiplication

by HCLWattsUp API [132] using the methodology explained in Section A.3.1. Second, we measure the execution time. Lastly, we collect all the PMCs available on our platforms using Likwid tool [38].

Likwid can be used using a simple command-line invocation as given below where the *EVENTS* represents PMCs (4 at maximum in one invocation) of the given application, *APP*:

likwid-perfctr -f -C S0:0-11@S1:12-23 -g EVENTS APP

The application (APP) during its execution is pinned to physical cores (0–11, 12–23) of our platform. Likwid use *likwid-pin* to bind the application to the cores on any platform and lack the facility to bind an application to memory. Therefore, we have used *numactl*, a command-line Linux tool to pin our applications to available memory blocks.

For Intel Haswell and Intel Skylake platform, Likwid offers 164 PMCs and 385 PMCs, respectively. We eliminate PMCs with counts less than or equal to 10 since we found them to have no physical significance for modeling the

dynamic energy consumption and they are non-reproducible over several runs of the same application on our platforms.

The reduced set contains 151 PMCs for Intel Haswell and 323 for Intel Skylake. As in a single application run, we can collect only 4 PMCs, the processor of PMC collection is tedious. Moreover, some PMCs can only be collected individually or in sets of two or three for an application run. Therefore, we observe that each application must be executed about 53 and 99 times on Intel Haswell and Intel Skylake platform, respectively, to collect all the PMCs.

3.2 Accuracy of Linear Energy Predictive Models and Limitations

To facilitate clarity of exposition, the mathematical form of the linear regression models can be stated as follows: $\forall a = (a_k)_{k=1}^{n}, a_k \in \mathbb{R}$,

$$f_{_E}(a) = \beta_0 + \beta \times a = \sum_{k=1}^{n} \beta_k \times a_k \tag{3.1}$$

where β_0 is the intercept and $\beta = \{\beta_1, ..., \beta_n\}$ is the vector of coefficients (or the regression coefficients) In real life, there usually is stochastic noise (measurement errors) Therefore, the measured energy is typically expressed as

$$\tilde{f}_{_E}(a) = f_{_E}(a) + \epsilon \tag{3.2}$$

where the error term or noise ϵ is a Gaussian random variable with expectation zero and variance σ^2, written $\epsilon \sim \mathcal{N}(0, \sigma^2)$.

We now divide our experiments into the following two classes, Class A and Class B, as follows:

1. Class A: In this class, we study the accuracy of platform-level linear regression models using a diverse set of applications.

2. Class B: In this class, we study the accuracy of application-specific linear regression models.

3.2.1 Class A: Accuracy of Platform-Level Linear PMC-Based Models

We select Intel Haswell multicore CPU platform (Table 3.1) for this class of experiments. We select the PMCs commonly used by these models and they are listed below:

- IDQ_MITE_UOPS (X_1)
- IDQ_MS_UOPS (X_2)
- ICACHE_64B_IFTAG_MISS (X_3)
- ARITH_DIVIDER_COUNT (X_4)
- L2_RQSTS_MISS (X_5)
- FP_ARITH_INST_RETIRED_DOUBLE (X_6)

These PMCs count floating-point and memory instructions and are considered to have a very high positive correlation with energy consumption. Table 3.3 shows the correlation of the PMCs with the dynamic energy consumption.

Table 3.3: Correlation of performance monitoring computers (PMCs) with dynamic energy consumption (E_D). Correlation matrix showing relationship of dynamic energy with PMCs. 100% correlation is denoted by 1.

	E_D	X_1	X_2	X_3	X_4	X_5	X_6
E_D	1	0.53	0.50	0.42	0.58	0.99	0.99
X_1	0.53	1	0.41	0.25	0.39	0.45	0.44
X_2	0.50	0.41	1	0.19	0.99	0.48	0.48
X_3	0.42	0.25	0.19	1	0.21	0.41	0.40
X_4	0.58	0.39	0.99	0.21	1	0.57	0.56
X_5	0.99	0.45	0.48	0.41	0.57	1	0.99
X_6	0.99	0.44	0.48	0.40	0.56	0.99	1

We used all the applications listed in Table 3.2 with different configurations of problem sizes to build a data-set of 277 points. Each point represents the data for one application configuration containing its dynamic energy consumption and the PMC counts. We split this data-set into two subsets, one for

training (with 227 points) the models and the other to test (50 points) the accuracy of models. We used this division based on best practices and experts' opinions in this domain.

Using the dataset, we build 6 linear models {A, B, C, D, E, F} using regression analysis. Model A employs all the selected PMCs as predictor variables. Model B is based on five best PMCs with the least energy correlated PMC (X_3) removed. Model C uses four PMCs with two least correlated PMCs (X_2, X_3) removed, and so on until Model F, which contains just one the most correlated PMC (X_6).

The models are summarized in the Table 4.6. We also show the minimum, average, and maximum prediction errors of RAPL.

We will now focus on the minimum, average, and maximum prediction errors of these models. They are (2.7%, 32%, 99.9%) respectively for Model A. Model B based five most correlated PMCs has prediction errors of (0.53%, 21.80%, 72.9%) respectively. The average prediction error significantly dropped from 32% to 21%. The prediction errors for Model C are (0.75%, 29.81%, 77.2%) respectively. The average prediction error, in this case, is in between that of Model A and Model C. Model F with just one most correlated PMC (X_6) has the least average prediction error of 14%. The prediction errors of RAPL are (4.1%, 30.6%, 58.9%) From these results, we conclude that selecting PMCs using correlation with energy does not provide any consistent improvements in the accuracy of linear energy predictive models.

Table 3.4: Linear predictive models (A-F) with intercepts and RAPL with their minimum, average and maximum prediction errors.

Model	PMCs	Intercept Followed by Coefficients	Percentage Prediction Errors [min, avg, max]
A	$X_1, X_2, X_3, X_4, X_5, X_6$	3×10^{-9}, 1.9×10^{-8}, 3.3×10^{-7}, -1×10^{-6}, 6×10^{-8}, -9.3×10^{-11}, 10	(2.7, 32, 99.9)
B	X_1, X_2, X_4, X_5, X_6	3×10^{-9}, 1.9×10^{-8}, -1×10^{-6}, 6.2×10^{-8}, -1.2×10^{-10}, 230	(0.53, 21.80, 72.9)
C	X_1, X_4, X_5, X_6	3.7×10^{-9}, 7.9×10^{-9}, 7.5×10^{-8}, -5.1×10^{-10}, 270	(0.75, 29.81, 77.2)
D	X_4, X_5, X_6	6.7×10^{-8}, 9.4×10^{-8}, -9.7×10^{-10}, 490	(0.21, 23.19, 80.42)
E	X_5, X_6	9.7×10^{-8}, -1.02×10^{-9}, 520	(2, 21.03, 83.40)
F	X_6	1.5×10^{-9}, 740	(2.5, 14.39, 34.64)
RAPL			(4.1, 30.6, 58.9)

We also identified a few more causes of inaccuracy in linear regression-based models by looking at the coefficients of PMCs employed in them. Salient

observations of these models are outlined below:

- All the models have a significant intercept (β_0) Therefore, the model would give predictions for dynamic energy based on the intercept values even for the case when there is no application executing on the platform, which is erroneous. We consider this to be a serious drawback of existing linear energy predictive models (Chapter 2), which do not understand the physical significance of the parameters with dynamic energy consumption.

- Model A has negative coefficients ($\beta = \{\beta_1, ..., \beta_6\}$) for PMCs, X_4 and X_6. Similarly, Model B has negative coefficients for PMC X_4 and X_6. and in Models C-E, X_6 has a negative coefficient. The negative coefficients in these models can give rise to negative energy consumption predictions for specific applications where the counts for X_4 and X_6 are relatively higher than the other PMCs.

3.2.2 Class B: Accuracy of Application-Specific PMC-Based Models

In this section, we study the accuracy of application-specific energy predictive models built using linear regression. We choose a single-socket Intel Skylake server (Table 3.1) for the experiments. We choose two highly optimized scientific kernels: Fast Fourier Transform (FFT) and Dense Matrix-Multiplication application (DGEMM), from Intel Math Kernel Library (MKL).

We select six PMCs (Y1-Y6) listed in the Table 5.9, which have been employed as predictor variables in energy predictive models given in literature (Chapter 2).

We build a dataset containing 362 and 330 points representing DGEMM and FFT for a range of problem sizes from 6400×6400 to 29,504 $\times$ 29,504 and 22,400 $\times$ 22,400 to 41,536 $\times$ 41,536, respectively, with a constant step sizes of 64. We split the dataset into training and test datasets. Training dataset for DGEMM and FFT contains 271 and 255 points used to train the

Table 3.5: Selected PMCs for Class B experiments along with their energy correlation for DGEMM and FFT. 0 to 1 represents positive correlation of 0% to 100%.

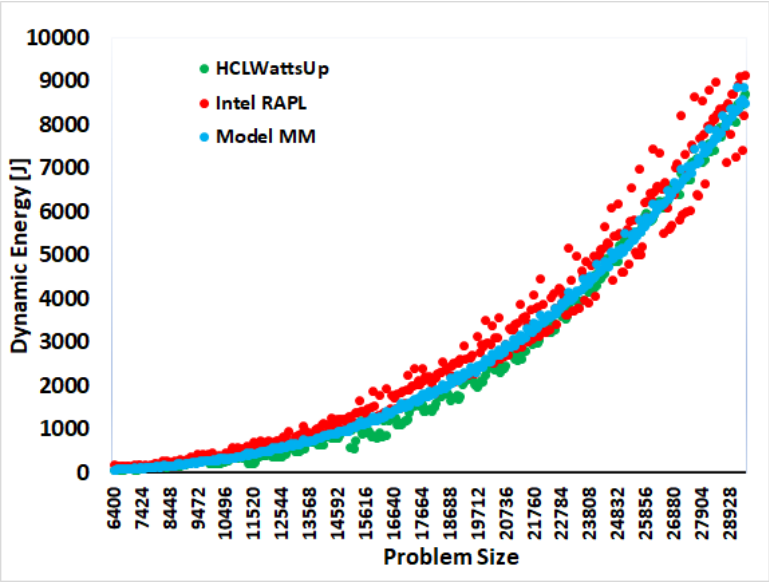
	Selected PMCs	Corr DGEMM	Corr FFT
Y1	FP_ARITH_INST_RETIRED_DOUBLE	0.99	0.98
Y2	MEM_INST_RETIRED_ALL_STORES	0.99	0.99
Y3	MEM_INST_RETIRED_ALL_LOADS	0.98	0.55
Y4	MEM_LOAD_RETIRED_L3_MISS	0.60	0.99
Y5	MEM_LOAD_RETIRED_L1_HIT	0.98	0.34
Y6	ICACHE_64B_IFTAG_MISS	0.99	0.77

energy predictive models. Test dataset contains 91 and 75 points for both applications respectively.

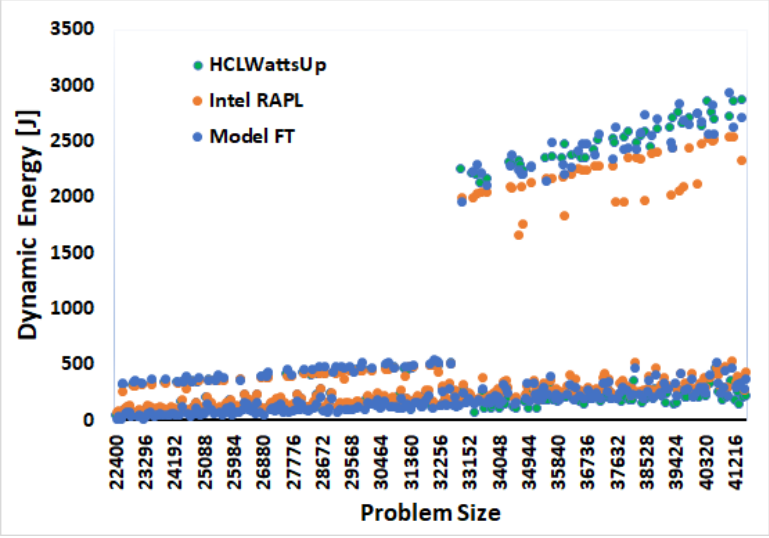
Using the datasets, we build two linear models for both applications. These are *Model MM* and *Model FT*. Figure 3.2a,b shows the percentage deviations of dynamic energy consumption of PMC models and RAPL from HCLWattsUp for DGEMM and FFT, respectively.

Figure 3.1a,b show the dynamic energy profiles of 2D FFT and DGEMM from Intel MKL on HCLServer2 obtained using HCLWattsUp, Intel RAPL and predictive models, respectively. The maximum and the average difference between Intel RAPL and Model MM profiles from HCLWattsUp is 205% and 36.13%, and, 218% and 26%. For FFT, the maximum and the average difference between both profiles from HCLWattsUp is 156.38% and 28.67%, and, 155% and 31%.

Comparing with HCLWattsUp, the minimum, average and maximum error for DGEMM using *Model MM* and RAPL are (0, 26, 218) and (0.4, 35, 161), respectively. In the case of FFT, the minimum, average, and maximum error using *Model FT* and RAPL is (0.8, 27, 147) and (0.3, 31, 155) respectively. We observe that both models perform better in terms of average prediction accuracy than RAPL.

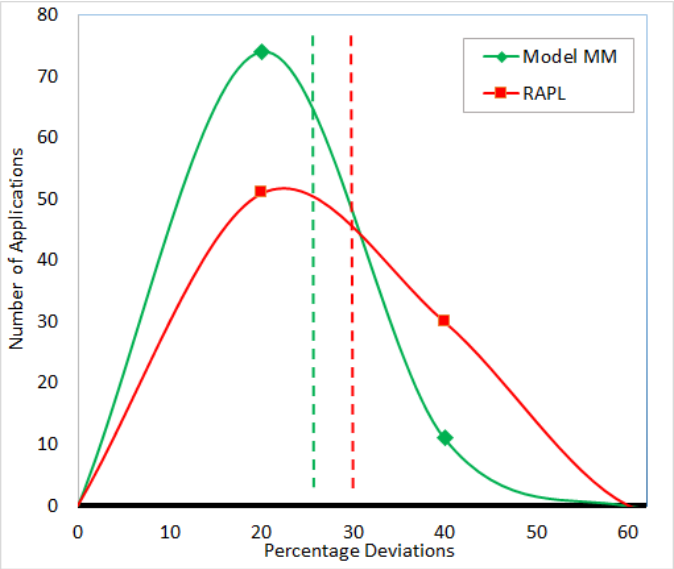


(a) MKL-DGEMM

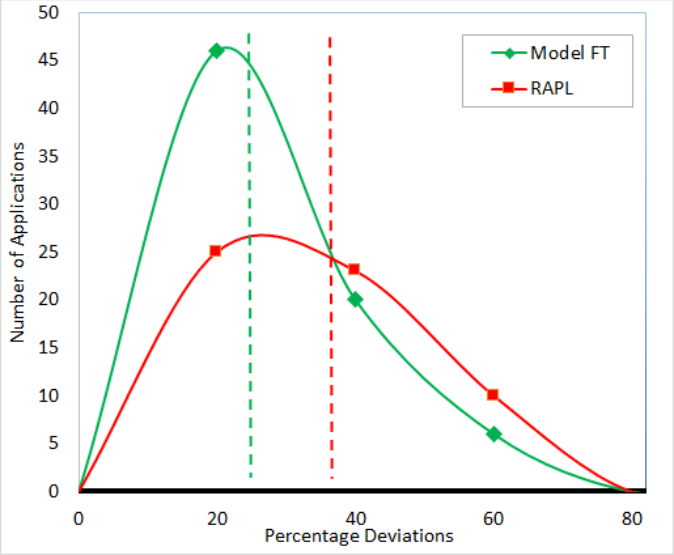


(b) 2D MKL-FFT

Figure 3.1: Dynamic energy consumption of Predictive Models, Intel RAPL and HCLWattsUp on HCLServer2.



(a) Model MM



(b) Model FT

Figure 3.2: Percentage deviations of predictive models and RAPL from HCLWattsUp. The dotted lines represent the averages.

3.3 Summary

In this chapter, we show that energy consumption measurements using Intel RAPL significantly deviate from system-level energy measurements using external power-meters. Therefore, we presented a comprehensive study comparing the accuracy of state-of-the-art energy predictive models (that emerged as a preeminent alternative to on-chip sensors) against system-level physical measurements using external power meters, which we consider to be the ground truth. To compare the approaches reliably, we used a published methodology to determine the accurate component-level dynamic energy consumption of an application using system-level physical measurements using power meters, which are obtained using HCLWattsUp API.

For the study comparing the prediction accuracy of energy predictive models with the ground truth, we use an experimental platform containing a test suite of seventeen benchmarks executed on an Intel Haswell multicore CPU and an Intel Skylake multicore CPU. The average error between energy predictive models employing performance monitoring counters (PMCs) as predictor variables and the ground truth ranges from 14% to 32% and the maximum reaches 100%. The sources of this inaccuracy are the following:

- Model parameters in most cases are not deterministic and reproducible.
- Model parameters are selected chiefly based on statistical correlation with energy and not their physical significance originating from fundamental physical laws such as conservation of energy of computing.

Our experimental results illustrated that methods solely based on correlation with energy to select PMCs are not effective in improving the average prediction accuracy.

We will now state our recommendations and possible future directions. Since system-level physical measurements based on power meters are accurate and the ground truth, we recommend using this approach as the fundamental building block for the fine-grained device-level decomposition of the energy consumption during the parallel execution of an application executing on several independent computing devices in a computer.

We envisage hardware vendors maturing their on-chip sensor technology to an extent where energy optimization programmers will be provided necessary information of how a power measurement is determined for a component, the frequency or sampling rate of the measurements, its reported accuracy and finally how to programmatically obtain this measurement with sufficient accuracy and low overhead.

Linear energy predictive models can be employed in the optimization of applications for dynamic energy provided they meet the aforementioned limitations. In the next chapter, we will present additivity as a selection criterion for PMCs to be used in energy predictive models.

Chapter 4

Energy Predictive Models for Computing: Theory, Practical Implications, and Experimental Analysis on Multicore CPUs

Software energy predictive models emerged as a popular alternative to determine the energy consumption of an application. A vast majority of the models employ Performance events or performance monitoring counters (PMCs) as the model variables. PMCs are special-purpose registers provided in modern microprocessors to store the counts of software and hardware activities. We will use the acronym PMCs to refer to software events, which are pure kernel-level counters such as *page-faults*, *context-switches*, etc. as well as micro-architectural events originating from the processor and its performance monitoring unit called the hardware events such as *cache-misses*, *branch-instructions*, etc. They have been developed primarily to aid low-level performance analysis and tuning. Remarkably while PMCs have not been used for performance modeling, over the years, they have become dominant predictor variables for energy predictive modeling. This chapter proposes a novel selection criterion for PMCs called *additivity*, which can be used to determine the subset of PMCs that can potentially be used for reliable energy predictive

modeling [1].

Modern hardware processors provide a large set of PMCs. Consider the Intel Haswell multicore server CPU whose specification is shown in Table 3.1. On this server, the PAPI tool [37] provides 53 hardware performance events. The Likwid tool [38], [39] provides 167 PMCs. This includes events for uncore (collection of components of a processor not in the core but essential for core performance) and micro-operations (μops) of CPU cores specific to Haswell architecture that are not provided by PAPI. However, all the PMCs can not be determined using a single application run since only a limited number of registers are dedicated to collecting them. For example, to collect all the Likwid PMCs for a single runtime configuration of an application on the server, the application must be executed 53 times. It must be also pointed out that energy predictive models based on PMCs are not portable across a wide range of architectures. While a model based on either Likwid PMCs or PAPI PMCs may be portable across Intel and AMD architectures, it will be unsuitable for GPU architectures.

Therefore, there are three serious constraints that pose difficult challenges to employing PMCs as predictor variables for energy predictive modeling. First, there is a large number of PMCs to consider. Second, a lot of programming effort and time are required to automate and collect all the PMCs. This is because all the PMCs can not be collected in one single application run. Third, a model purely based on PMCs lacks portability. In this chapter, we focus mainly on techniques employed to select a subset of PMCs to be used as predictor variables for energy predictive modeling. We now present a brief survey of them.

O'Brien et al. [40] survey the state-of-the-art energy predictive models in HPC and present a case study demonstrating the ineffectiveness of the dominant PMC-based modeling approach for accurate energy predictions. In the case study, they use 35 carefully selected PMCs (out of a total of 390 available in the platform) in their linear regression model for predicting dynamic energy consumption. [41], [42], [43] select PMCs manually, based on in-depth study of architecture and empirical analysis. [45], [46], [47], [48], [49], [50] select

[1] This chapter is chiefly based on [54] and [135].

PMCs that are highly correlated with energy consumption using Spearman's rank correlation coefficient (or Pearson's correlation coefficient) and principal component analysis (PCA). [41], [49] use variants of linear regression to remove PMCs that do not improve the average model prediction error.

We classify the existing techniques for the selection of PMCs into three categories. The first category contains techniques that consider all the PMCs with the goal to capture all possible contributors to energy consumption. To the best of our knowledge, we found no research works that adopt this approach. This could be due to several reasons: a) Gathering all PMCs requires a lot of programming effort and time; b) Interpretation (for example, visual) of the relationship between energy consumption and PMCs is difficult especially when there is a large number of PMCs; c) Dynamic or runtime models must choose PMCs that can be gathered in just one application run; d) Typically, simple models (those with fewer parameters) are preferred over complex models not because they are accurate but because simplicity is considered a desirable virtue.

The second category consists of techniques that are based on a statistical methodology. The last category contains techniques that use expert advice or intuition to pick a subset (that may not necessarily be determined in one application run) and that, in experts' opinion, is a dominant contributor to energy consumption.

To summarize, PMCs are now the dominant predictor variables for modeling energy consumption. Modern hardware processors provide a large set of PMCs. Determination of the best subset of PMCs for energy predictive modeling is a non-trivial task given the fact that all the PMCs can not be determined using a single application run. Several techniques have been devised to address this challenge. While some techniques are based on a statistical methodology, some use expert advice to pick a subset (that may not necessarily be obtained in one application run) that, in experts' opinion, are significant contributors to energy consumption. However, the existing techniques have not considered a fundamental property of predictor variables that should have been applied in the first place to remove PMCs unfit for modelling energy. We first address this oversight in this chapter (see Section 5.2.1).

While the main advantage of a software energy predictive model is the determination of fine-grained decomposition of energy consumption during the execution of an application at less cost compared to the ground truth approach using power meters, this approach suffers from serious drawbacks such as the complexity of model construction and lack of consensus among the research works, which report prediction accuracies ranging from poor to excellent. Moreover, a vast majority of research works select PMCs solely on the basis of their high positive correlation with energy consumption without any deep understanding of the physical significance of the model variables. In summary, a sound theoretical framework to understand the fundamental significance of the model variables with respect to the energy consumption and the causes of inaccuracy or the reported wide variance of accuracy of the models is lacking. We also bridge the gap in this chapter.

We organize the rest of this chapter as follows. We start with our formal theory of *energy of computing*. We summarize and generalize the assumptions behind the existing work on PMC-based energy predictive modelling. We use a model-theoretic approach to formulate the assumed properties of the existing models in a mathematical form. We extend the formalism by adding properties, heretofore unconsidered, that are basic implications of the universal energy conservation law. The new properties are intuitive and have been experimentally validated. The extended formalism defines our theory of *energy of computing*. We term an energy predictive model satisfying the properties of the extended model a *consistent* energy model. Using the theory, we prove that a consistent energy predictive model is linear if and only if its each PMC variable is *additive* in the sense that the PMC for a serial execution of two applications is the sum of PMCs for the individual execution of each application. Then, we present experimental results followed by the conclusion.

4.1 Energy Predictive Models for Computing: Intuition, Motivation, and Theory

We summarize and generalize the assumptions behind the existing work on PMC-based power/energy modelling. We use a model-theoretic approach to formulate the assumed properties of these models in a mathematical form. Then we extend the formalism by adding properties, which are intuitive and which we have experimentally validated but which have never been considered previously. The properties are manifestations of the fundamental physical law of energy conservation. We introduce two definitions based on the properties of the extended model, called *weak composability* and *strong composability*. We term an energy predictive model satisfying all the properties of the extended model a *consistent* energy model. The extended model and the two definitions define our theory of energy predictive models for computing.

Finally, we mathematically derive properties of *linear* consistent energy predictive models. We prove that a consistent energy model is linear if and only if it is strongly composable with each PMC variable being additive. The practical implication of this theoretical result is that each PMC variable of a linear energy predictive model must be *additive*. The significance of this property is that it can be efficiently tested and hence used in practice to identify PMC variables that must not be included in the model.

4.1.1 Intuition and Motivation

The essence of PMC-based energy predictive models is that an application run can be accurately characterized by a n-vector of PMCs over $\mathbb{R}_{\geq 0}$. Any two application runs characterized by the same PMC vector are supposed to consume the same amount of energy. The applications in these runs may be different but the same computing environment is always assumed. Thus, PMC-based models are computer system-specific.

Based on these assumptions, any PMC-based energy model is formalized by a set of PMC vectors over $\mathbb{R}_{\geq 0}$, and a function, $f_E : \mathbb{R}^n_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$, mapping these vectors in the set to energy values. *No other properties of the set and*

the function are assumed.

In this work, we extend this model by adding properties that characterize the behavior of serial execution of two applications. To aid the exposition, we follow some notation and terminology. A *compound application* is defined as the serial execution of two applications, which we call the *base* applications. If the base applications are A and B, we denote their compound application by $A \oplus B$. We will refer solely to energy predictive models hereafter since there exists a linear functional mapping from PMC-based power predictive models to them. When we say energy consumption, we mean dynamic energy consumption. The energy consumption that is experimentally observed during the execution of an application A is denoted by $E(A)$. The energy consumption of the compound application $A \oplus B$, $E(A \oplus B)$, is the energy consumption that is experimentally observed during the execution of the compound application.

First, we aim to reflect in the model the observation that in a stable and dedicated environment, where each run of the same application is characterized by the same PMC vector, for any two applications, the PMC vector of their serial execution will always be the same. To introduce this property, we add to the model a (infinite) set of applications denoted by $\mathcal{A}$. We postulate the existence of binary operators, $\mathcal{O} = \{\circ_{_{AB,k}} : \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}, A, B \in \mathcal{A}, k \in [1, n]\}$ so that for each $A, B \in \mathcal{A}$ and their PMC vectors $a = \{a_k\}_{k=1}^{n}, b = \{b_k\}_{k=1}^{n} \in \mathbb{R}_{\geq 0}^{n}$ respectively, the PMC vector of the compound application $A \oplus B$ will be equal to $\{a_k \circ_{_{AB,k}} b_k\}_{k=1}^{n}$.

Next, we introduce properties, which are manifestations of the universal energy conservation law. The following property essentially states that doing nothing (signified by a null vector of PMCs, $\mathcal{NULL} = \{0\}_{k=1}^{n} \in \mathbb{R}_{\geq 0}^{n}$) does not consume or generate energy,

$$f_{_E}(\mathcal{NULL}) = 0$$

The following property postulates that an application with a PMC vector that is not $\mathcal{NULL}$ must consume some energy. The intuition behind this property is that since PMCs account for energy consuming activities of applications, an application with any energy consuming activity higher than zero activity (a

$\mathcal{NULL}$ PMC vector), must consume more energy than zero.

$$\forall a \in \mathbb{R}^n_{\geq 0} \wedge a \neq \mathcal{NULL}, \ f_{_E}(a) > 0$$

Finally, we aim to reflect the observation that the consumed energy of compound application $A \oplus B$ is always equal to the sum of energies consumed by the individual applications A and B respectively,

$$E(A \oplus B) = E(A) + E(B) \tag{4.1}$$

To introduce this property in the extended model, we postulate the following,

$$\forall A, B \in \mathcal{A}, a = \{a_k\}^n_{k=1}, b = \{b_k\}^n_{k=1} \in \mathbb{R}^n_{\geq 0}, \ \circ_{_{AB,k}} \in \mathcal{O},$$
$$f_{_E}(\{a_k \circ_{_{AB,k}} b_k\}^n_{k=1}) = f_{_E}(a) + f_{_E}(b)$$

To summarize, while existing models are focused on abstract application runs and lack any notion of applications, we introduce this notion in the extended model. The additional structure introduced in the extended model allows one to prove the mathematical properties of energy predictive models.

4.1.2 Formal Summary of Properties of Extended Model

The formal summary of the properties of the extended model follows:

Property 4.1.1 (Inherited from Basic Model)**.** *An abstract application run is accurately characterized by a set of n-vector of PMCs over $\mathbb{R}_{\geq 0}$. A null vector of PMCs is represented by $\mathcal{NULL} = \{0\}^n_{k=1}$. There exists a function, $f_{_E} : \mathbb{R}^n_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$, mapping the vectors to energy values and $\forall p, q \in \mathbb{R}^n_{\geq 0}, p = q \implies f_{_E}(p) = f_{_E}(q)$.*

Property 4.1.2 (Weak Composability, Applications and Operators)**.** *There exists an application space, $(\mathcal{A}, \oplus)$, where $\mathcal{A}$ is a (infinite) set of applications and $\oplus$ is a binary function on $\mathcal{A}$, $\oplus : \mathcal{A} \times \mathcal{A} \rightarrow \mathcal{A}$. There exists a (infinite) set of binary operators, $\mathcal{O} = \{\circ_{_{PQ,k}} : \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}, P, Q \in \mathcal{A}, k \in [1, n]\}$ so that for each $P, Q \in \mathcal{A}$ and their PMC vectors $p = \{p_k\}^n_{k=1}, q = \{q_k\}^n_{k=1} \in \mathbb{R}^n_{\geq 0}$*

respectively, the PMC vector of the compound application $P \oplus Q$ *will be equal to* $\{p_k \circ_{PQ,k} q_k\}_{k=1}^{n}$.

Property 4.1.3 (Zero Energy, Energy Conservation)**.** $f_{_E}(\mathcal{NULL}) = 0.$

Property 4.1.4 (Positive-definiteness, Energy Conservation)**.** $\forall p \in \mathbb{R}_{\geq 0}^{n} \wedge p \neq \mathcal{NULL},\ f_{_E}(p) > 0.$

Property 4.1.5 (Weak Composability, Energy Conservation)**.** $\forall P, Q \in \mathcal{A},\ p = \{p_k\}_{k=1}^{n}, q = \{q_k\}_{k=1}^{n} \in \mathbb{R}_{\geq 0}^{n},\ \circ_{PQ,k} \in \mathcal{O}, f_{_E}(\{p_k \circ_{PQ,k} q_k\}_{k=1}^{n}) = f_{_E}(p) + f_{_E}(q).$

We term an energy predictive model satisfying all the above properties of the extended model a *consistent* energy model.

4.1.3 Strong Composability: Definition

The definition of *strong composability* of models follows:

Definition 4.1.1 (Strong Composability)**.** *A consistent energy model is* strongly composable *if* $\forall P, Q, R, S \in \mathcal{A}, p = \{p_k\}_{k=1}^{n}, q = \{q_k\}_{k=1}^{n}, r = \{r_k\}_{k=1}^{n}, s = \{s_k\}_{k=1}^{n} \in \mathbb{R}_{\geq 0}^{n}, k \in [1, n], \circ_{PQ,k} = \circ_{RS,k}$.

The *strong composability* property of a model essentially states that binary operators used in the model to compute PMC vectors of compound applications are not application specific. In other words, the set $\mathcal{O}$ consists of only n binary operators, one for each PMC parameter, $\mathcal{O} = \{\circ_k\}_{k=1}^{n}$, so that for any $P, Q \in \mathcal{A}$ and their PMC vectors $p = \{p_k\}_{k=1}^{n}, q = \{q_k\}_{k=1}^{n} \in \mathbb{R}_{\geq 0}^{n}$, the PMC vector of the compound application $P \oplus Q$ will be equal to $\{p_k \circ_k q_k\}_{k=1}^{n}$.

4.1.4 Mathematical Analysis of Linear Energy Predictive Models Based on The theory of Energy Predictive Models for computing

In this section, we mathematically derive properties of *linear* consistent energy predictive models, that is, linear energy models satisfying properties (4.1.1 to 4.1.5).

By definition, a model is *linear* iff $f_{_E}(x)$ is a linear function.

To the best of our knowledge, all the state-of-the-art energy predictive models for multicore CPUs are based on linear regression. While they model total energy consumption, we consider dynamic energy consumption for reasons described previously. The mathematical form of these models can be stated as follows: $\forall p = (p_k)_{k=1}^n, p_k \in \mathbb{R}_{\geq 0}$,

$$f_{_E}(p) = \beta_0 + \beta \times p = \beta_0 + \sum_{k=1}^{n} \beta_k \times p_k \tag{4.2}$$

where β_0 is called the model intercept, the $\beta = \{\beta_1, ..., \beta_n\}$ is the vector of regression coefficients or the model parameters. In real life, there usually is stochastic noise (measurement errors). Therefore, the measured energy is typically expressed as

$$\tilde{f}_{_E}(p) = f_{_E}(p) + \epsilon \tag{4.3}$$

where the error term or noise ϵ is a Gaussian random variable with expectation zero and variance σ^2, written $\epsilon \sim \mathcal{N}(0, \sigma^2)$. We will ignore the noise term in our mathematical proofs to follow.

Theorem 1. *If a linear energy predictive model (4.2) is consistent, the model intercept must be zero and the model coefficients must be positive.*

Proof. From the energy conservation property 4.1.3,

$$\begin{aligned} \mathcal{NULL} = \{0\}_{k=1}^n \in \mathbb{R}_{\geq 0}^n,\ f_{_E}(\mathcal{NULL}) = 0 \\ \implies \beta_0 + \sum_{k=1}^{n} \beta_k \times 0 = 0 \\ \implies \beta_0 = 0 \end{aligned}$$

From the energy conservation property 4.1.4,

$$\begin{aligned}\forall k \in [1,n], p = \{0,...,0,p_k,0,...,0\} \wedge p \neq \mathcal{NULL},\\ f_{_E}(p) > 0\\ \implies \sum_{i=1}^{n} \beta_i \times p_i > 0\\ \implies \beta_k \times p_k > 0\\ \implies \beta_k > 0 \text{ since } p_k > 0\end{aligned}$$

□

To summarize, a linear energy predictive model satisfying energy conservation properties (4.1.3 and 4.1.4) has a zero model intercept and positive model coefficients. Also as we only consider models satisfying property 4.1.3, then the linearity of function $f_{_E}(x)$ can be equivalently defined as follows: for any $\alpha \in \mathbb{R}_{\geq 0}$ and $p, q \in \mathbb{R}^n_{\geq 0}$

$$f_{_E}(p+q) = f_{_E}(p) + f_{_E}(q) \tag{4.4}$$

and

$$f_{_E}(\alpha \times p) = \alpha \times f_{_E}(p) \tag{4.5}$$

Theorem 2. *If a consistent energy model is linear, then it is strongly composable with O = {+}.*

Proof. From properties 4.1.2 and 4.1.5 of *weak composability*, we have

$$\begin{aligned}\forall P, Q \in \mathcal{A}, \forall k \in [1,n], p = \{0,...,0,p_k,0,...,0\},\\ q = \{0,...,0,q_k,0,...,0\} :\\ f_{_E}(\{0,...,0,p_k \circ_{_{PQ,k}} q_k,0,...,0\}) = f_{_E}(p) + f_{_E}(q)\end{aligned}$$

Using the property (4.4) of a linear predictive model,

$$\begin{aligned}
&f_E(p+q) = f_E(p) + f_E(q) \\
&\implies f_E(p+q) = f_{_E}(\{0,...,0,p_k \circ_{_{PQ,k}} q_k, 0,...,0\}) \\
&\implies f_E(\{0,...,p_k+q_k,0,...,0\}) \\
&\quad = f_{_E}(\{0,...,0,p_k \circ_{_{PQ,k}} q_k, 0,...,0\}) \\
&\implies p_k + q_k = p_k \circ_{_{PQ,k}} q_k \quad (\text{from linearity of} f_E(x)) \\
&\implies \circ_{_{PQ,k}} = +
\end{aligned}$$

Therefore, if a consistent energy model is linear, then it is strongly composable with $O = \{+\}$. □

Theorem 3. *If a consistent energy model is strongly composable with O = {+} and function $f_{_E}(x)$ is continuous, then it is linear.*

Proof. First, we prove the first defining linearity property (4.4),

$$f_{_E}(p+q) = f_{_E}(p) + f_{_E}(q)$$

for any $p, q \in \mathbb{R}^n_{\geq 0}$.

As the model is *strongly composable* with $O = \{+\}$, then

$$\forall P, Q \in \mathcal{A}, \forall k \in [1,n] : \circ_{_{PQ,k}} = +$$

From property 4.1.5 of *weak composability*,

$$\begin{aligned}
&f_{_E}(\{p_k \circ_{_{PQ,k}} q_k\}_{k=1}^n) = f_{_E}(p) + f_{_E}(q) \\
&\implies f_{_E}(p) + f_{_E}(q) = f_{_E}(\{p_k \circ_{_{PQ,k}} q_k\}_{k=1}^n) \\
&\implies f_{_E}(p) + f_{_E}(q) = f_{_E}(\{p_k + q_k\}_{k=1}^n) = f_E(p+q)
\end{aligned}$$

This proves the first property of linearity.

We now prove the second defining property of linearity (4.5),

$$f_{_E}(\alpha \times p) = \alpha \times f_{_E}(p)$$

for any $p \in \mathbb{R}^n_{\geq 0}$ and $\alpha \in \mathbb{R}_{\geq 0}$.

For any integer $m > 0$,

$$\begin{aligned} f_E(m \times p) &= f_E(p + p + ... + p) \\ &= f_E(p) + f_E(p) + ... + f_E(p) \\ &= m \times f_E(p) \end{aligned}$$

For any integer $n > 0$,

$$\begin{aligned} f_E(p) &= f_E(\frac{p}{n}) + f_E(\frac{p}{n}) + ... + f_E(\frac{p}{n}) \\ &= n \times f_E(\frac{p}{n}) \\ \implies \frac{1}{n} f_E(p) &= f_E(\frac{q}{n}) \end{aligned}$$

Thus, for any rational $\frac{m}{n} > 0$,

$$\begin{aligned} \frac{m}{n} f_E(q) &= \frac{1}{n} f_E(q) + \frac{1}{n} f_E(q) + ... + \frac{1}{n} f_E(q) \\ &= f_E(\frac{q}{n}) + f_E(\frac{q}{n}) + ... + f_E(\frac{q}{n}) \\ &= f_E(m \times \frac{q}{n}) = f_E(\frac{m}{n} q) \end{aligned}$$

By definition, any real number α is a limit of an infinite sequence of rational numbers. Consider a sequence $\{\alpha_k\}$ of positive rational numbers such that $\lim_{k \to +\infty} \alpha_k = \alpha$. Then,

$$\begin{aligned} f_E(\alpha \times p) &= f_E((\lim_{k \to +\infty} \alpha_k) \times p) \\ &= f_E(\lim_{k \to +\infty} (\alpha_k \times p)) \\ &= \lim_{k \to +\infty} f_E(\alpha_k \times p) \quad (\text{from continuity of} f_E(x)) \end{aligned}$$

As α_k are positive rational numbers, $f_E(\alpha_k \times p) = \alpha_k \times f_E(p)$. Therefore,

$$\begin{aligned} f_E(\alpha \times p) &= \lim_{k \to +\infty} (\alpha_k \times f_E(p)) \\ &= f_E(p) \times \lim_{k \to +\infty} \alpha_k \\ &= f_E(p) \times \alpha \end{aligned}$$

□

An implication of the proof that a consistent energy model is linear if and only if it is strongly composable with O = {+} is that each PMC variable of a linear energy predictive model must be additive.

4.1.5 Discussion

In this section, we present practical implications of our proposed theory of energy predictive models for computing that can be employed to construct accurate and reliable linear energy predictive models and some guidelines, in general, for the design of reliable energy predictive models.

- The basic practical implications of the theory for improving the prediction accuracy of linear energy predictive models are unified in a *consistency* test. The test includes the following selection criteria for model variables, model intercept, and model coefficients:
  - Each model variable must be deterministic and reproducible.
  - Each model variable must be additive in the sense that the value of the model variable for a serial execution of two applications be equal to the sum of its values obtained for the individual execution of each application.
  - The model intercept must be zero.
  - Each model coefficient must be positive.

  The first two properties are combined into an *additivity* test for the selection of PMCs. A linear energy predictive model employing PMCs and

which violates the properties of the consistency test will have poor prediction accuracy.

- By definition and intuition, PMCs are all pure counters of energy-consuming activities in modern processor architectures and as such must be additive. Therefore, according to our theory of energy predictive models for computing, any consistent, and hence accurate, energy model, which only employs PMCs, must be linear. This also means that any non-linear energy model only employing PMCs will be inconsistent and hence inherently inaccurate.

- Thus, we can conclude that in order to be accurate, a non-linear energy model must employ non-additive parameters in addition to PMCs.

- If the prediction accuracy of the best linear energy predictive model satisfying the properties of the *consistency* test is still low, then one must explore the use of non-linear modelling techniques, which employ non-additive model variables that are highly positively correlated with dynamic energy consumption. However, non-linear models must still be consistent (i.e., must satisfy all the properties of the extended model) to be reliable and accurate.

Our experiments apply the practical implications of the theory of energy predictive models for computing to the state-of-the-art energy predictive models to study their prediction accuracy.

4.2 Organization of Experimental Results

We now present our experiments and results to build and validate the effectiveness of consistency test. Furthermore, another motivation of the experiments is to confirm that employment of a consistent and reliable energy model yield better results when employed in the optimization of applications.

We incorporate the implications of our theory of energy predictive models for computing in the state-of-the-art models and study their prediction accuracy

using a strict experimental methodology on two modern Intel multicore servers, HCLServer1, and HCLServer2 (Table 3.1). The experiments are divided into five main groups: *Group 1*, *Group 2*, *Group 3*, and *Group 4*. Group 1 and 2 are performed on HCLServer1. Experiments in Group 3 use HCLServer2. The last group uses both the servers.

The experimental studies in the groups are summarized below:

1. **Group 1:** A study of the additivity of PMCs for compound applications using an additivity test.

2. **Group 2:** A study of the impact of consistency test on prediction accuracy of models built using linear regression.

3. **Group 3:** A study of the impact of consistency test on the accuracy of application-specific energy predictive models. We present an experimental analysis showing the effectiveness of employing the energy conservation property of additivity along with selection methods for PMCs based on a high positive correlation for constructing accurate energy predictive models.

4. **Group 4:** A study of optimization of a parallel matrix-matrix application for dynamic energy using two measurement tools, IntelRAPL [30] which is a popular mainstream tool and system-level physical power measurements using power meters (using the HCLWattsUp interface [132]).

4.3 Group 1: Study of *Additivity* of PMCs

4.3.1 Additivity: Definition

The *additivity* criterion is based on a simple and intuitive rule that the value of a PMC for a compound application is equal to the sum of its values for the executions of the base applications constituting the compound application.

We brand a PMC *non-additive* on a platform if there exists a compound application for which the calculated value significantly differs from the value

observed for the application execution on the platform (within a tolerance of 5.0%). If the experimentally observed PMCs (sample means) of two base applications are $\overline{e_1}$ and $\overline{e_2}$ respectively, then a *non-additive* PMC of the compound application will exhibit a count experimentally that does not lie between $(\overline{e_1} + \overline{e_2}) \times (1 - \epsilon)$ and $(\overline{e_1} + \overline{e_2}) \times (1 + \epsilon)$, where the tolerance, $\epsilon = 0.05$.

If we fail to find a compound application (typically from a large set of diverse compound applications) for which the *additivity* criterion fails, we term the PMC as potentially *additive*, which means that it can potentially be used for reliable energy predictive modeling. By definition, a potentially *additive* PMC must be deterministic and reproducible, that is, it must exhibit the same value (within a tolerance of 5.0%) for different executions of the same application with same runtime configuration on the same platform.

The use of a *non-additive* PMC as a predictor variable in a model renders it inconsistent and therefore unreliable. We explain this point using a simple example. Consider an instance of an energy prediction model that uses a *non-additive* PMC as a predictor variable. A natural and intuitive approach to predict the energy consumption of an application that executes two base applications one after the other is to substitute the sum of the PMCs for the base applications in the model. However, since the PMC is *non-additive*, the prediction would be very inaccurate.

Therefore, using *non-additive* PMCs in energy predictive models adds noise and can significantly damage the predicting power of energy models based on them.

We now present a test to determine if a PMC is *non-additive* or potentially *additive*. We call it the *additivity test*.

4.3.2 Additivity Test

The test consists of two stages. A PMC must pass both stages to be pronounced *additive* for a given compound application on a given platform.

1. In the first stage, we determine if the PMC is deterministic and reproducible.

2. In the second stage, we examine how the PMC of the compound application relates to its values for the base applications. At first, we collect the values of the PMC for the base applications by executing them separately. Then, we execute the *compound* application and obtain its value of the PMC. Typically, the core computations for the compound application consist of the core computations of the base applications programmatically placed one after the other. This has to be the case for PAPI PMCs. It must also be ensured that the execution of the *compound* application takes place under platform conditions similar to those for the execution of its constituent *base* applications.

If the PMC of the *compound* application is equal to the sum of the PMCs of the base applications (with a tolerance of 5.0%), we classify the PMC as potentially *additive*. Otherwise, it is *non-additive*.

We call the PMC that passes the *additivity test* as potentially additive. For it to be called *absolutely additive* on a platform, ideally, it must pass the test for all conceivable compound applications on the platform. Therefore, we avoid this definition.

For each PMC, we determine the maximum percentage error. For a *compound* application, the percentage error (averaged over several runs) is calculated as follows:

$$Error(\%) = (|\frac{(\overline{e_{b1}} + \overline{e_{b2}}) - \overline{e_c}}{(\overline{e_{b1}} + \overline{e_{b2}} + \overline{e_c})/2}|) \times 100 \tag{4.6}$$

where $\overline{e_c}, \overline{e_{b1}}, \overline{e_{b2}}$ are the sample means of predictor variables for the compound application and the constituent base applications respectively. The maximum percentage error is then calculated as the maximum of the errors for all the *compound* applications in the experimental test suite.

4.3.3 Experimental Methodology to Obtain Likwid and PAPI PMCs

In this section, we present our experimental setup to determine the *additivity* of PMCs.

The experiments are performed on the Intel Haswell multicore CPU platform (specifications given in Table 3.1). We used diverse range of applications (both compute-bound and memory-bound) in our testsuite composed of NAS parallel benchmarking suite (NPB), Intel math kernel library (MKL), HPCG [136], and *stress* [137] (description given in Table 3.2).

For each run of an application in our test suite, we measure the following:

- Dynamic energy consumption
- Execution time
- PMCs

The dynamic energy consumption during the application execution is measured using a *WattsUp Pro* power meter and obtained programmatically via the HCLWattsUp interface [132]. The power meter is periodically calibrated using an ANSI C12.20 revenue-grade power meter, Yokogawa WT210. We explain the procedure of calibration in Appendix D. The application programming interface for HCLWattsUp is further explained in Appendix A.2. We would like to mention that the output variables (or response variables) in the performance and energy predictive models, i.e., energy consumption and execution time, are *additive*. We confirm this via thorough experimentation and therefore we will not discuss them hereafter.

We present our experimental methodologies for determining Likwid and PAPI PMCs in Appendix C

4.3.4 Steps to Ensure Reliable Experiments

To ensure the reliability of our results, we follow a statistical methodology where a sample mean for a response variable is obtained from multiple experimental runs. The sample mean is calculated by executing the application repeatedly until it lies in the 95% confidence interval and a precision of 0.025 (2.5%) has been achieved. For this purpose, Student's t-test is used assuming that the individual observations are independent and their population follows

the normal distribution. We verify the validity of these assumptions by plotting the distributions of observations. Appendix A.3 presents our experimental methodology to determine the sample mean.

The server is fully dedicated to the experiments. To ensure reliable energy measurements, we took the following precautions:

1. HCLWattsUp API [132] gives the total energy consumption of the server during the execution of an application using system-level physical power measurements from the external power meters. This includes the contribution of components such as NIC, SSDs, fans, etc. To ensure that the value of dynamic energy consumption is purely due to CPUs and DRAM, we verify that all the components other than CPUs and DRAM are idle using the following steps:

    - Monitoring the disk consumption before and during the application run. We ensure that there is no I/O performed by the application using tools such as *sar*, *iotop*, etc.
    - Ensuring that the problem size used in the execution of an application does not exceed the main memory and that swapping (paging) does not occur.
    - Ensuring that the network is not used by the application using monitoring tools such as *sar*, *atop*, etc.
    - Binding an application during its execution to resources using cores-pinning and memory-pinning.

2. Our platform supports three modes to set the speeds of the fans: *minimum*, *optimal*, and *full*. The speeds of all the fans are set to *optimal* during the execution of our experiments. We make sure there is no contribution to the dynamic energy consumption from fans during an application run, by following the steps below:

    - We continuously monitor the temperature of the server and the speed of fans, both when the server is idle, and during the application run. We obtain this information by using the Intelligent Platform Management Interface (IPMI) sensors.

- We observed that both the temperature of the server and the speeds of the fans remained the same whether the given application is running or not.
- We set the fans at *full* speed before starting the application run. The results from this experiment were the same as when the fans were run at *optimal* speed.
- To make sure that pipelining, cache effects, etc, do not happen, the experiments are not executed in a loop and sufficient time (120 seconds) is allowed to elapse between successive runs. This time is based on observations of the times taken for the memory utilization to revert to base utilization and processor (core) frequencies to come back to the base frequencies.

We now present our experimental study on the additivity of PMCs. We divide our exploration into two classes, i.e., Class A and Class B. Class A presents an initial study on the additivity of PMCs using a small set of compound applications. However, Class B presents more detailed insights of the additivity of PMCs with several compound applications.

4.3.5 Class A: A Preliminary Study on the Additivity of PMCs Using Two Popular Tools

Additivity of Likwid PMCs

In this section, we determine the *additivity* of Likwid PMCs. We execute *compound* applications where each application is composed from two base applications in our testsuite (shown in Table 3.2).

The list of potentially *additive* PMCs is shown in the Table 4.1. The list of *non-additive* PMCs is presented in Table 4.2, which also reports the maximum percentage error for each PMC.

It is noteworthy that some *non-additive* PMCs are used as predictor variables in many energy predictive models [43, 138, 139, 140, 44]. These are *ICache events*, *L2 Transactions*, and *L2 Requests*.

Table 4.1: List of potentially additive *Likwid PMCs*

BR_INST_EXEC_ALL_BRANCHES	IDQ_UOPS_NOT_DELIVERED_CYCLES_0 _UOPS_DELIV_CORE
BR_MISP_EXEC_ALL_BRANCHES	IDQ_UOPS_NOT_DELIVERED_CYCLES_FE_WAS_OK
BR_INST_RETIRED_ALL_BRANCHES	UOPS_EXECUTED_PORT_PORT_0
BR_MISP_RETIRED_ALL_BRANCHES	UOPS_EXECUTED_PORT_PORT_1
DRAM_CLOCKTICKS	UOPS_EXECUTED_PORT_PORT_2
SNOOPS_RSP_AFTER_DATA_LOCAL	UOPS_EXECUTED_PORT_PORT_3
SNOOPS_RSP_AFTER_DATA_REMOTE	UOPS_EXECUTED_PORT_PORT_4
RXL_FLITS_G1_DRS_NONDATA	UOPS_EXECUTED_PORT_PORT_5
RXL_FLITS_G0_NON_DATA	UOPS_EXECUTED_PORT_PORT_6
TXL_FLITS_G0_NON_DATA	UOPS_EXECUTED_PORT_PORT_7
CPU_CLK_UNHALTED_ANY	UOPS_EXECUTED_PORT_PORT_0_CORE
CPU_CLOCK_UNHALTED_THREAD_P	UOPS_EXECUTED_PORT_PORT_1_CORE
CPU_CLOCK_UNHALTED_THREAD_P_ANY	UOPS_EXECUTED_PORT_PORT_2_CORE
CPU_CLOCK_UNHALTED_REF_XCLK	UOPS_EXECUTED_PORT_DATA_PORTS
CPU_CLOCK_UNHALTED_REF_XCLK_ANY	L2_RQSTS_ALL_DEMAND_REFERENCES
HA_R2_BL_CREDITS_EMPTY_LO_HA0	L2_RQSTS_L2_PF_MISS
HA_R2_BL_CREDITS_EMPTY_LO_HA1	MEM_UOPS_RETIRED_ALL
CPU_CLOCK_THREAD_UNHALTED _ONE_THREAD_ACTIVE	UOPS_EXECUTED_PORT_PORT_3_CORE
CPU_CLOCK_UNHALTED_TOTAL_CYCLES	UOPS_EXECUTED_PORT_PORT_4_CORE
OFFCORE_REQUESTS_OUTSTANDING _DEMAND_DATA_RD	UOPS_EXECUTED_PORT_PORT_5_CORE
OFFCORE_REQUESTS_OUTSTANDING _CYCLES_WITH_DATA_RD	UOPS_EXECUTED_PORT_PORT_6_CORE
OFFCORE_REQUESTS_OUTSTANDING _DEMAND_DATA_RD_C6	UOPS_EXECUTED_PORT_PORT_7_CORE
UOPS_EXECUTED_PORT_DATA_PORTS	UOPS_EXECUTED_PORT_ARITH_PORTS
OFFCORE_REQUESTS_DEMAND_DATA_RD	UOPS_EXECUTED_PORT_ARITH_PORTS_CORE
HA_R2_BL_CREDITS_EMPTY_HI_R2_NCB	UOPS_EXECUTED_PORT_DATA_PORTS
CPU_CLOCK_UNHALTED_THREAD_P	UOPS_RETIRED_CORE_TOTAL_CYCLES
CPU_CLOCK_UNHALTED_THREAD_P_ANY	LSD_CYCLES_4_UOPS
CPU_CLOCK_UNHALTED_REF_XCLK	UOPS_EXECUTED_THREAD
CPU_CLOCK_UNHALTED_REF_XCLK_ANY	UOPS_EXECUTED_USED_CYCLES
CPU_CLOCK_THREAD_UNHALTED _ONE_THREAD_ACTIVE	UOPS_EXECUTED_STALL_CYCLES
CPU_CLOCK_UNHALTED_TOTAL_CYCLES	UOPS_EXECUTED_TOTAL_CYCLES
ICACHE_MISSES	UOPS_EXECUTED_CYCLES_GE_1_UOPS_EXEC
L2_RQSTS_RFO_MISS	UOPS_EXECUTED_CYCLES_GE_2_UOPS_EXEC
L2_RQSTS_ALL_RFO	UOPS_EXECUTED_CYCLES_GE_3_UOPS_EXEC
L2_RQSTS_CODE_RD_HIT	UOPS_EXECUTED_CYCLES_GE_4_UOPS_EXEC
L2_RQSTS_CODE_RD_MISS	UOPS_EXECUTED_CORE
UOPS_EXECUTED_PORT_DATA_PORTS	UOPS_EXECUTED_CORE_USED_CYCLES
MEM_LOAD_UOPS_RETIRED_ALL_ALL	UOPS_EXECUTED_CORE_STALL_CYCLES
UOPS_ISSUED_ANY	UOPS_EXECUTED_CORE_TOTAL_CYCLES
UOPS_ISSUED_USED_CYCLES	UOPS_EXECUTED_CORE_CYCLES_GE_1_UOPS_EXEC
UOPS_ISSUED_STALL_CYCLES	UOPS_EXECUTED_CORE_CYCLES_GE_2_UOPS_EXEC
UOPS_ISSUED_TOTAL_CYCLES	UOPS_EXECUTED_CORE_CYCLES_GE_3_UOPS_EXEC
UOPS_ISSUED_CORE_USED_CYCLES	UOPS_EXECUTED_CORE_CYCLES_GE_4_UOPS_EXEC
UOPS_ISSUED_CORE_STALL_CYCLES	UOPS_RETIRED_ALL
UOPS_ISSUED_CORE_TOTAL_CYCLES	UOPS_RETIRED_CORE_ALL
IDQ_MITE_ALL_UOPS	UOPS_RETIRED_RETIRE_SLOTS
IDQ_DSB_UOPS	UOPS_RETIRED_CORE_RETIRE_SLOTS
IDQ_MS_UOPS	UOPS_RETIRED_USED_CYCLES
IDQ_ALL_DSB_CYCLES_ANY_UOPS	UOPS_RETIRED_STALL_CYCLES
IDQ_ALL_DSB_CYCLES_4_UOPS	UOPS_RETIRED_TOTAL_CYCLES
IDQ_ALL_MITE_CYCLES_ANY_UOPS	UOPS_RETIRED_CORE_USED_CYCLES
IDQ_UOPS_NOT_DELIVERED_CORE	UOPS_RETIRED_CORE_STALL_CYCLES
CAS_COUNT_RD	CAS_COUNT_WR
CAS_COUNT_ALL	

Table 4.2: List of non-additive *Likwid PMCs*

Event Name	**Additivity Error (%)**
UNCORE_CLOCK	16.98
CBOX_CLOCKTICKS	16.98
SBOX_CLOCKTICKS	17.08
WBOX_CLOCKTICKS	17.57
BBOX_CLOCKTICKS	16.98
PBOX_CLOCKTICKS	16.98
RBOX_CLOCKTICKS	16.98
QBOX_CLOCKTICKS	17.57
HA_R2_BL_CREDITS_EMPTY_LO_R2_NCB	45.27
HA_R2_BL_CREDITS_EMPTY_LO_R2_NCS	48.28
HA_R2_BL_CREDITS_EMPTY_HI_HA0	203.15
HA_R2_BL_CREDITS_EMPTY_HI_HA1	213.15
HA_R2_BL_CREDITS_EMPTY_HI_R2_NCS	250.56
OFFCORE_RESPONSE_0_DMND_DATA_RD_ANY	47.50
ICACHE_IFETCH_STALL	86.60
L2_RQSTS_RFO_HIT	27.44
ARITH_DIVIDER_UOPS	3075.23
IDQ_UOPS_NOT_DELIVERED_CYCLES_LE_1_ UOP_DELIV_CORE	163.64
IDQ_UOPS_NOT_DELIVERED_CYCLES_LE_2_ UOP_DELIV_CORE	89.16
L2_RQSTS_L2_PF_HIT	39.41
ICACHE_HIT	105.45
RXL_FLITS_G0_DATA	176.62
OFFCORE_REQUESTS_OUTSTANDING_ ALL_DATA_RD	33.76
OFFCORE_REQUESTS_ALL_DATA_RD	42.45
IDQ_MITE_UOPS	42.06
L2_RQSTS_ALL_DEMAND_DATA_RD	52.76
L2_TRANS_DEMAND_DATA_RD	24.29
L2_RQSTS_ALL_DEMAND_DATA_RD_MISS	29.14
L2_RQSTS_ALL_DEMAND_DATA_RD_HIT	35.09
L2_RQSTS_ALL_DEMAND_DATA_RD	39.43
L2_TRANS_DEMAND_DATA_RD	52.43
L2_RQSTS_ALL_DEMAND_DATA_RD_MISS	56.23
L2_RQSTS_ALL_DEMAND_DATA_RD_HIT	72.32
L2_RQSTS_ALL_DEMAND_DATA_RD	35.03
L2_TRANS_DEMAND_DATA_RD	75.24
L2_RQSTS_ALL_DEMAND_DATA_RD	80.33
RXL_FLITS_G2_NCB_DATA	100
RXL_FLITS_G2_NCB_NONDATA	100
TXL_FLITS_G0_DATA	100
TXL_FLITS_G1_DRS_DATA	100
TXL_FLITS_G1_DRS_NONDATA	100
TXL_FLITS_G2_NCB_DATA	100
LSD_UOPS	42

Additivity of PAPI PMCs

In this section, we determine the *additivity* of PAPI PMCs. We again execute all the *compound* applications where each application is composed from two base applications in our test suite (shown in Table 3.2).

The list of potentially *additive* PMCs is shown in Table 4.3. The list of *non-additive* PMCs is shown in Table 4.4, which also reports the maximum percentage error for each PMC.

It should be mentioned that some of these *non-additive* PMCs such as $PAPI\_L1\_ICM$ and $PAPI\_L2\_ICM$ have been widely used in energy and performance predictive models [46, 141, 51, 142, 143, 144]. These represent L1 and L2 instruction cache misses.

Table 4.3: List of potentially additive *PAPI PMCs*

PAPI_L1_DCM	PAPI_FUL_CCY	PAPI_L2_DCW
PAPI_L2_DCM	PAPI_BR_UCN	PAPI_L3_DCW
PAPI_CA_SHR	PAPI_BR_CN	PAPI_L3_TCR
PAPI_CA_CLN	PAPI_BR_TKN	PAPI_L2_TCW
PAPI_CA_INV	PAPI_BR_NTK	PAPI_L3_TCW
PAPI_CA_ITV	PAPI_BR_MSP	PAPI_REF_CYC
PAPI_L1_STM	PAPI_BR_PRC	PAPI_L1_TCM
PAPI_L2_LDM	PAPI_TOT_INS	PAPI_L2_TCM
PAPI_L2_STM	PAPI_L2_DCR	PAPI_BR_INS
PAPI_PRF_DM	PAPI_L3_DCR	PAPI_RES_STL
PAPI_TOT_CYC	PAPI_L2_DCA	PAPI_L3_DCA
PAPI_L2_TCA	PAPI_L2_TCR	PAPI_L3_TCA

Core and Memory Pinning

Pinning is the process or binding an application to a specific core or memory bank to improve its performance by increasing the percentage of local memory accesses.

We ran two sets of experiments, one with the application pinned to the cores and the other with the application pinned to cores and memory.

Table 4.4: List of non-additive *PAPI PMCs*

Event Name	Additivity Error (%)
PAPI_CA_SNP	40.23
PAPI_TLB_DM	31.54
PAPI_TLB_IM	23.70
PAPI_STL_CCY	31.43
PAPI_LD_INS	32.06
PAPI_SR_INS	21.98
PAPI_LST_INS	45.87
PAPI_L1_ICM	37.28
PAPI_L2_ICM	37.50
PAPI_L2_ICH	107.12
PAPI_L2_ICA	30.65
PAPI_L3_ICA	30.2
PAPI_L2_ICR	30.65
PAPI_L3_TCM	14.54
PAPI_L3_LDM	74.68
PAPI_L1_LDM	200.82
PAPI_L3_ICR	19.48

While the percentage errors were reduced slightly when the application is pinned to both the cores and the memory, we observed that memory pinning has no effect on *additive* PMCs but, most importantly, *non-additive* PMCs remained *non-additive* (within a tolerance of 5%).

4.3.6 Class B: Extended Study to Rank PMCs Using Additivity Test

We further conduct a detailed study of the additivity of PMCs offered by *Likwid*.

For the experimental results, we prepare a dataset consisting of 100 compound applications composed of the base applications presented in Table 3.2. No PMC is found to be additive within a specified tolerance of 5%. If we increase the tolerance to 20%, 50 PMCs become additive. Increasing the tolerance to 30% makes 109 PMCs additive. We observe that a PMC can be non-additive with an error as high as 4315% and there are many PMCs where

the error is over 100%.

4.3.7 Evolution of *Additivity* of PMCs from Single-core to Multicore Architectures

To identify the cause of this non-additivity, we perform an experimental study to observe the additivity of PMCs with different configurations of threads/cores employed in an application.

We choose for this study three applications: 1). MKL DGEMM, 2). MKL FFT and 3). *naive* matrix-vector (MV) multiplication. We perform additivity test for the applications for four different core configurations (2-core, 8-core, 16-core, and 24-core). In the 2-core configuration, the application is pinned to one core of each socket. In the 8-core configuration, the application is pinned to four cores of each socket and so on. We design multiple compound applications from the chosen set of problem sizes. For each application and core configuration, we note the maximum percentage error for each PMC and count the number of non-additive PMCs that exceed the input tolerance of 5%.

Figure 4.1 shows the increase in non-additivity of PMCs as the number of cores is increased for DGEMM, FFT and *naive* MV. For DGEMM, 51 PMCs are non-additive for 2-core configuration. The number increases to 126 for 24-core configuration. For FFT, the number increases from 61 to 146 and for *naive* MV, the number increases from 22 to 58 from 2-core to 24-core configurations. The minimum number of non-additive PMCs is for the 2-core configuration for each application.

Therefore, we conclude that the number of non-additive PMCs increases with the increase in cores employed in an application execution because of the two inherent characteristics of modern computing platforms 1) severe resource sharing and 2) contention. Severe resource contention is due to tight integration of tens of cores organized in multiple sockets with multi-level cache hierarchy and contending for shared on-chip resources such as last level cache (LLC), interconnect (For example Intel's Quick Path Interconnect, AMD's Hyper Transport), and DRAM controllers. However, the non-uniform memory access (NUMA) where the time for memory access between a core and main memory

is not uniform and where main memory is distributed between locality domains or groups called NUMA nodes. These two characteristics causes a lot of variations in the energy consumption profiles of the applications. The variations have been a subject in the optimization of energy consumption in the recent years and the non-additivity of performance events can explain the increasing amount of variations in the profiles as the number of cores increase.

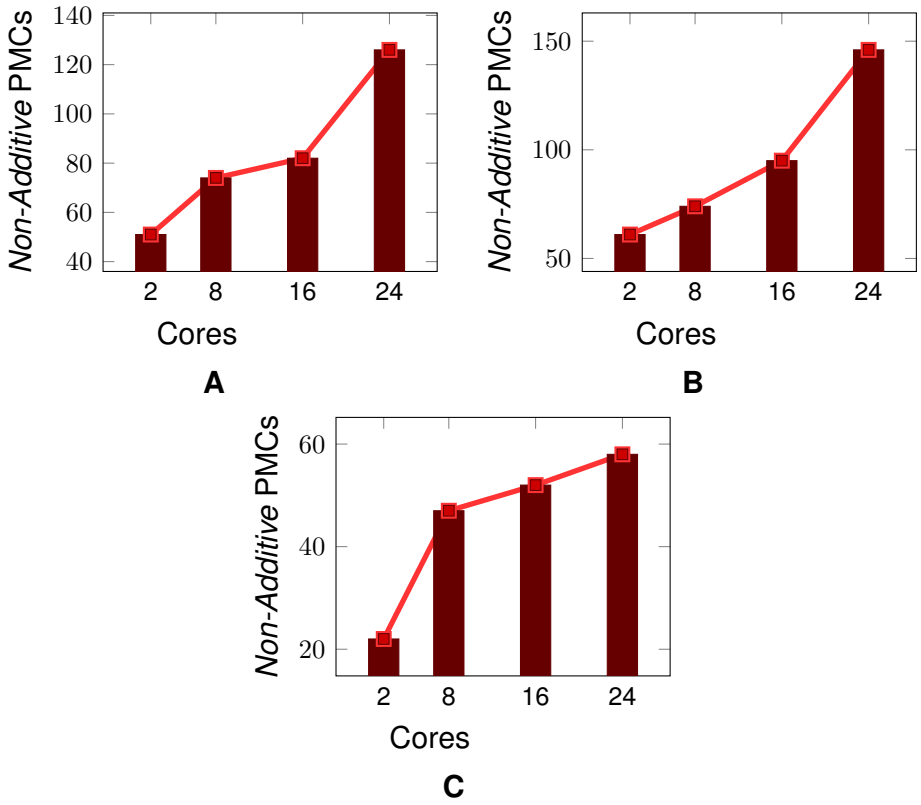


Figure 4.1: Increase in number of non-additive PMCs with threads/cores used in an application. (A), (B), and (C) shows non-additive PMCs for Intel MKL DGEMM, Intel MKL FFT and naive *matrix-vector multiplication.*

4.3.8 Discussion

We first discuss some of the insights from our initial study of the additivity of PMCs. From Table 4.1 and Table 4.2 showing *potentially additive* and *non-additive* Likwid PMCs respectively, one can observe that out of a total of 151

PMCs, 43 PMCs are *non-additive*.

The event *ARITH_DIVIDER_UOPS* exhibits the highest maximum percentage error of about 3075%. This event belongs to the μOPS group of Likwid PMCs responsible for gathering PMCs related to the instruction pipeline.

Several PMCs such as 1) *HA_R2_BL_CREDITS_EMPTY_HI_HA0*, 2) *HA_R2_BL_CREDITS_EMPTY_HI_HA1*, and 3) *HA_R2_BL_CREDITS_EMPTY_HI_R2_NCS* show maximum percentage error of about 200%. These events specifically belong to the *Home Agent (HA)* group of Likwid PMCs. HA is the central unit that is responsible for providing PMCs from the protocol side of memory interactions.

There are several PMCs that show the maximum percentage error of about 100%. They are mainly from the *QPI* group of Likwid PMCs responsible for packetizing requests from the caching agent on the way out to the system interface.

Similarly, from Table 4.3 and Table 4.4 showing *potentially additive* and *non-additive* PAPI PMCs respectively, 17 PMCs out of a total of 53 PMCs are *non-additive*. One PMC, *PAPI_L1_LDM*, demonstrates the highest maximum percentage error of about 200%. It represents L1 load misses. Another PMC, *PAPI_L2_ICH*, demonstrates a maximum percentage error of over 100%. It represents L2 instruction cache hits.

If we increase the tolerance to about 20%, then only 8 *non-additive* Likwid PMCs will become *potentially additive*. For PAPI, only two *non-additive* PMCs will become *potentially additive*, *PAPI_L3_TCM* and *PAPI_L3_ICR*. They represent L3 cache misses and L3 instruction cache reads respectively. Increasing the tolerance to about 30% results in other 3 *non-additive* Likwid PMCs and 5 *non-additive* PAPI PMCs becoming potentially *additive*.

Thus, one can see that there are still a large number of PMCs that are *non-additive* even after increasing the tolerance to as high as 30%. Some of these PMCs have been used as key predictor variables in energy predictive models [43, 138, 139, 140, 44, 46, 141, 51, 142, 143, 144].

To summarize, our initial study in Class A suggest that the *non-additive* PMCs that exceed a specified tolerance must be excluded from the list of PMCs to be considered as predictor variables for energy predictive modeling,

because they can potentially damage the prediction accuracy of these models due to their highly non-deterministic nature. Also, the list of potentially *additive* PMCs must be further tested exhaustively for more diverse applications and platforms to secure more confidence in their *additivity*.

A detailed study of additivity of PMCs shows that no PMCs are additive for all the applications. Therefore, we conclude that all the PMCs fail the additivity test with a specified tolerance of 5% on current multicore platforms.

We also discovered that the number of non-additive PMCs rises with an increase in the number of cores employed in the application. We consider this to be an inherent trait of modern multicore computing platforms because of severe resource contention and non-uniform memory access (NUMA).

4.4 Group 2: Improving Prediction Accuracy of Platform-Level Energy Predictive Models Using *Consistency Test*

In this section, we study the impact of the additivity of PMCs on the prediction accuracy of state-of-the-art energy predictive models constructed using the popular mainstream technique, i.e., linear regression (LR).

A *LR* model can be represented as $Y_i = \beta_0 + \sum_{j=1}^{M} \beta_j X_{ij} + \epsilon_i$, where $i = 1, 2, ..., N$ represent the number of observations, and $j = 1, 2, ..., M$ represent the number of independent variables. In our case, Y_i are dynamic energy measurements obtained using HCLWattsUp, and X_{ij} are the observed values of the PMCs. ϵ is the error term or noise in measurement. We build a linear model using regression technique by estimating the value of model intercept (β_0) and the model coefficients (β).

We now present the experiments in this group to study the energy predictive models using linear regression.

4.4.1 Experiments and Analysis

We select six PMCs common to the state-of-the-art models [138, 43, 140, 87, 76, 145]. The PMCs ($\{X_1, \cdots, X_6\}$) are listed in the Table 4.5. They belong to the following dominant PMC groups: cache, branch instructions, micro-operations (uops), floating-point instructions, and main memory accesses. They fail the additivity test for an input tolerance of 5%. The PMC X_6 is highly additive compared to the rest.

Table 4.5: Correlation of PMCs with dynamic energy consumption (E_D). (A) List of selected PMCs for modelling with their additivity test errors (%). (B) Correlation matrix showing positive correlations of dynamic energy with PMCs. 100% correlation is denoted by 1. X_4, X_5, and X_6 are highly correlated with E_D.

Selected PMCs	***Additivity* Test Error(%)**
X_1: *IDQ_MITE_UOPS*	13
X_2: *IDQ_MS_UOPS*	37
X_3: *ICACHE_64B_IFTAG_MISS*	36
X_4: *ARITH_DIVIDER_COUNT*	80
X_5: *L2_RQSTS_MISS*	14
X_6: *UOPS_EXECUTED_PORT_PORT_6*	10

A

	E_D	X_1	X_2	X_3	X_4	X_5	X_6
E_D	1	0.53	0.50	0.42	0.58	0.99	0.99
X_1	0.53	1	0.41	0.25	0.39	0.45	0.44
X_2	0.50	0.41	1	0.19	0.99	0.48	0.48
X_3	0.42	0.25	0.19	1	0.21	0.41	0.40
X_4	0.58	0.39	0.99	0.21	1	0.57	0.56
X_5	0.99	0.45	0.48	0.41	0.57	1	0.99
X_6	0.99	0.44	0.48	0.40	0.56	0.99	1

B

Table 4.6: Linear predictive models (MA_1-MG_1) with intercepts and their minimum, average, and maximum prediction errors. Coefficients can be positive or negative.

Model	**PMCs**	**Intercept followed by Coefficients**	**Percentage prediction errors (min, avg, max)**
MA_1	$X_1, X_2, X_3, X_4, X_5, X_6$	10.2, 3.06E-09, 1.95E-08, 3.30E-07, -1.02E-06, 6.18E-08, -9.39E-11	(2.7, 32, 99.9)
MB_1	X_1, X_2, X_3, X_5, X_6	12.8, 3.68E-09, 2.26E-10, 3.43E-07, 7.40E-08, -4.763E-10	(2.5, 23.32, 80.42)
MC_1	X_1, X_3, X_5, X_6	16.4, 3.71E-09, 3.34E-07, 7.45E-08, -4.87E-10	(2.5, 21.86, 76.9)
MD_1	X_1, X_5, X_6	29.9, 3.72E-09, 7.54E-08, -5.076E-10	(2.5, 21.78, 77.33)
ME_1	X_1, X_6	130, 4.21E-09, 1.456E-09	(2.5, 18.01, 89.23)
MF_1	X_6	749, 1.53E-09	(2.5, 14.39, 34.64)
MG_1	X_4, X_5, X_6	492, 6.79E-08, 9.45E-08, -9.60E-10	(2.5, 23.46, 80)

We build three types of linear regression models as follows:

- **Type 1**: Models MA_1-MG_1 with no restrictions on intercepts and coefficients.
- **Type 2**: Models MA_2-MG_2 whose intercepts are forced to zero.

Table 4.7: Linear predictive models (MA_2-MG_2) with zero intercepts and their minimum, average, and maximum prediction errors. Coefficients can be positive or negative.

Model	PMCs	Coefficients	Percentage prediction errors (min, avg, max)
MA_2	$X_1, X_2, X_3, X_4, X_5, X_6$	1.08E-09, 1.96E-08, 3.51E-07, -1.02E-06, 6.19E-08, -9.78E-11	(2.5, 32, 78.7)
MB_2	X_1, X_2, X_3, X_5, X_6	3.71E-09, 2.37E-10, 3.69E-07, 7.42E-08, -4.82E-10	(2.5, 23.32, 80.57)
MC_2	X_1, X_3, X_5, X_6	3.75E-09, 3.66E-07, 7.48E-08, -4.95E-10	(2.5, 22.1, 77.5)
MD_2	X_1, X_5, X_6	3.80E-09, 7.61E-08, -5.27E-10	(2.5, 22.4, 78.5)
ME_2	X_1, X_6	4.60E-09, 1.46E-09	(2.5, 18.01, 89.45)
MF_2	X_6	1.60E-09	(3.0, 68.53, 90.53)
MG_2	X_4, X_5, X_6	1.34E-07, 1.22E-07, -1.65E-09	(2.5, 47.5, 111.22)

Table 4.8: Linear predictive models (MA_3-MG_3) with zero intercepts. Coefficients cannot be negative. The minimum, average, and maximum prediction errors of IntelRAPL and the linear predictive models.

Model	PMCs	Coefficients	Percentage prediction errors (min, avg, max)
MA_3	$X_1, X_2, X_3, X_4, X_5, X_6$	3.83E-09, 3.67E-10, 5.30E-07, 0, 5.56E-08, 0	(6.6, 31.2, 61.9)
MB_3	X_1, X_2, X_3, X_5, X_6	3.83E-09, 3.67E-10, 5.30E-07, 0, 5.56E-08	(6.6, 31.2, 61.9)
MC_3	X_1, X_3, X_5, X_6	3.75E-09, 5.34E-07, 5.58E-08, 0	(2.5, 25.3, 62.1)
MD_3	X_1, X_5, X_6	4.00E-09, 5.59E-08, 0	(2.5, 23.86, 100.3)
ME_3	X_1, X_6	4.60E-09, 1.46E-09	(2.5, 18.01, 89.45)
MF_3	X_6	1.60E-09	(2.5, 68.5, 90.5)
MG_3	X_4, X_5, X_6	1.72E-07, 5.86E-08, 0	(2.5, 50, 77.9)
IntelRAPL			(4.1, 30.6, 58.9)

- **Type 3**: Models MA_3-MG_3 whose intercepts are forced to zero and whose coefficients cannot be negative.

Within each type t, MA_t employs all the PMCs as predictor variables. MB_t is based on five PMCs with the least additive PMC (X_4) removed. MC_t uses four PMCs with two most non-additive PMCs (X_2, X_4) removed and so on until MF_t containing only the most additive PMC (X_6). MG_t uses three PMCs (X_4, X_5, X_6) with the highest correlation with dynamic energy consumption.

For constructing all the models, we use a dataset of 277 points where each point contains dynamic energy consumption and the PMC counts for the execution of one base application from Table 3.2 with some particular input. For testing the prediction accuracy of the models, we construct a test dataset of 50 different compound applications. We used this division (227 for training, 50 for testing) based on best practices and experts' opinions in this domain.

Table 4.6 summarizes the type 1 models. Following are the salient observations:

- The model intercepts are significant. In our theory of *energy of computing* where we consider modelling of dynamic energy consumption, the intercepts are not present since they have no real physical meaning. Consider the case where no application is executed. The values of the PMCs will be zero and therefore the models must output the dynamic energy consumption to be zero. The models, however, output the values of their intercepts as the dynamic energy consumption. This violates the energy conservation property in the theory.

- MA_1 has negative coefficients for PMCs, X_4 and X_6. Models MB_1-MD_1 have negative coefficients for PMC, X_6. The negative coefficients in these models can give rise to negative predictions for applications where the counts for X_4 and X_6 are higher than the other PMCs. We illustrate this case by designing a microbenchmark that stresses specifically hardware components resulting in large counts for the PMCs with the negative coefficients. Since, in our case, X_4 and X_6 count the division and floating-point instructions, our microbenchmark is a simple *assembly* language program that performs floating-point division operations in a loop. When run for forty seconds, the PMC counts for this application on our platform were: X_1=7022011, X_2=623142, X_3=121489, X_4=5101219180, X_5=33210, and X_6=186971207082. The energy consumption predictions for this application from our four models $\{A_1, B_1, C_1, D_1\}$ are $\{-5210.52, -76.23, -74.59, -64.98\}$ which violate the energy conservation law.

- Since the predictor variables have a high positive correlation with energy consumption, their coefficients should exhibit the same relationship. The coefficients, however, have different signs for different models. Consider, for example, X_4 in MA_1 and MC_1. While it has a positive coefficient of A_1, it has a negative coefficient of MC_1. Similarly, X_6 in A_1 and B_1 has negative coefficient, whereas in MF_1 it has a positive coefficient. We have found that the research works that propose linear models using these PMCs do not contain any sanity check for these coefficients. Therefore, we believe that using them in models without understanding

the true meaning or the nature of their relationship with dynamic energy consumption can lead to serious inaccuracy.

The type 2 models are built using specialized linear regression, which forces the intercept to be zero. Table 4.7 contains their summary. All the models excepting ME_2 and MF_2 contain negative coefficients and therefore present the same issues that violate the energy conservation law.

The type 3 models are built using penalized linear regression using the *R programming* interface that forces the coefficients to be non-negative. All the models of this type have zero intercepts and are summarized in Table 4.8. They incorporate basic sanity checks that disallow violations of energy conservation property.

We will now focus on the minimum, average, and maximum prediction errors of type 3 models. They are (6.6%, 31.2%, 61.9%) for MA_3. Since the coefficients are constrained to be non-negative, X_6 ends up having a zero coefficient. We remove the PMC with the next highest non-additivity (X_4) and construct MB_3 based on the remaining five PMCs. In this model, X_5 has a zero coefficient. Its prediction errors are (6.6%, 31.2%, 61.9%). We then remove the PMC with the next highest non-additivity (X_2) from the list of four and build MC_3 based on the remaining PMCs. Its prediction errors are (2.5%, 25.3%, 62.1%). Finally, we build MF_3 with just one most additive PMC (X_6). Its prediction errors are (2.5%, 68.5%, 90.5%). The prediction errors of RAPL are (4.1%, 30.6%, 58.9%). The prediction errors of MG_3 are (2.5%, 50%, 77.9%).

We derive the following conclusions:

- As we remove non-additive PMCs one by one, the average prediction accuracy of the models improves significantly. ME_3 with two most additive PMCs is the best in terms of average prediction accuracy. We, therefore, conclude that employing non-additive PMCs can significantly impair the prediction accuracy of models and that inclusion of highly additive PMCs improves the prediction accuracy of models drastically.

- We highlight two examples demonstrating the dangers of pure fitting exercise (for example: applying linear regression) without understanding

the true physical significance of a predictor variable.

  - The PMC X_6, which has the highest significance in terms of contribution to dynamic energy consumption (highest additivity), ends up having a zero coefficient in MA_3, MC_3, MD_3, and MG_3. MD_3 has only two PMCs, X_1 and X_5, effectively. The linear fitting method picks X_5 instead of X_6 thereby impairing the prediction accuracy of MD_3 (and also MG_3). This is because X_5 and X_6 have a high positive correlation between themselves but the fitting method does not know that X_6 is highly additive.
  - MF_3 containing one PMC with the highest additivity, X_6, has the lowest prediction accuracy. The linear fitting method is unable to find a good fit.

- The average prediction accuracy of *RAPL* is equal to that of the MA_3 and MB_3, which contains the highest number of non-additive PMCs. If the model of *RAPL* is disclosed, one can check how much its prediction accuracy can be improved by removing non-additive PMCs and including highly additive PMCs.

- MG_3 fares worse than *RAPL* and MA_3 even though it contains PMCs that are highly correlated with dynamic energy consumption. ME_3 with two most additive PMCs has better average prediction accuracy than MG_3, which demonstrates that additivity is a more important criterion than correlation.

Figure 4.2 presents the percentage deviations in dynamic energy consumption predictions by type 3 models (Table 4.8) from the system-level physical power measurements obtained using HCLWattsUp (using WattsUp Pro power meters) for different compound applications. $RAPL$, MA_3, and MG_3 exhibit higher average percentage deviations than the best model, ME_3. While $RAPL$ distribution is normal, MA_3 and MG_3 demonstrate non-normality suggesting systemic (not fully random) deviations from the average.

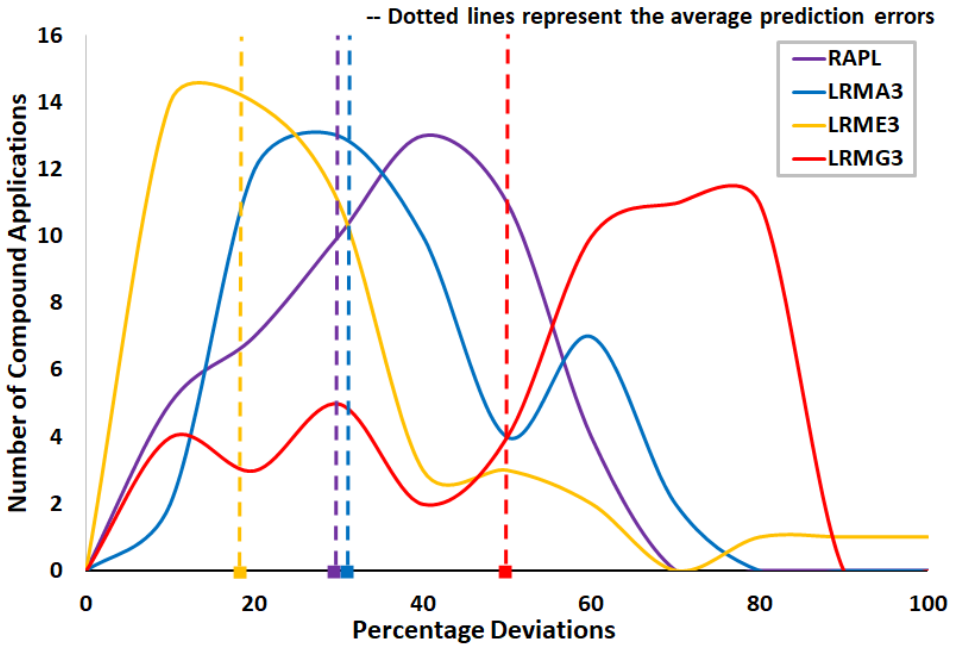


Figure 4.2: Percentage deviations of the type 3 models shown in Table 4.8 from the system-level physical power measurements provided by power meters (HCLWattsUp). The dotted lines represent the averages.

4.4.2 Discussion

This section presents our salient observations from the analysis of models as below:

- We observe that, in general, the prediction accuracy of the models improves as we remove one by one the non-additive PMCs employed in them.
- The linear regression model ME_3 with two highly additive PMCs performs best in terms of average prediction accuracy with an error of 18%.
- The highest average prediction errors are exhibited by models employing one PMC as a predictor variable. This suggests that it is less likely for a single PMC to capture all the possible sources of energy consumption on a modern multicore CPU platform.

4.5 Group 3: Impact of Consistency Test on the Accuracy of Application-Specific Energy Predictive Models

In this section, we study the accuracy limits of application-specific energy predictive models constructed using linear regression. The application-specific models are commonly used computing setups with homogeneous servers executing a limited set of applications to predict the dynamic energy consumption.

We choose the single-socket Intel Skylake server (Table 3.1) for the experiments. A total of 323 PMCs are exposed using Likwid tool on this platform. In order to collect all the PMCs, an application has to be run 99 times.

There is no PMC which is additive within a tolerance of 5% if we consider the large dataset of compound applications composed for the test suite of applications in Table 3.2. However, we discover that many PMCs are additive (within 5%) for a small subset of applications.

We select two such applications, MKL FFT and MKL DGEMM. We now describe below the process of selection of PMCs for the applications:

- From the literature (section 2.3.5) and based on the nature of the applications, we find that the PMCs that mainly reflect the hardware and software activities and that contribute towards the dynamic energy consumption are from the following dominant PMC groups: cache, branch instructions, micro-operations (μops), floating-point instructions, and main memory accesses. We call the PMCs in these groups as *prime PMCs*. For our platform (Table 3.1), prime PMCs are 122.

- We make sure that all the prime PMCs are deterministic and reproducible by executing the DGEMM and FFT applications with the same problem sizes for a number of times. We find that the PMC counts for successive runs of applications are within an accuracy of 99.99%.

- We study the additivity of the prime PMCs. We build a dataset of 50 *base* applications using different problem sizes for DGEMM and FFT. The range of problem sizes for DGEMM is 6500×6500 to 20000×20000, and for FFT is 22400×22400 to 29000×29000. We select this range because of reasonable execution time ($>$ 3 seconds) of the applications on our platform. For each application in a dataset, we measure the following: PMCs, dynamic energy consumption, and execution time. We then build a dataset of 30 *compound* applications from these *base* applications. The additivity test based on the two datasets reveals that there are a number of PMCs that are additive for both the applications.

- We select nine highly additive PMCs (with additivity test errors of less than 1%) for both the applications. These are labeled as $A1$, $A2$, $A3$, ..., $A9$.

- We then select nine non-additive PMCs but which have been employed as predictor variables in notable energy predictive models in the literature (Section 2.3.5). We label them, $NA1$, $NA2$, $NA3$, ..., $NA9$. The set of additive PMCs is denoted by *PA* and non-additive PMCs by *PNA*.

- Finally, we determine the correlations of the PMCs with dynamic energy consumption, which are given in Table F.1.

Table 4.9: Selected additive and non-additive PMCs and their correlation with dynamic energy consumption. 0 to 1 represents positive correlation of 0% to 100%.

	***Additive* PMCs**	**Correlation**
A1	UOPS_RETIRED_CYCLES_GE_4_UOPS_EXEC	0.992
A2	FP_ARITH_INST_RETIRED_DOUBLE	0.993
A3	MEM_INST_RETIRED_ALL_STORES	0.870
A4	UOPS_EXECUTED_CORE	0.993
A5	UOPS_DISPATCHED_PORT_PORT_4	0.870
A6	IDQ_DSB_CYCLES_6_UOPS	0.981
A7	IDQ_ALL_DSB_CYCLES_5_UOPS	0.972
A8	IDQ_ALL_CYCLES_6_UOPS	0.993
A9	MEM_LOAD_RETIRED_L3_MISS	-0.112
	***Non-additive* PMCs**	
NA1	ICACHE_64B_IFTAG_MISS	0.960
NA2	CPU_CLOCK_THREAD_UNHALTED	0.600
NA3	BR_MISP_RETIRED_ALL_BRANCHES	0.992
NA4	MEM_LOAD_L3_HIT_RETIRED_XSNP_MISS	-0.020
NA5	FRONTEND_RETIRED_L2_MISS	0.806
NA6	ITLB_MISSES_STLB_HIT	0.111
NA7	L2_TRANS_CODE_RD	0.860
NA8	IDQ_MS_UOPS	0.99
NA9	ARITH_DIVIDER_COUNT	0.986

We build a dataset containing 401 workload sizes for DGEMM ranging from 6400×6400 to 38400×38400. We also build a dataset for FFT containing 300 workload sizes ranging from 22400×22400 to 41536×41536. Both ranges use a constant step size of 64. We collect the dynamic energy consumption using HCLWattsUP API and the selected PMCs (Table F.1) using Likwid tool for each application. The dataset is further divided for two subsets, one for training and the other for testing of models. 300 and 101 points are used for the training and testing of DGEMM models, respectively. Furthermore, the training and testing subsets for FFT contain 225 and 75 points, respectively.

We build four models, {DGEMM-A, DGEMM-NA, FFT-A, FFT-NA}. The models {DGEMM-A, FFT-A} are trained using PMCs belonging to *PA* and the models {DGEMM-NA, FFT-NA} are trained using PMCs belonging to *PNA*.

We build an extended dataset containing 701 points that combine the datasets for DGEMM and FFT. We collect the dynamic energy consumption using HCLWattsUP API and the selected PMCs (Table F.1) using Likwid tool for each application. The combined dataset is also divided into training and testing subsets. We use 551 points to train and 150 points to test the models. Using the sets of additive (PA) and non-additive (PNA) PMCs, we build two linear models, {M-A, M-NA}.

Table 4.10a show the percentage prediction errors obtained from models. One can see that the models based on PA have better average prediction accuracy than the models based on PNA.

DGEMM-A, FFT-A, and M-A yield $1.3\times$, $2.5\times$, and $2.4\times$ improvement in average prediction accuracy in comparison with DGEMM-NA, FFT-NA, NA, respectively.

Models constructed for single applications, {DGEMM-A, FFT-A}, exhibit better average prediction accuracy than models for the combined dataset of the two applications. This suggests that a specific set of carefully selected PMCs may yield a more accurate application-specific model.

Table 4.10: Prediction accuracies of application-specific models. (a) Energy predictive models using nine PMCs. (b) Energy predictive models using four high positively-correlated PMCs.

Model	**PMCs**	**Prediction Errors (%) [Min, Avg, Max]**
DGEMM-A	*PA*	(0.094, 22.62, 125.48)
DGEMM-NA	*PNA*	(0.218, 31.23, 173.9)
FFT-A	*PA*	(0.447, 36.31, 182.2)
FFT-NA	*PNA*	(3.510, 92.68, 397.2)
M-A	*PA*	(0.005, 35.32, 225.5)
M-NA	*PNA*	(0.449, 85.61, 4039)

(a)

Model	**PMCs**	**Prediction Errors (%) [Min, Avg, Max]**
DGEMM-A4	$PA4$	(0.004, 16.12, 87.25)
DGEMM-NA4	$PNA4$	(0.091, 33.61, 212.3)
FFT-A4	$PA4$	(0.042, 25.12, 190.15)
FFT-NA4	$PNA4$	(5.12, 98.01, 450.2)
M-A4	$PA4$	(0.024, 25.12, 87.25)
M-NA4	$PNA4$	(0.449, 85.61, 4039)

(b)

4.5.1 Impact of Additivity of PMCs and Correlation with Energy on the Accuracy of Energy Predictive Models

Fast construction of accurate online energy consumption predictive models is crucial for real-time systems that require quick reading of the application's energy consumption. Since only 4 PMCs can be collected in a single application run on our platform, the selection of such a reliable subset is crucial to the prediction accuracy of online energy models.

We use *PA* and *PNA* (Table F.1) to build two sets of four most energy correlated PMCs. The first set *PA4*, $\{A1, A2, A4, A8\}$, is constructed using *PA* and the second set *PNA4*, $\{NA1, NA3, NA8, NA9\}$, using *PNA*.

We build six linear models, {DGEMM-A4, DGEMM-NA4, FFT-A4, FFT-NA4, M-A4, M-NA4}. The models {DGEMM-A4, FFT-A4, M-A4} are trained using PMCs belonging to PA4 and the models {DGEMM-NA4, FFT-NA4, M-NA4} are trained using PMCs belonging to PNA4.

Table 4.10b shows the prediction error percentages of the models. We observe that models based on PNA4 built using highly correlated but non-additive PMCs do not demonstrate any improvement in average prediction accuracy compared to models PNA based on nine non-additive PMCs. However, the average prediction accuracy of the models based on PA4 having four highly energy correlated and additive PMCs is significantly better than models based on PA with nine additive PMCs. Model DGEMM-A4 has the least average prediction error of 16%.

4.5.2 Study to Explore Accuracy Limits for PMC-based Application-Specific Models

In this section, we present a study to explore how accurate a PMC-based application-specific model can be for our platform.

We find that all the application-specific models studied so far and with less than 4 PMCs as predictor variables demonstrate poor prediction accuracy compared to models using PA4. Therefore, we build four sets (set A, set B, set C, set D) of models each based on more than 4 PMCs and contain-

ing an increasing number of highly positively correlated PMCs with dynamic energy consumption from the set of most additive PMCs, PA (Table F.1). Set A contains models for DGEMM and FFT using the top five highly correlated PMCs, set B contains models with 6 most highly correlated PMCs, and so on for set D with models based on 8 highly correlated PMCs. The models in each set are given below:

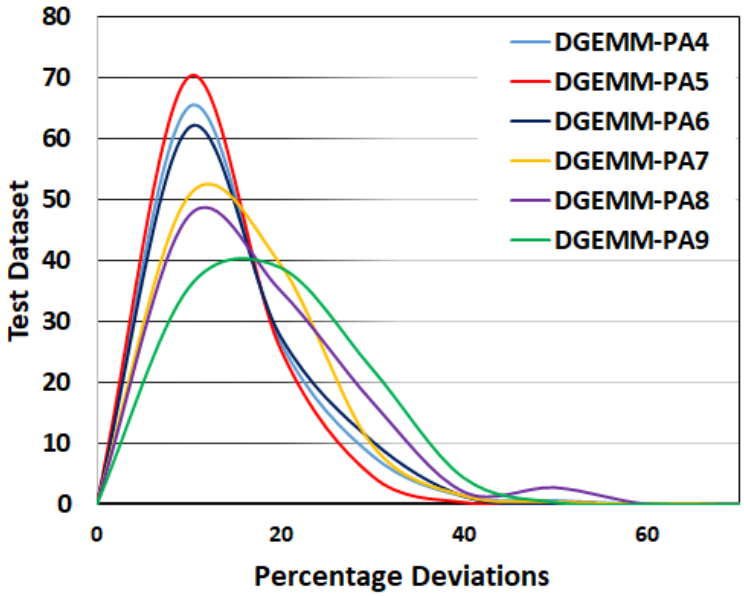
- set A: {DGEMM-PA5, FFT-PA5} employ A1, A2, A4, A6, and A8 as predictor variables for DGEMM and FFT, respectively.
- set B: {DGEMM-PA6, FFT-PA6} employ A1, A2, A4, A6, A7, and A8 as predictor variables for DGEMM and FFT, respectively.
- set C: {DGEMM-PA7, FFT-PA7} employ A1, A2, A3, A4, A6, A7, and A8 as predictor variables for DGEMM and FFT, respectively.
- set D: {DGEMM-PA8, FFT-PA8} employ A1, A2, A3, A4, A5, A6, A7, and A8 as predictor variables for DGEMM and FFT, respectively.

Table 4.11 and 4.12 shows the minimum, average, and maximum percentage errors for the models for DGEMM and FFT, respectively. Figure 4.3(a) and 4.3(b) show the prediction error distribution for the models for DGEMM and FFT applications for our test data-set (containing 101 and 75 data points), respectively.

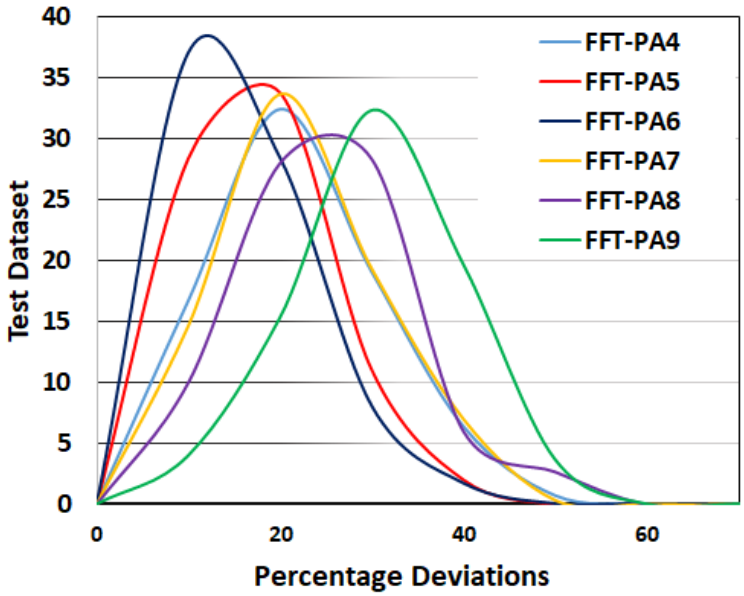
The results show that the prediction errors are the least for the model with five PMCs for DGEMM (13.41%) and the model with six PMCs for FFT (19.21%).

Table 4.11: Accuracy of application-specific energy predictive models for DGEMM employing 5, 6, 7, 8 most positively energy correlated and highly additive PMCs

Model	Prediction Errors (%) [Min, Avg, Max]
DGEMM-PA5	(0.94, 13.41, 119.43)
DGEMM-PA6	(0.32, 16.65, 123.16)
DGEMM-PA7	(0.17, 19.17, 142.32)
DGEMM-PA8	(0.03, 21.18, 126.83)



(a)



(b)

*Figure 4.3: Percentage deviations of the application-specific models shown in Table 4.10, 4.11 and 4.12 from the system-level physical power measurements provided by power meters (HCLWattsUp) for **(a).** DGEMM and **(b).** FFT.*

Table 4.12: Accuracy of application-specific energy predictive models for FFT employing 5, 6, 7, and 8 PMCs that are most positively correlated with energy and highly additive.

Model	Prediction Errors (%) [Min, Avg, Max]
FFT-PA5	(0.42, 22.41, 82.36)
FFT-PA6	(0.62, 19.21, 85.49)
FFT-PA7	(0.23, 27.31, 136.16)
FFT-PA8	(0.48, 29.62, 130.72)

4.5.3 Discussion

Following are the salient observations from the results:

- High positive correlation of the model variables with dynamic energy consumption alone is not sufficient to provide good average prediction accuracy but the model variables must also satisfy the properties of the *consistency* test that takes into account the physical significance of the model variables originating from the conservation of energy of computing, which is the manifestation of the fundamental physical law of energy conservation.

- For our experiments, the training data set is constructed by executing the DGEMM and FFT with a constant increment of workload sizes using a fixed step size of 64. Although, the generation of training data is a tedious task. However, once a model with the desired accuracy is constructed, it represents the relationship of a specific combination of PMCs and understands the underlying pattern with dynamic energy consumption. Therefore, the model can be used to predict the energy consumption for any workload size for an application or a set of applications.

- In general, the average prediction accuracy for single application models is better than that for models using the combined dataset for two applications. This suggests that a specific set of carefully selected PMCs may yield a more accurate application-specific model. However, since the number of additive PMCs for every application differs, the set of PMCs

used as predictor variables for accurate energy predictive modelling may change for each application.

- According to our experimental results, the best PMC based linear energy predictive models for DGEMM and FFT applications have a prediction error of 13.41% and 19.21%, respectively. Our analysis of PMCs on modern computing platforms shows that all they are supposed to be additive by intuition and definition. Our theory of energy predictive models for computing (Section 4.1) proves that a model employing only additive predictor variables must be linear. Therefore, as we employ in the studied linear models only additive and highly correlated PMCs as predictor variables, using any other set of PMCs or non-linear models would not further improve prediction accuracy. To obtain a model with an even better prediction accuracy, one has to explore the use of non-additive model variables, other than PMCs, such as high-level utilization metrics that represent all the energy-consuming activities of the applications executing on a platform, and employ them in non-linear models.

- Since the most accurate linear models for DGEMM and FFT applications employ five and six PMCs as predictor variables, at least six hardware registers must be dedicated to storing the PMCs so that the models can be employed online. Currently, 3-4 hardware registers are dedicated to storing PMCs during an application run on our experimental platforms.

4.6 Group 4: Study of Dynamic Energy Optimization using *Intel RAPL* and System-level Physical Measurements

In this section, we demonstrate that using inaccurate energy measuring tools in energy optimization methods may lead to significant energy losses.

We explain the experimental observations that lead us to investigate if inaccurate energy measurements using Intel RAPL can affect application-level

energy optimizations in Appendix H

We study optimization of a parallel matrix-matrix multiplication application for dynamic energy using the most accurate PMC based linear energy predictive model, *LR MM* (constructed based on the theory of energy predictive models for computing) and two measurement tools, *Intel RAPL* [30] which is a popular mainstream tool and system-level physical power measurements using power meters (HCLWattsUp [132]) which is the ground truth.

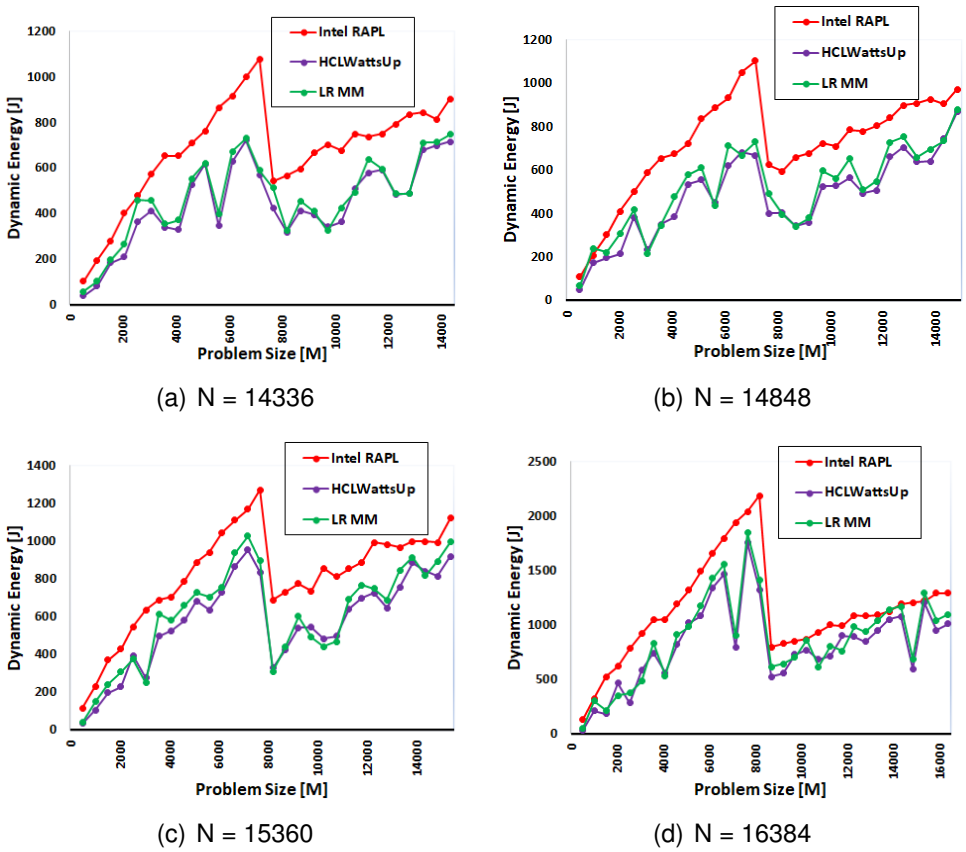
For this purpose, we employ a data-parallel application that uses *Intel MKL DGEMM* as a building block. The experimental platform consists of two servers, HCLServer1 (Table 3.1) and HCLServer2 (Table 3.1). To find the partitioning of matrices between the servers that minimizes the dynamic energy consumption, we use a model-based data partitioning algorithm, which takes as input dynamic energy functional models of the servers. We compare the total dynamic energy consumptions of the solutions returned when the input dynamic energy models of the servers are built using LR MM, IntelRAPL [30], and system-level physical power measurements using power meters (HCLWattsUp [132]. We follow the same strict experimental methodology as in the previous experimental setup to make sure that our experimental results are reliable.

The parallel application computes a matrix product of two dense square matrices A and B of sizes $N \times N$ and is executed using two processors, HCLServer1 and HCLServer2. The matrix A is partitioned between the processors as A_1 and A_2 of sizes $M \times N$ and $K \times N$ where $M + K = N$. Matrix B is replicated at both the processors. Processor HCLServer1 computes the product of matrices A_1 and B and processor HCLServer2 computes the product of matrices A_2 and B. There are no communications involved.

The decomposition of the matrix A is computed using a model-based data partitioning algorithm. The inputs to the algorithm are the number of rows of the matrix A, N, and the dynamic energy consumption functions of the processors, $\{E_1, E_2\}$. The output is the partitioning of the rows, (M, K). The discrete dynamic energy consumption function of processor P_i is given by $E_i = \{e_i(x_1, y_1), ..., e_i(x_m, y_m)\}$ where $e_i(x, y)$ represents the dynamic energy consumption during the matrix multiplication of two matrices of sizes $x \times y$ and $y \times y$ by the processor i. Figures 4.4a–d show the discrete dynamic en-

ergy consumption functions of IntelRAPL, LR MM, and HCLWattsUp for the processors, HCLServer1 and HCLServer2. The dynamic energy profiles are for four problem sizes 14336, 14848, 15360, and 16384. For HCLServer1, the dimension x ranges from 512 to $y/2$ in increments of 512. For HCLServer2, the dimension x ranges from $y - 512$ to $y/2$ in decrements of 512.

$$(M, K) = \underset{\substack{M \in (512, N/2),\\ K \in (N-512, N/2),\\ M+K=N}}{\arg\min} (e_1(M, N) + e_2(K, N))$$



(a) N = 14336 (b) N = 14848

(c) N = 15360 (d) N = 16384

Figure 4.4: Dynamic energy consumption of Intel MKL DGEMM application multiplying two matrices of sizes: $M \times N$ and $N \times N$ on HCLServer1, and $K \times N$ and $N \times N$ on HCLServer2. $M + K = N$.

The main steps of the data partitioning algorithm are as follows:

1. **Plane intersection of dynamic energy functions:** Dynamic energy

consumption functions $\{E_1, E_2\}$ are cut by the plane $y = N$ producing two curves that represent the dynamic energy consumption functions against x given y is equal to N.

2. Determine M and K:

We use four workload sizes $\{14336, 14848, 15360, 16384\}$ in our test data. For each workload size, we determine the workload distribution using the data partitioning algorithm employing a model based on LR MM and IntelRAPL. We execute the parallel application using this workload distribution and determine its dynamic energy consumption. We represent it as e_{lrmm} and e_{rapl}. We obtain the workload distribution using the data partitioning algorithm employing a model based on HCLWattsUp. We execute the parallel application using this workload distribution and determine its dynamic energy consumption. We represent it as $e_{hclwattsup}$. We calculate the percentage loss of dynamic energy consumption provided by HCLWattsUp compared to LR MM and IntelRAPL. Losses for the four workload sizes for LR MM and IntelRAPL are $\{1, 3, 10, 16\}$ and $\{54, 37, 31, 84\}$, respectively.

RAPL is shown to exhibit good prediction accuracy for applications employing decision variables such as dynamic voltage and frequency scaling (DVFS) [34] and the number of application-level threads [35] but keeping the workload size fixed. However, Fahad et al. [36] demonstrate that RAPL shows poor correlation with real measurements if the workload size is varied and all the other parameters are fixed. We validate this finding here but most importantly show that employing RAPL in energy optimization methods where the decision variable is workload distribution, leads to significant energy losses.

4.7 Summary

In this chapter, we proposed a novel selection criterion for PMCs called *additivity*, which can be used to determine the subset of PMCs that can potentially be considered for reliable energy predictive modeling. It is based on the experimental observation that the energy consumption of a serial execution of two applications is the sum of energy consumptions observed for the individual

execution of each application. A linear predictive energy model is consistent if and only if its predictor variables are additive in the sense that the vector of predictor variables for a serial execution of two applications is the sum of vectors for the individual execution of each application. Furthermore, the model must have non-negative coefficients and zero intercept.

We studied the *additivity* of PMCs offered by two popular tools, *Likwid* and *PAPI*, using a detailed statistical experimental methodology on a modern Intel Haswell multicore server CPU. We showed that many PMCs in Likwid and PAPI are *non-additive* and that some of these PMCs are key predictor variables in energy predictive models thereby bringing into question the reliability and reported prediction accuracy of these models. We showed that a PMC can be non-additive with error as high as 3075% and there are many PMCs where the error is over 100%.

We discovered that the number of non-additive PMCs rises with an increase in the number of cores employed in the application. We consider this to be an inherent trait of modern multicore computing platforms because of severe resource contention and non-uniform memory access (NUMA).

We summarized the assumptions behind the existing models and used a model-theoretic approach to formulate their assumed properties in a mathematical form. We extended the formalism by adding properties, heretofore unconsidered, that are basic implications of the universal energy conservation law. The extended formalism forms our theory of *energy of computing*. We term an energy predictive model satisfying all the properties of the extended model a *consistent* energy model. Using the theory, we proved that a consistent energy predictive model is linear if and only if its each PMC variable is *additive* in the sense that the PMC for a serial execution of two applications is the sum of PMCs for the individual execution of each application. The basic practical implications of the theory for improving the prediction accuracy of linear energy predictive models are unified in a *consistency* test, which contains a suite of properties that include determinism, reproducibility, and additivity to select model variables and constraints for model coefficients.

We applied the practical implications of our theory to improve the prediction accuracy of the state-of-the-art energy predictive models. We studied

the additivity of PMCs on a modern Intel platform. We showed that a PMC can be non-additive with error as high as 3075% and there are many PMCs where the error is over 100%. We discovered that the number of non-additive PMCs rises with an increase in the number of cores employed in the application. We consider this to be an inherent trait of modern multicore computing platforms because of severe resource contention and non-uniform memory access (NUMA).

We demonstrated how the accuracy of energy predictive models built using linear regression can be improved by selecting PMCs based on a property of additivity. We selected six PMCs which are common in the state-of-the-art energy predictive models and which are positively correlated with dynamic energy consumption. We constructed seven linear regression models with the PMCs as predictor variables and that pass the constraints. We demonstrated that the prediction accuracy of the models improves as we removed one by one from them highly non-additive PMCs. We also highlighted the drawbacks of pure fitting exercise (for example: applying linear regression) without understanding the true physical significance of a predictor variable. We showed that linear regression methods select PMCs based on high positive correlation with dynamic energy consumption and ignore PMCs that have a high significance in terms of contribution to dynamic energy consumption (due to high additivity) thereby impairing the prediction accuracy of the models.

We demonstrated that high positive correlation of the model variables with dynamic energy consumption alone is not sufficient to provide good prediction accuracy for a model but the model variables must also satisfy the properties of the consistency test that take into account the physical significance of the model variables originating from the conservation of energy of computing, which is the manifestation of the fundamental physical law of energy conservation.

We explored the construction of most accurate PMC-based application-specific models on our platform. The results show that the prediction errors are the least for the model with five PMCs for DGEMM (13.41%) and the model with six PMCs for FFT (19.21%). Since the most accurate models employ five and six PMCs as predictor variables, at least six hardware registers must be

dedicated to storing the PMCs so that the models can be employed online. Currently, 3-4 hardware registers are dedicated to storing PMCs during an application run on our experimental platforms thereby hindering the employment of accurate online energy predictive models.

We studied the prediction accuracy of platform-level and socket-level energy predictive models for data-parallel applications executing on a multi-socket multicore CPU platform where the sockets are independently powered. The study demonstrates that socket-level models employing socket-level PMCs, which represent the resource utilization of individually powered components, yield a more accurate energy predictive model.

Finally, we studied the optimization of a parallel matrix-matrix multiplication application for dynamic energy using two measurement tools, IntelRAPL [30], which is a popular mainstream tool, and power meters (HCLWattsUp [132]) providing accurate system-level physical power measurements. We demonstrated that a significant amount of energy (up to 84% for applications used in the experiments) is lost by using IntelRAPL most likely because it does not take into account the properties of the theory of energy predictive models for computing (we found no explicit evidence that it does).

Chapter 5

A Comparative Study of Techniques for Energy Predictive Modelling using Performance Monitoring Counters on Modern Multicore CPUs

A theory of energy predictive models for computing has progressively matured in the previous chapters starting with proposal of a criterion for selection of PMCs followed by a formal description of the theory and its practical implications in Chapter 4. We proposed a novel property of PMCs called *additivity*, which is true for PMCs, whose value for a serial execution of two applications is equal to the sum of values for the individual execution of each application, and study the additivity of PMCs offered by the popular state-of-the-art tools, Likwid [38] and PAPI [37] on a modern Intel Haswell multicore server CPU. A linear predictive energy model is consistent if and only if its predictor variables are *additive* in the sense that the vector of predictor variables for a serial execution of two applications is the sum of vectors for the individual execution of each application. The property, therefore, is based on a simple and intuitive rule that the value of a PMC for a serial execution of two applications is equal

to the sum of its values obtained for the individual execution of each application. They show that many PMCs in Likwid and PAPI that are widely used in models as key predictor variables are non-additive.

We further proposed a novel theory of energy predictive models for computing and its practical implications to improve the prediction accuracy of linear energy predictive models. The implications are unified in a *consistency* test, which contains a suite of properties that include determinism, reproducibility, and additivity to select model variables and constraints for model coefficients. The authors show that failure to satisfy the requirements of the test worsens the prediction accuracy of linear energy predictive models.

In this chapter, we compare two types of energy predictive models constructed from the same set of experimental data and at two levels, platform and application. The first type contains linear regression (LR) models employing PMCs selected using the theoretical model of the energy of computing. The second type has sophisticated statistical learning models, random forest (RF), and neural network (NN), that are based on PMCs selected using correlation and principal component analysis [1].

We divide the experiments in this chapter into two main groups: Group 1 and Group 2. In Group 1, we experimentally compare the prediction accuracy of platform-level energy predictive models on HCLServer1 (Table 3.1). The models are analyzed in two configurations. In the first configuration, the models are trained and tested using datasets that contain all the applications. In the second configuration, the dataset of applications is split into two datasets, one for training models and the other for testing models. We demonstrate that LR models exhibit better prediction accuracies than RF and NN models in both the configurations (5.09$\times$ and 4.37$\times$ times specifically for the first configuration).

In Group 2, we study the accuracy of application-specific energy predictive models using HCLServer2 (Table 3.1). This group also contains models of the two types. We choose two well-known and highly optimized scientific kernels offered by the Intel Math Kernel Library (MKL), 2D fast Fourier transform (FFT) and dense matrix multiplication (DGEMM). We select a set of nine most *addi-*

[1] This chapter is chiefly based on [146] and [147].

tive PMCs (PA) and a set of nine PMCs that are *non-additive* (PNA) that are common for both the applications. PNA belongs to the dominant PMC groups reflecting the energy-consuming activities and have been widely employed in the models found in the literature (Section 2.3.5). We build LR models employing PA and PNA. We demonstrate that the models based on PA have better prediction accuracy than the models based on PNA. To build online energy predictive models based on four PMCs, we compose two subsets of PMCs, *PA4* and *PNA4* from *PA* and *PNA*, containing four PMCs highly positively correlated with energy. Models that use *PA4* exhibit 3.44$\times$ and 1.71$\times$ better average prediction accuracy than models using *PNA4*. We conclude, therefore, that a high positive correlation with dynamic energy consumption alone is not sufficient to provide good prediction accuracy but should be combined with methods such as *additivity* that take into account the physical significance of the model variables originating from the theory of energy conservation of computing. For the same two applications, we compare the LR models based on the set of four most additive and highly positively correlated PMCs (PA4) with the RF and NN models based on four PMCs selected using correlation and PCA. The results show that the LR model performs 1.57$\times$ and 1.74 $\times$ times better than RF and NN models.

Based on our experiments, we conclude that linear regression models based on PMCs selected using the theoretical model of energy of computing perform better than RF and NN models using the standard statistical approaches.

To summarize, our key contribution in this work is that we present the first comprehensive experimental study comparing linear regression models employing PMCs selected using a theoretical model of energy of computing with sophisticated statistical learning models, random forest and neural network, that are constructed using PMCs selected based on correlation and principal component analysis. We show that the LR models perform better than the RF and NN models thereby highlighting two important points. First, the consistent accuracy of LR models highlight the importance of taking into account domain-specific knowledge for model variable selection, in this case, the physical significance of the PMCs originating from the theory of energy predictive

models for computing. Second, according to the theory of energy predictive models for computing, any non-linear energy model (in this case, the RF and NN models) employing PMCs only will be inconsistent and hence inherently inaccurate. A non-linear energy model, in order to be accurate, must employ non-additive model variables in addition to PMCs.

The rest of this chapter is organized as follows. First we present the terminology and then recap the practical implications of the theory of energy predictive models for computing. We then present our experimental setup including the platform and application details, tools and modelling techniques. The following section presents the experimental results and discussions. Finally, last section concludes the chapter.

5.1 Terminology Related to Energy, Prediction Error Measures, and Statistical Techniques

5.1.1 Energy Consumption

Total energy consumption can be represented as a sum of static energy and dynamic energy. We determine the static energy consumption by multiplying the base or idle power of the system (i.e., with no running application) with the application's execution time. However, we calculate the dynamic energy consumption (energy consumption of the application) by subtracting the static energy from the total energy utilized by the system during the application execution. In other words, if P_S represents the base or idle power of the system, E_T is the total energy consumption of the system during an application run for T_E seconds, then the dynamic energy consumption E_D can be determined by using Equation 5.1.

$$E_D = E_T - (P_S \times T_E) \tag{5.1}$$

The rationale backing the use of dynamic energy consumption rather than total energy consumption is given in the Appendix A.1.

5.1.2 Prediction Error Measures

We compare the prediction accuracy of models using two measures: a) Relative error, and b) Proportional error. The relative error p of a predicted dynamic energy consumption e with respect to the ground truth dynamic energy consumption r is given below:

$$p = \frac{|r - e|}{r} \times 100 \tag{5.2}$$

The measure p gives a lower relative error for a model that underestimates than a model that overestimates (for example: when you consider the same proportion for the underestimated and the overestimated values of e with r). This can negatively impact the interpretation of the results. Juan-Antonio et al. [148] propose the proportional error μ to correct the anomaly. The proportional error for model prediction e with the ground truth r is a ratio of a maximum of the two values with the minimum of the two values. It is represented by the following equation 5.3.

$$\mu = \frac{max(r, e)}{min(r, e)} \tag{5.3}$$

μ is always greater than 1 if there exists an error, and equal to 1 otherwise.

5.1.3 Model Variable Selection Techniques

We employ two model variable selection methods for random forest and neural network models. They are: 1). Correlation, and 2). Principal Component Analysis (PCA).

Correlation is a statistical metric to understand the relationship between two variables and is calculated using the following equation 5.4.

$$C_{ep} = \frac{\sum(e_i - \overline{e})(p_i - \overline{p})}{\sqrt{\sum(e_i - \overline{e})^2 \sum(p_i - \overline{p})^2}} \tag{5.4}$$

where, C_{ep} is the correlation coefficient between the dynamic energy consumption e and the PMC p_i. e_i represents energy consumption of an application and $\overline{e}$ is the mean of the energy of all the applications in the data-set.

p_i is the PMC count and $\overline{p}$ is its mean for all the applications in the data-set. The value of the correlation coefficient is between -1 to 1. A value of -1 for C_{ep} means perfect negative correlation, 0 signifying no correlation, and +1, a perfect correlation between the energy and the PMC.

Principal Component Analysis (PCA) [149] is applied to determine the most statistically influential PMCs. It is a multivariate statistical technique for feature extraction and is used for dimensionality reduction in high-dimensional data. It uses a correlation matrix to ease the analysis by selecting the most valuable features in a data-set. The top principal component captures the maximum variability in the data, and each succeeding component has the highest variability subject to the constraint imposing orthogonality with the previous principal components.

5.2 Theory of Energy Predictive Models for Computing: Practical Implications

A theory of energy predictive models for computing has been progressively developed starting with the proposal of a criterion for selection of PMCs in the research work [54] followed by a formal description of the theory and its practical implications for improving the prediction accuracy of linear energy predictive models in [135].

The theory of energy predictive models for computing is a formalism containing properties of PMC-based energy predictive models that are manifestations of the fundamental physical law of energy conservation. The properties capture the essence of single application runs and characterize the behavior of serial execution of two applications. They are intuitive and experimentally validated and are formulated based on the following observations:

- In a stable and dedicated environment, where each run of the same application is characterized by the same PMC vector, for any two applications, the PMC vector of their serial execution will always be the same.
- An application run that does not perform any work does not consume or

generate energy. It is represented by a null PMC vector (where all the PMC values are zeroes).

- An application with a PMC vector that is not null must consume some energy. Since PMCs account for energy consuming activities of applications, an application with any energy consuming activity higher than zero activity must consume more energy than zero.

- Finally, the consumed energy of compound application is always equal to the sum of energies consumed by the individual applications. A compound application is defined as the serial execution of two applications, which we call the base applications.

The practical implications of the theory for constructing accurate and reliable linear energy predictive models are unified in a *consistency* test. The test includes the following selection criteria for model variables, model intercept, and model coefficients:

- Each model variable must be deterministic and reproducible. In the case of PMC-based energy predictive models, the multiple runs of an application keeping the operating environment constant must return the same PMC count.

- Each model variable must be additive. The property of additivity is further summarized in the following section.

- The model intercept must be zero.

- Each model coefficient must be positive.

The first two properties are combined into a *additivity* test for the selection of PMCs. A linear energy predictive model employing PMCs and which violates the properties of the consistency test will have poor prediction accuracy.

By definition and intuition, PMCs are all pure counters of energy-consuming activities in modern processor architectures and as such must be additive. Therefore, according to the theory of energy predictive models for

computing, any consistent, and hence accurate, energy model, which only employs PMCs, must be linear. This also means that any non-linear energy model employing PMCs only, will be inconsistent and hence inherently inaccurate. A non-linear energy model, in order to be accurate, must employ non-additive model variables in addition to PMCs.

5.2.1 *Additivity* of PMCs

The property of *additivity* is based on a simple and intuitive rule that if a PMC is intended as a model variable in a linear energy predictive model, then its value for a *compound* application should be equal to the sum of its values for the executions of the base applications constituting the compound application. It is based on the experimental observation that the dynamic energy consumption of a serial execution of two applications is the sum of dynamic energy consumption observed for the individual execution of each application.

The additivity of a PMC is determined as follows. At first, we collect the values of the PMC for the base applications by executing them separately. Then, we execute the *compound* application and obtain its value of the PMC. Typically, the core computations for the compound application consist of the core computations of the base applications programmatically placed one after the other. If the PMC of the *compound* application is equal to the sum of the PMCs of the base applications (with a tolerance of 5.0%), we classify the PMC as potentially *additive*. Otherwise, it is *non-additive*.

For each PMC, we determine the maximum percentage error. For a *compound* application, the percentage error is calculated as follows:

$$Error(\%) = |\frac{(e_{b1} + e_{b2}) - e_c}{(e_{b1} + e_{b2} + e_c)/2}| \times 100 \tag{5.5}$$

where e_c, e_{b1}, e_{b2} are the PMCs for the compound application and the constituent base applications respectively. Additivity test error for a PMC is the maximum of percentage errors for all the *compound* applications in the experimental test-suite.

We use a tool called *AdditivityChecker* (Appendix B.2), that automates the

determination of the additivity value of a PMC.

5.3 Experimental Setup

5.3.1 Evaluation Platform

The experiments are carried out on two modern multi-core platforms: 1). Intel Haswell dual-socket server, and 2). Intel Skylake single-socket server. The specifications for both are given in Table 3.1.

5.3.2 Experimental Applications

Our test suite (Table 3.2) comprises a diverse set of benchmarks containing highly memory-bound and compute-bound scientific computing applications such as DGEMM and FFT from Intel math kernel library (MKL), scientific applications from NAS Parallel benchmark suite, Intel HPCG, *stress*, and two unoptimized applications.

5.3.3 Experimental Tools

We measure the following during an application execution: 1). Dynamic energy consumption, 2). Execution time, and 3). PMCs. The experimental workflow is shown in the Figure 5.1. The dynamic energy consumption is determined using system-level power measurements provided by WattsUp pro power meter. The readings are obtained programmatically using a detailed statistical methodology employing HCLWattsUp API [132]. The power meters are periodically calibrated using an ANSI C12.20 revenue-grade power meter, Yokogawa WT210. To ensure the reliability of our results, we follow a statistical methodology where a sample mean for a response variable is obtained from several experimental runs. We follow a strict statistical methodology to ensure the reliability of our experiments (Appendix A.3).

We use *Likwid* package [38] to obtain the PMCs. It offers 164 PMCs and 385 PMCs on Intel Haswell and Intel Skylake platform, respectively. We elim-

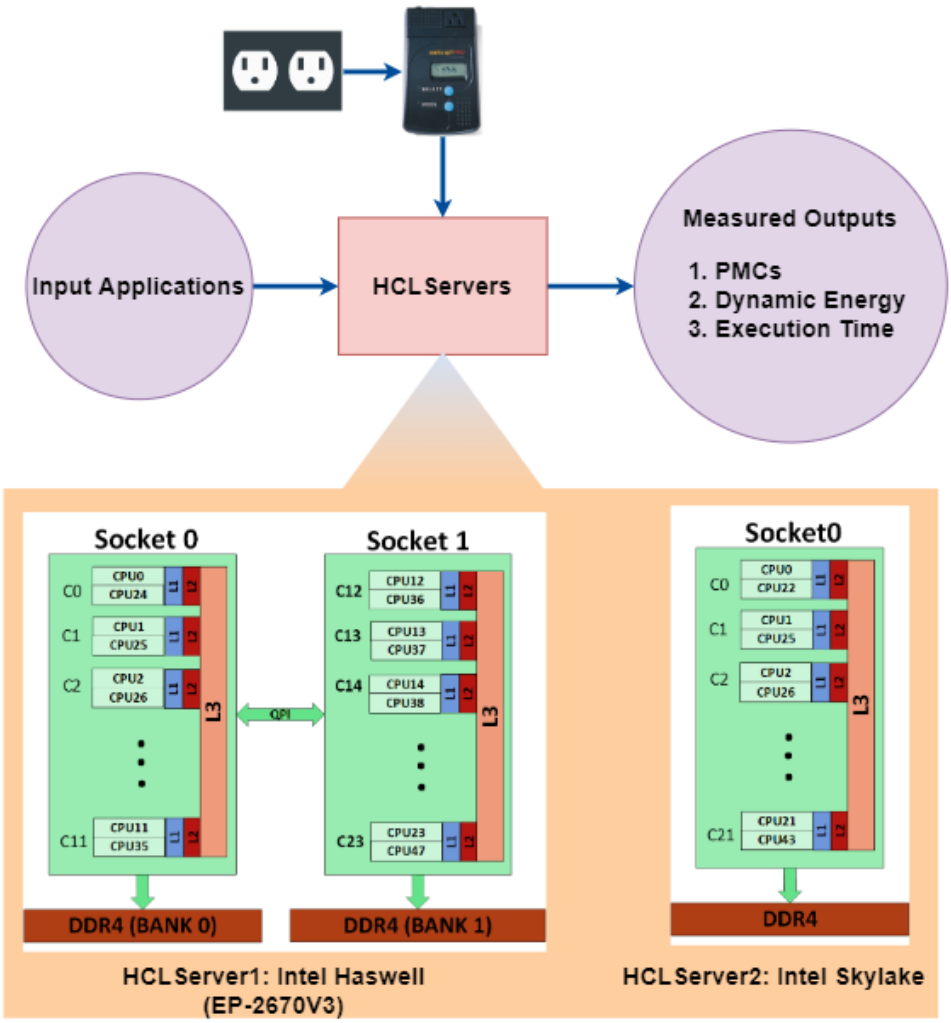


Figure 5.1: Experimental workflow to determine the PMCs for our HCLServer platforms.

inate PMCs with counts less than or equal to 10. These PMCs have no significance in modelling the dynamic energy consumption of our platform since they are non-reproducible over several runs of the same application on our platform.

The reduced set contains 151 PMCs for Intel Haswell and 323 for Intel Skylake. The collection of all of them is tedious since only four PMCs can be obtained in a single application run. This is because of a limited number of hardware registers dedicated to storing them. We also notice that some PMCs can only be collected individually or in sets of two or three for a single execution of an application. Therefore, we observe that each application must be executed about 53 and 99 times on Intel Haswell and Intel Skylake platform, respectively, to collect all the PMCs.

5.3.4 Energy Predictive Modelling Techniques

Table 5.1: Modelling Parameters

Linear Regression	
Estimation Method	Least-squares
Model Coefficients	Positive
Intercept	0
Random Forests	
Software Tool	R Programming Studio
Variables in each split	3
Type	Regression
Number of Trees	500
Neural Networks	
Network type	Feed-forward back propagation
Input parameters	PMCs
Output parameters	Dynamic energy
Training algorithm	Bayesian regularization
Performance function	Mean-Squared Error (MSE)
Number of neurons	20
Network layers	2
Activation function	Linear

The two types of energy predictive models employed in our work are described below.

- **Linear Regression (*LR*)**: A *LR* based model can be represented as:

  $$Y_i = \sum_{j=0}^{M} \beta_j X_{ij} + \epsilon_i$$

  where, $i = 1, 2, ..., N$ represent the number of observations and $j = 1, 2, ..., M$ represent the number of independent variables.

  In our models, Y_i are dynamic energy measurements obtained using HCLWattsUp API and X_i are the PMCs. ϵ represents the error/noise in measurements. We build a specialized linear model using a regression technique that constrains the coefficients (β) to be positive.

- **Random Forest (*RF*)**: A RF technique is a supervised learning algorithm using a decision tree-based approach to train on a data-set and output mean prediction from individual trees. It is considered for its accuracy in classification and regression-based tasks [150]. It is a non-linear machine learning model build by constructing many linear boundaries. The overall non-linearity is because a single linear function can not be used to classify and regress on each iteration of the decision tree.

  We apply the *RF* based regression to the various data-set and determine its prediction accuracy for PMCs in different experimental configurations.

- **Neural Networks (*NN*)**: A *NN* model is inspired by neurons of a human brain and contains an interconnected group of nodes where each node computes weights and biases and give an output prediction. The learning function is Bayesian regularization that gives optimal regularization parameters in an automated fashion [151]. Bayesian regularization updates the weight and biases by using the Levenberg-Marquardt algorithm [152], which is used to train the NN up to 100 times quicker in comparison with the commonly used gradient-descent and back-propagation method. We set the activation function as linear. It's an indicative implementation; a motivated neural network expert would likely produce a better network than this.

The training parameters employed to build the models are given in Table 5.1. Figure 5.2 explains the machine learning model pipeline. It has four main stages: 1). Data collection, 2). PMC selection, 3). Model training, and 4). Model testing or validation. After the collection of the data-set from HCLServers, the data is passed through a PMC selection stage which first normalizes the PMC counts. To construct the LR models, PMCs are first checked for their additivity using the Additivity Test and the top most additive PMCs are selected as model variables. To build the RF and NN models, the PMCs are first evaluated based on their statistical correlation. The set of top positively correlated PMCs is then further pruned using PCA. The correlation and PCA methods are explained in detail in the section 5.4. The set of selected PMCs is then split into two subsets. One for training the models and the other for testing their accuracy.

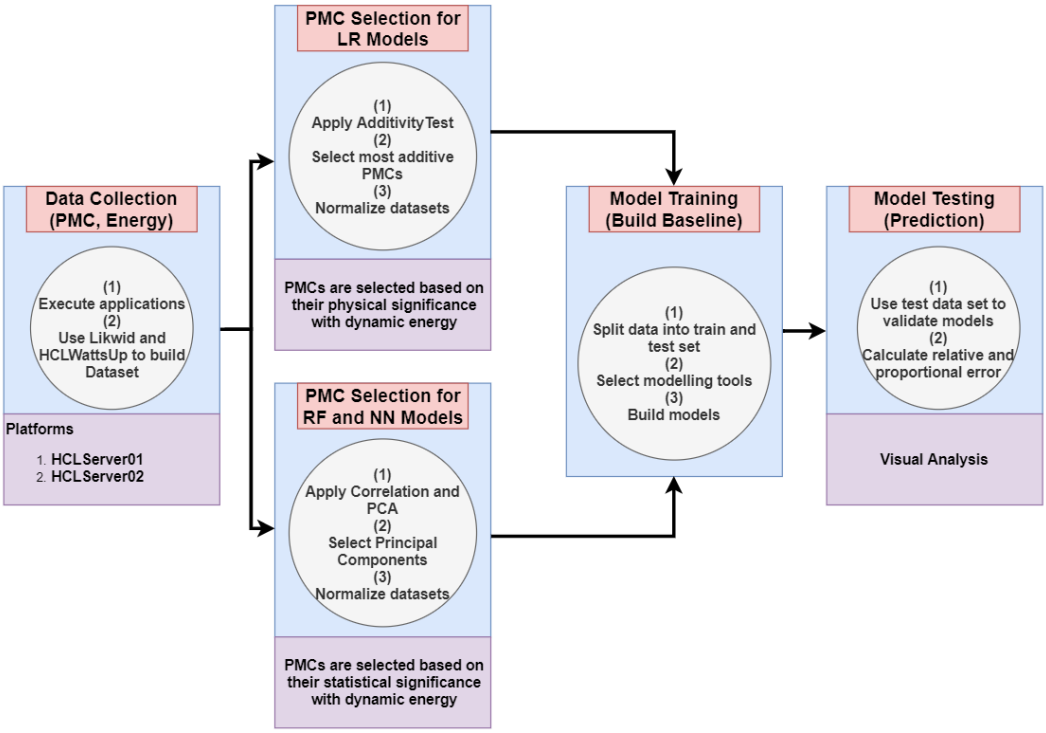


Figure 5.2: Machine Learning Model Building and Evaluation Pipelines for LR, RF and NN

5.3.5 Selection Methods for PMCs

We now summarize the steps to select model variables or PMCs using two approaches as described below:

1. PMCs are selected based on the consistency test from the theory of energy predictive models for computing [135] and employed as model variables in linear regression (LR) models. The steps for the PMC selection include:
    - The most additive PMCs for a set of applications are selected.
    - During the execution of the applications, the individually powered computing components (memory and CPU) with activities that result in dynamic energy consuming are identified.
    - The most additive PMCs that belong to computing components contributing towards dynamic energy consumption are then selected as model variables.
2. PMCs are selected using correlation and PCA based statistical methods and then employed in non-linear models such as random forest (RF) and neural network (NN). The selection method is composed of two stages:
    - In the first stage, we list all the PMCs in the increasing order of positive correlation with dynamic energy consumption. We select all the PMCs with a correlation coefficient of over 0.90.
    - In the second stage, we apply the principal component analysis (PCA) on the PMCs selected in the first stage to pick the most statistically influential PMCs. Figure 5.3 illustrates this PMC selection process.

5.4 Experimental Results

We divide our experiments into two groups, Group 1 and Group 2, as follows:

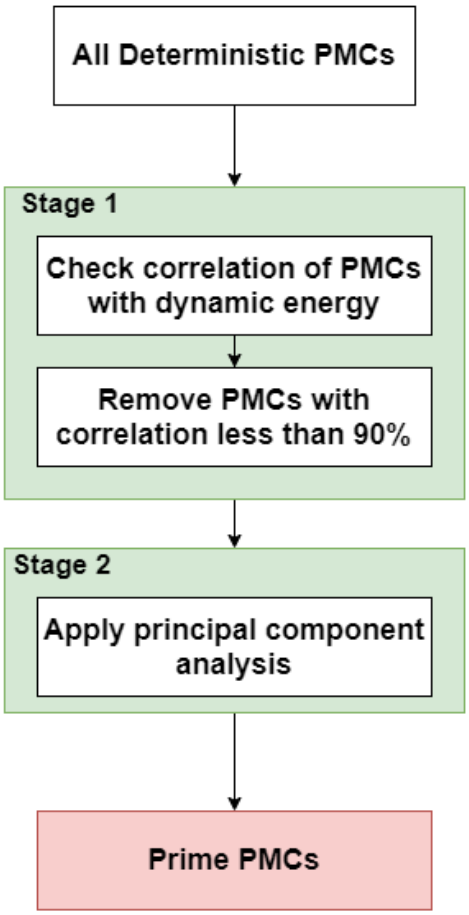


Figure 5.3: PMC selection process using Statistical Methods.

- **Group 1:** We employ this group to study the prediction accuracy of the platform-level energy predictive models. We use two experimental configurations. In the first configuration, we split the full data-set representing all the applications into two subsets, one for the training and the other for testing. The training and test data-sets contain data points encompassing all the applications. In the second configuration, the models are trained on a data-set for one set of applications and tested against a different set of applications. The experiments are performed on HCLServer1 (Table 3.1).
- **Group 2:** We employ this group to study the accuracy limits of the application-level energy predictive models. Two highly memory-bound and compute-bound scientific computing applications, DGEMM and FFT from Intel MKL, are used for this purpose. The experiments are performed on HCLServer2 (Table 3.1).

Group 1: Comparison of Prediction Accuracy of Platform-Level Energy Predictive Models

Using a diverse application set (Table 3.2), we build platform-level energy predictive models employing model variables that are selected using two aforementioned approaches, consistency test, and statistical methods.

Energy Predictive Models Using Consistency Test

Table 5.2: List of selected PMCs and their additivity *test errors (%).*

Selected PMCs	***Additivity* Test Error(%)**
PL1: AVX_INSTS_ALL	7
PL2: UOPS_EXECUTED_PORT_PORT_6	8
PL3: IDQ_MITE_UOPS	11
PL4: CPU_CLOCK_THREAD_UNHALTED	13
PL5: L2_RQSTS_MISS	17
PL6: BR_INST_RETIRED_ALL_BRANCHES	18

Table 5.3: Linear regression models with their minimum, average, and maximum prediction errors.

Model	PMCs	Relative Errors p in [%] (Min, Avg, Max)	Proportional Errors μ (Min, Avg, Max)
LR1	PL1,PL2,PL3,PL4,PL5,PL6	(2.1, 27.9, 85.1)	(1.04, 1.703, 5.190)
LR2	PL1,PL2,PL3,PL4,PL5	(1.9, 26.01, 82.91)	(1.02, 1.61, 5.021)
LR3	PL1,PL2,PL3,PL4	(1.9, 26.01, 82.91)	(1.02, 1.61, 5.021)
LR4	PL1,PL2,PL3	(1.21, 25.15, 80.2)	(1.01, 1.56, 4.91)
LR5	PL1,PL2	(0.66, 21.80, 71)	(1.003, 1.382, 4.65)
LR6	PL1	(0.96, 24.6, 79)	(1.004, 1.50, 4.85)

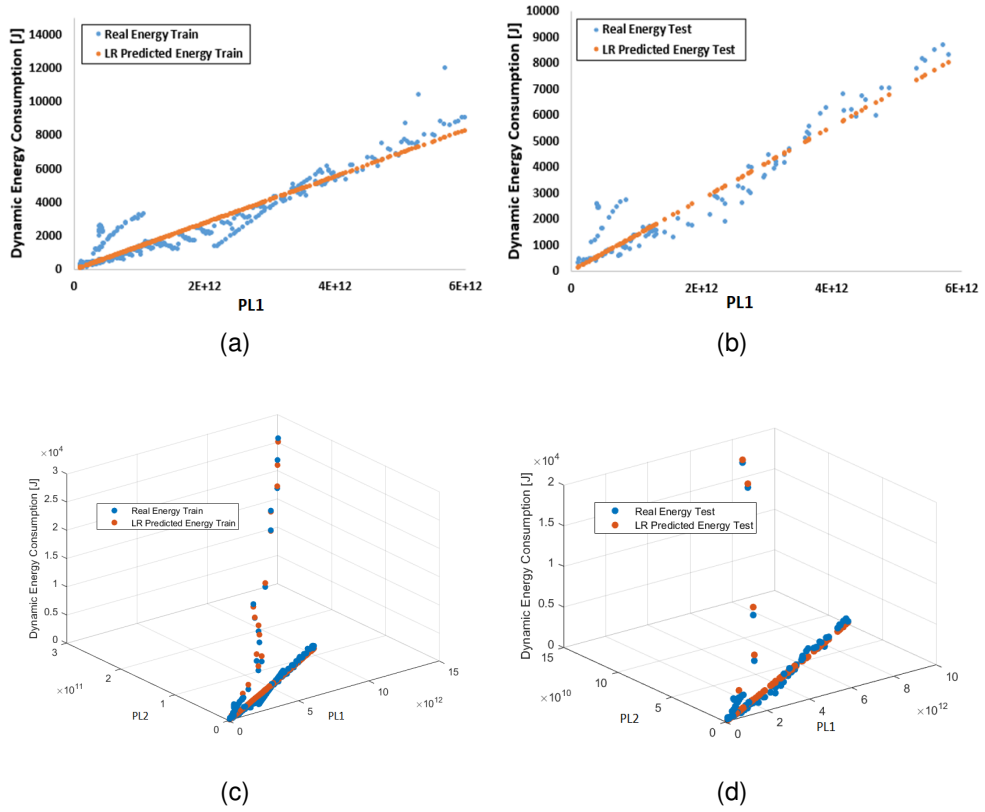


Figure 5.4: Real and predicted dynamic energy consumptions using HCLWattsUp and linear regression models versus (a). PMC PL1 for train set applications, (b). PMC PL1 for test set applications, (c). PMCs, PL1 and PL2, for train set applications, and (d). PMCs, PL2 and PL2, for test set applications.

Table 5.4: Prediction accuracies for linear regression models for configuration A2.

Model	**Data-set**	**Relative Errors p in [%] (Min, Avg, Max)**	**Proportional Errors μ (Min, Avg, Max)**
LR5-A2	Training	(1.19, 25, 136)	(1.002, 1.527, 4.77)
LR5-A2	Testing	(1.03, 24, 101)	(1.009, 1.416, 4.71)

The experimental methodology for measuring and selecting the PMCs follows:

- The PMCs are obtained using Likwid tool, which classifies them into performance groups. The list of the performance groups is given in Appendix B.1. We apply the first step of the consistency test, which is to check if the PMCs are deterministic and reproducible using the following two steps:
  - PMCs with counts near zero are removed. These PMCs have no statistical significance on modelling energy consumption of our platform because we found them to be non-reproducible. Several PMCs with counts equal to zero are also removed. The reduced set contains 151 and 298 PMCs on Intel Haswell and Intel Skylake, respectively.
  - We broadly compare the PMCs obtained using Likwid, PAPI, and Linux Perf. We remove PMCs that show different counts for different tools. The final set contains 115 and 224 PMCs on Intel Haswell and Intel Skylake platform.
- We discover that all the work performed during the execution of the applications in our test suite is due to CPU and memory activities. We run a set of experiments to evaluate the contribution of both of these components towards the dynamic energy consumption. We summarize them below:
  - We execute a synthetic application (app-cpu) performing floating-point operations on all the processor cores for 10 seconds and

measure its dynamic energy consumption. HCLWattsUp reports the dynamic energy consumption to be 1337 joules.

  - We then execute another synthetic application (app-mem) performing *memcpy()* operations on all the memory blocks for 10 seconds and measure the dynamic energy consumption. We find the energy consumption to be insignificant and can not be measured within the statistical confidence of 95%.
  - We further execute app-cpu for 20 seconds and 30 seconds and found the dynamic energy consumption to be equal to 2596 joules and 3821 joules, respectively. However, the execution of app-mem for 20 and 30 seconds results in dynamic energy consumption less than 5 joules.

- Based on the above experiments, we remove the PMCs that belong to Likwid main memory group for any further analysis due to two reasons. First, the memory activities do not reflect any contributions to the dynamic energy consumption on our platforms. Second, low counts for memory PMCs add noise that affects the training of models and unduly worsen the prediction accuracy of the models.

- The CPU activities during the application run are represented by PMCs that belong to the following dominant groups: cache, branch instructions, micro-operations (uops), floating-point instructions, instruction decode queue, and cycles.

- We then study the additivity of PMCs belonging to the dominant groups.

  - We build a data-set of 277 points as base applications by executing the applications from our test suite with different problem sizes. Each point contains the dynamic energy consumption and PMCs corresponding to the base applications.
  - We execute another set of 50 compound applications from the serial combination of base applications and record their dynamic energy consumption and PMCs.

  - For all PMCs, we calculate the percentage errors of each compound application with the sum of base applications. The additivity test error for each PMC is the maximum of the percentage errors for all compound applications.

- We found no PMC to be absolutely additive (with the additivity test error of less than 5%), in general, for all applications in the test suite (Table 3.2). Therefore, we select one top additive PMC for each dominant PMC group.

Table 5.2 list the selected PMCs (PL1,$\cdots$,PL6) in the order of increasing additivity test error. The PMC PL1 is highly additive compared to the rest.

We construct a data-set of 448 points for different configurations for applications in our test suite (Table 3.2). We split the data-set into two subsets, 335 points for training the models, and 113 points for testing their prediction accuracy. We used this division based on best practices and expert opinions in this domain.

We build six LR models, {LR1, LR2, LR3, LR4, LR5, LR6}. To impose the constraints of the consistency test, the linear models are built using penalized linear regression using R programming interface that forces the coefficients to be non-negative and to have zero intercepts. The models contain decreasing number of *non-additive* PMCs. Model LR1 employs all the selected PMCs as predictor variables. Model LR2 is based on five most *additive* PMCs. PMC PL6 is removed because it has the highest *non-additivity*. Model LR3 uses four most *additive* PMCs and so on until Model LR6 containing the highest *additive* PMC, which is PL1.

We compare the predictions of the models with system-level physical measurements using HCLWattsUp, *which we consider to be the ground truth* [36]. The minimum, average, and maximum prediction errors for the models are given in Table 5.3. One can see that the accuracy of the models improves as we remove the highest non-additive PMCs one by one until Model LR5, which exhibits the least average (p, μ) of (21.8%, 1.382), respectively. LR5 employs two most additive PMCs, PL1 and PL2. PL1 accounts for the floating-point operations and PL2 accounts for a portion of micro-operations executing inside

the CPU cores during the execution of an application. We observe that LR1 has the worst average (p, μ) of (27.9%, 1.703) due to the poor linear fit.

Figures 5.4(a) and 5.4(b) show the plots for ground truth and predicted dynamic energy consumptions obtained using HCLWattsUp and LR6 against the top additive PMC (that is, PL1) for the train and test data-sets, respectively. Similarly, Figure 5.4(c) and 5.4(d) shows the plots for ground truth and predicted dynamic energy consumptions obtained using HCLWattsUp and LR5 against the top two additive PMCs (that are, PL1 and PL2) for train and test data-sets, respectively. It can be seen that the training dataset and the test dataset include all the applications. Furthermore, the combined use of PL1 and PL2 as model variables in LR5 increases its prediction power and makes it the most accurate and consistent model. This is because only one PMC (despite being most additive) is not able to track all the dynamic energy-consuming activities for applications in a modern multicore CPU.

The model LR5, employing PL1 and PL2 as model variables, is built using a training data set that contains all the applications and is tested against the test dataset that also contains all the applications. Let us denote this split configuration of training and test datasets as A1. We consider a different split configuration, A2. In A2, the test set applications do not include the training set applications. The training set contains 335 points and the test set contains 113 points.

We then build a model LR5-A2 using the training data-set from configuration A2. The minimum, average, and maximum p and μ for train and test set using LR5-A2 are given in Table 5.4. Comparison of the average p and μ for both test sets using LR5 (or LR5-A1) and LR5-A2 shows only a minor increase from 21.8% to 24% and 1.382 to 1.416, respectively. Therefore, we conclude that the accuracy and consistency of the LR model employing the two most additive PMCs is generic and therefore the LR model generalizes well.

Energy Predictive Models Using Statistical Methods

We first present the experimental methodology to select the model variables. The data-set used for this purpose includes 277 base applications. Each ap-

Table 5.5: List of PMCs selected in stage 1 of approach B where the PMCs are listed in the increasing order of positive correlation with dynamic energy consumption.

UOPS	**L2**
UOPS_EXECUTED_TOTAL_CYCLES	L2_RQSTS_ALL_DEMAND_DATA_RD_HIT
UOPS_EXECUTED_CORE	L2_RQSTS_ALL_DEMAND_DATA_RD
UOPS_EXECUTED_CORE_STALL_CYCLES	L2_RQSTS_CODE_RD_HIT
UOPS_RETIRED_CORE_USED_CYCLES	L2_RQSTS_CODE_RD_MISS
BRANCHES	**ICACHE**
BR_INST_RETIRED_ALL_BRANCHES	ICACHE_ACCESSES
BR_MISP_RETIRED_ALL_BRANCHES	**CPU CLOCK**
IDQ	CPU_CLK_UNHALTED_ANY
IDQ_MITE_UOPS	CPU_CLOCK_THREAD_UNHALTED_ONE_THREAD
IDQ_DSB_UOPS	CPU_CLOCK_UNHALTED_TOTAL_CYCLES
IDQ_ALL_DSB_CYCLES_4_UOPS	**AVX**
MEM	AVX_INSTS_ALL
MEM_LOAD_UOPS_RETIRED_ALL_ALL	
MEM_UOPS_RETIRED_ALL	

Table 5.6: List of prime PMCs obtained after applying principal component analysis.

PL7: UOPS_EXECUTED_TOTAL_CYCLES
PL8: UOPS_EXECUTED_CORE
PL9: UOPS_EXECUTED_CORE_STALL_CYCLES
PL10: CPU_CLOCK_THREAD_UNHALTED_ONE_THREAD
PL11: CPU_CLOCK_UNHALTED_TOTAL_CYCLES
PL12: BR_INST_RETIRED_ALL_BRANCHES

Table 5.7: Prediction accuracies for random forest models.

Model	**Data-set**	**Relative Errors p in [%] (Min, Avg, Max)**	**Proportional Errors μ (Min, Avg, Max)**
RF-A1	Training	(1.12, 26.6, 63)	(1.003, 2.13, 8.51)
RF-A1	Testing	(1.24, 28.7, 57)	(1.002, 3.61, 8.22)
RF-A2	Training	(2.37, 25.2, 61)	(1.001, 2.92, 7.76)
RF-A2	Testing	(3.92, 37, 293)	(1.001, 7.21, 21.41)

Table 5.8: Prediction accuracies for neural network models.

Model	**Data-set**	**Relative Errors p in [%] (Min, Avg, Max)**	**Proportional Errors μ (Min, Avg, Max)**
NN-A1	Training	(2.31, 26.6, 65)	(1.002, 3.32, 9.12)
NN-A1	Testing	(2.09, 25.1, 58)	(1.01, 2.21, 6.28)
NN-A2	Training	(2.17, 25.4, 57)	(1.005, 2.56, 5.39)
NN-A2	Testing	(4.96, 44, 492)	(1.033, 6.19, 19.74)

plication is represented by a data point that contains the dynamic energy consumption and 151 PMCs. We apply the first stage of PMC selection using statistical methods shown in Figure 5.3. We list all the PMCs in the increasing order of positive correlation with dynamic energy consumption. All the PMCs with a correlation coefficient of over 0.90 are then selected. The selected PMCs are listed in Table 5.5 based on their groups. In the second stage, we apply the principal component analysis (PCA) on the selected PMCs from the first stage. The most statistically influential PMCs obtained after stage 2 are termed as prime PMCs, which are shown in Table 5.6.

We employ prime PMCs in RF and NN models using a data-set of 448 points. Each point in a data-set contains dynamic energy consumption for an application with particular input and the prime PMCs. We divide the data-set into a training dataset (335 points) and a test dataset (113 points). The splitting is done using two aforementioned configurations, A1 and A2.

We build two sets of models, RFS = {A1-RF, A2-RF}, and NNS = {A1-NN, A2-NN}. The RR and NN parameters used to build the models are given in Table 5.1. We compare the predictions of the models with the ground truth. Tables 5.7 and 5.8 show the minimum, average, and maximum p and μ from RFS and NNS. Figures 5.5(a) and 5.5(b) compare the average prediction accuracies of the most accurate LR model with the RF and NN models. The least average p and μ is obtained for the LR model with two model variables, that is, 21.80% and 1.38. RF-A2 and NN-A2 for the test set yields the highest average (p, μ) of (37%, 7.21) and (44%, 6.19), respectively.

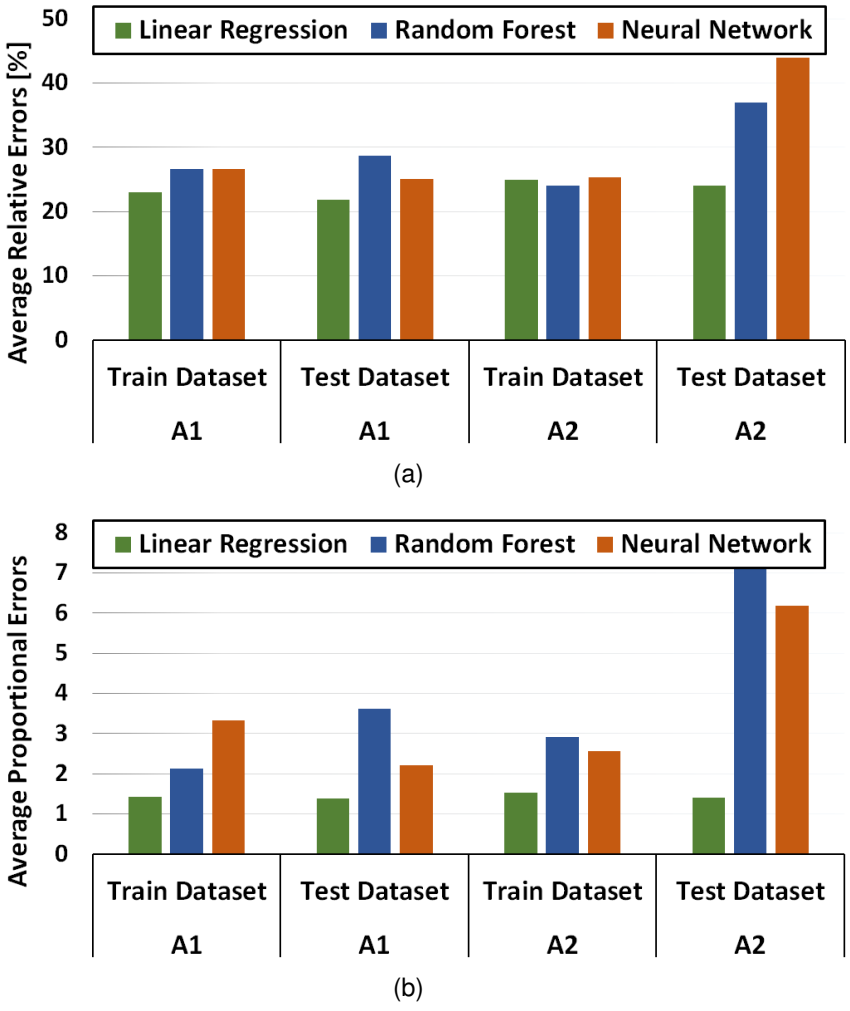


Figure 5.5: Comparison of (a). average relative prediction accuracies, and (b). average proportional prediction accuracies, for LR5, RF, and NN.

Discussion

Following are the salient observations from the results:

- The PMCs selected for training the models should represent the dynamic energy-consuming activities during the application execution. We discover that, in our platform, the contribution of memory-centric operations towards the dynamic energy consumption is insignificant. Therefore, we use CPU-centric PMCs such as floating-point operations and micro-operations originating from processor core for training the energy predictive models.
- The average relative and proportional prediction errors for LR is better than RF and NN for all the models. There is a significant difference in the average prediction errors for test applications in the configuration A2 where the test applications do not include the training set applications. RF and NN models perform poorly for A2. The average relative prediction accuracy of the best LR model only degrades by 2%. We conclude that a machine learning-based platform-level energy predictive model employing PMCs selected using statistical methods provide good prediction accuracy when data points in the train set and test set belong to the same set of applications. This is because of their ability to memorize well the input domain of energy values of the applications. However, their accuracy suffers when the train and test data sets contain different sets of applications suggesting their inability to provide good prediction accuracy for a general set of applications (that is, to generalize well).
- The average prediction accuracies (p) for LR is 1.54$\times$ and 1.84$\times$ better than RF and NN, respectively, for test applications in A2. The proportional prediction accuracies (μ) for LR is 5.09$\times$ and 4.37$\times$ better than RF and NN, respectively. This suggests that μ is better than p for the interpretation of results.
- The results highlight two important points. First, the consistent accuracy of LR models stresses the importance of taking into account domain-specific knowledge for model variable selection, in this case, the physical

significance of the PMCs originating from the conservation of energy of computing. Second, according to the theory of energy predictive models for computing, any non-linear energy model (in this case, the RF and NN models) employing PMCs only, will be inconsistent and hence inherently inaccurate. A non-linear energy model, to be accurate, must employ non-additive model variables in addition to PMCs.

Group 2: Comparision of Prediction Accuracy of Application-Level Energy Predictive Models

In this section, we study the accuracy of application-specific energy predictive models. The experiments are conducted HCLServer2 (Table 3.1). We compare the models built using PMCs selected via the aforementioned approaches, consistency test and statistical methods. First, we build LR models that satisfy the properties of the consistency test that is based on the theory of energy conservation of computing. We also build LR models employing non-additive PMCs that belong to the dominant PMCs groups reflecting energy-consuming activities for application execution and that have been widely employed by energy models found in the literature (Section 2.3.5). Finally, we build RF and NN models employing PMCs selected using statistical methods such as correlation and PCA.

We now present the experimental methodology using consistency test to build LR models:

- Out of the 385 PMCs available for this platform, we found no PMC to be *additive*, in general, within tolerance of 5% for all applications in our test suite (Table 3.2). However, many PMCs are highly additive for each application. We select for two highly optimized scientific kernels: Fast Fourier Transform (FFT) and Dense Matrix-Multiplication application (DGEMM), from Intel Math Kernel Library (MKL).

- We check if the PMCs are reproducible and deterministic by running the same application several times without any change in the operating environment. The PMC is considered to be reproducible and deterministic

Table 5.9: Additive and non-additive PMCs highly correlated with dynamic energy consumption. 0 to 1 represents positive correlation of 0% to 100%.

	***Additive* PMCs**	**Correlation**
AL1	UOPS_RETIRED_CYCLES_GE_4_UOPS_EXEC	0.992
AL2	FP_ARITH_INST_RETIRED_DOUBLE	0.993
AL3	BR_INST_RETIRED_ALL_BRANCHES	0.860
AL4	UOPS_EXECUTED_CORE	0.993
AL5	UOPS_DISPATCHED_PORT_PORT_4	0.870
AL6	IDQ_DSB_CYCLES_6_UOPS	0.981
AL7	IDQ_ALL_DSB_CYCLES_5_UOPS	0.972
AL8	IDQ_ALL_CYCLES_6_UOPS	0.993
AL9	L2_RQSTS_CODE_RD_HIT	0.821
	***Non-additive* PMCs**	
AL10	ICACHE_64B_IFTAG_MISS	0.960
AL11	CPU_CLOCK_THREAD_UNHALTED	0.600
AL12	BR_MISP_RETIRED_ALL_BRANCHES	0.992
AL13	UOPS_RETIRED_CORE_USED_CYCLES	0.751
AL14	FRONTEND_RETIRED_L2_MISS	0.806
AL15	ITLB_MISSES_STLB_HIT	0.111
AL16	L2_TRANS_CODE_RD	0.860
AL17	IDQ_MS_UOPS	0.99
AL18	ARITH_DIVIDER_COUNT	0.986

Table 5.10: Prediction accuracies of LR models using nine PMCs.

Model	**PMCs**	**Relative Errors p in [%] (Min, Avg, Max)**	**Proportional Errors μ (Min, Avg, Max)**
LR-A	*PA*	(0.013, 36.11, 226.1)	(1.006, 1.55, 6.53)
LR-NA	*PNA*	(0.513, 86.11, 4073)	(1.022, 2.41, 8.94)

Table 5.11: Prediction accuracies of LR models using four PMCs.

Model	**PMCs**	**Relative Errors p in [%] (Min, Avg, Max)**	**Proportional Errors μ (Min, Avg, Max)**
LR-A4	PA4	(0.024, 25.12, 87.25)	(1, 1.42, 6.49)
LR-NA4	PNA4	(0.449, 85.61, 4039)	(1.009, 2.43, 9)

Table 5.12: List of PMCs obtained for application-specific modelling using correlation.

AL19: BR_INST_RETIRED_ALL_BRANCHES	0.99
AL20: ICACHE_64B_IFTAG_MISS	0.96
AL21: OFFCORE_REQUESTS_ALL_DATA_RD	0.96
AL22: L2_RQSTS_MISS	0.97
AL23: MEM_LOAD_RETIRED_L3_MISS	0.99
AL24: CYCLE_ACTIVITY_CYCLES_MEM_ANY	0.96
AL25: ITLB_MISSES_WALK_COMPLETED	0.95
AL26: IDQ_MS_MITE_UOPS	0.99
AL27: LSD_UOPS	0.97

Table 5.13: Prediction accuracies of application-specific RF and NN models.

Model	**Relative Errors p in [%] (Min, Avg, Max)**	**Proportional Errors μ (Min, Avg, Max)**
RF-MA	(.82, 24.82, 82.7)	(1.004, 2.23, 5.32)
NN-MA	(0.91, 23.95, 91.4)	(1.002, 2.48, 4.17)

if its value for multiple executions of the same application lies within the confidence interval of 95%. We discover that 323 PMCs are reproducible and deterministic.

- A data-set of 50 *base* applications with a range of problem sizes for DGEMM and FFT is used to study the additivity of their representative PMCs. For DGEMM, the problem sizes vary from 6500×6500 to 20000×20000, and for FFT, the range of problem sizes is 22400×22400 to 29000×29000. These problem sizes are selected because of the applications' considerable execution time ($>$ 3 seconds) so that HCLWattsUp can accurately capture the dynamic energy consumptions. A data-set of 30 *compound* applications is build using the serial execution of *base* applications. The two data sets are then given as an input to AdditivityChecker that returns the additivity test errors as an output. We found some PMCs that are additive in common for both applications.

- We select nine PMCs that are highly additive with additivity test errors of less than 1%. We also select nine PMCs that are non-additive for both

the applications but which have been employed as predictor variables in energy predictive models given in the literature (Section 2.3.5). The set of *additive* PMCs are denoted by *PA* and *non-additive* PMCs by *PNA*. In both sets, there are no PMCs from the memory group because of the insignificant contribution of memory activities towards the dynamic energy consumption. The selected PMCs with their correlations are given in Table 5.9.

- By executing DGEMM and FFT, for the problem sizes ranging from 6400×6400 to 38400×38400 and 22400×22400 to 41536×41536, respectively, with a constant step sizes of 64, we build a data-set containing 801 points. We record the dynamic energy consumption and the selected PMCs (Table 5.9) for each application. The data-set is further divided into two subsets, one for training and the other for testing the models. The train data-set contains 651 points and the test data-set contains 150 points.

- We build two linear models, {LR-A, LR-NA}. The model LR-A is trained using PMCs belonging to PA and the model LR-NA is trained using PMCs belonging to PNA. Table 5.10 show the relative and proportional prediction errors of the models. One can see that the models based on PA have better average prediction accuracies than the models based on PNA.

- Since only four PMCs can be collected in a single application run, the selection of such a reliable subset is crucial to the prediction accuracy of online energy models. We use PA and PNA to build two sets of four most energy correlated PMCs. The first set PA4, {AL1, AL2, AL4, AL8}, is constructed using PA and the second set PNA4, {AL10, AL12, AL17, AL18}, using PNA. We build two linear models, {LR-A4, LR-NA4}. The model LR-A4 is trained using PMCs belonging to PA4 and the models LR-NA4 is trained using PMCs belonging to PNA4. The training and test data-sets are the same as before.

- Table 5.11 shows the relative and proportional prediction errors of the

models. We can see that model LR-NA4 built using highly correlated but *non-additive* PMCs do not demonstrate much improvement in average prediction accuracies when compared to model LR-NA based on nine non-additive PMCs. However, LR-A4 performs 1.43$\times$ and 1.09$\times$ better in terms of average relative and proportional accuracies, respectively.

The experimental methodology to select PMCs using statistical methods and building RF and NN models is:

- We select the same two highly optimized scientific kernels and remove the PMCs that are not reproducible and deterministic.
- We build a data-set of 50 applications using different problem sizes for DGEMM and FFT. The range of problem sizes used for DGEMM is 6500×6500 to 20000×20000, and for FFT is 22400×22400 to 29000×29000. We select this range because of reasonable execution time ($>$ 3 seconds) of the applications. We remove all the PMCs that have less than 90% positive correlation with dynamic energy. We then select the topmost correlated PMC from each Likwid group. The selected prime PMCs are {AL19, AL20, ..., AL27} and listed in Table 5.12.
- Since only four PMCs can be employed in an online model, we select the top four principal PMCs by applying principal component analysis. The final list of prime PMCs includes {AL19, AL23, AL26, AL27}.
- We build a data-set containing 801 points representing DGEMM and FFT for a range of problem sizes from 6400×6400 to 38400×38400 and 22400×22400 to 41536×41536, respectively, with a constant step size of 64. We record the dynamic energy consumption and the prime PMCs (AL19, AL23, AL26, and AL27) for each application. The data-set is further divided into two subsets, one for training and the other for testing the models. The train data-set contains 651 points and the test data-set contains 150 points.
- We build a random forest model (RF-MA) and a neural network model (NN-MA) using the training set. Table 5.13 shows the relative and pro-

portional prediction errors of the models. The average relative and proportional prediction accuracies for RF and NN models are 24.8% and 2.23, and, 23.9% and 2.48, respectively.

- If we compare the average prediction accuracies of LR-A4 with RF-MA and NN-MA, we do not see much difference in their relative prediction accuracies. However, average proportional errors show that LR-A4 is 1.57× and 1.74× better than RF-MA and NN-MA, respectively.

Discussion

Following are the salient observations from the experimental results:

- For our experiments, the training data set is constructed by executing the DGEMM and FFT with a constant increment of workload sizes using a fixed step size of 64. The generation of training data is a tedious task. However, once a model with the desired accuracy is constructed, it represents the relationship of a specific combination of PMCs and understands the underlying pattern with dynamic energy consumption. Therefore, the model can be used to predict the energy consumption for any workload size for an application or a set of applications.

- The statistical methods lack the ability to check the physical significance of the model variables with energy consumption. The PMCs used to build the models for using this approach contains memory parameters as model variables (for example, MEM_LOAD_RETIRED_L3_MISS in Table 5.12). The practical implications of the theory of energy predictive models for computing incorporate domain knowledge for linear dynamic energy predictive models by introducing properties for the selection of model variables, coefficients, and intercepts. In order to identify the processor components that are the dominating contributors to dynamic energy consumption during the application executions, we conducted experiments by stressing memory and CPU. We found that the dynamic energy consumption because of memory operations is negligible on our platforms. Therefore, linear regression models build using the theory

of energy predictive models for computing do not employ any memory PMCs as model variables.

- The models based on most additive and highly correlated PMCs have better average prediction accuracy when compared to the models based on non-additive and highly positively correlated PMCs. We conclude, therefore, that correlation with dynamic energy consumption alone is not sufficient to provide good average prediction accuracy but should be combined with methods such as *additivity* that take into account the physical significance of the model variables originating from the theory of energy predictive models for computing.

- Online LR models that employ PMCs selected using the theory of energy conservation of computing perform better in terms of average proportional accuracy than RF and NN based models that use purely statistical methods to select PMCs.

- While we do not see much difference in the relative prediction accuracies (p) of LR-A4, RF-MA, and NN-MA, average proportional errors show that LR-A4 is 1.57$\times$ and 1.74$\times$ better than RF-MA and NN-MA, respectively. This suggests that μ is a better statistic than p for accurate interpretation of results.

- The consistent accuracy of LR models highlight the importance of taking into account domain-specific knowledge for model variable selection, in this case, the physical significance of the PMCs originating from the conservation of energy of computing.

- The results also endorse the guidelines of the theory of energy predictive models for computing, which states that any non-linear energy model (in this case, the RF and NN models) employing PMCs only, will be inconsistent and hence inherently inaccurate. A non-linear energy model, in order to be accurate, must employ non-additive model variables in addition to PMCs.

5.5 Summary

Accurate and reliable measurement of energy consumption is essential to energy optimization at an application level. Energy predictive modelling using performance monitoring counters (PMCs) emerged as a promising approach, one of the main drivers being its ability to provide fine-grained component-level decomposition of energy consumption. Because of a limited number of hardware registers (3-4) dedicated to storing the PMCs, the selection of the best set of PMCs for energy predictive models is crucial.

In this chapter, we compared two types of energy predictive models constructed from the same set of experimental data and at two levels, platform and application. The first type contains linear regression (LR) models employing PMCs selected using a theoretical model of the energy of computing, which is the manifestation of the fundamental physical law of energy conservation. The second type contains sophisticated statistical learning models, random forest (RF), and neural network (NN), that are constructed using PMCs selected based on correlation and principal component analysis.

We demonstrated how the accuracy of LR models can be improved by selecting PMCs based on a theoretical model of the energy of computing. We compared the prediction accuracy of LR models with RF and NN models, which are based on PMCs selected using correlation and principal component analysis. We employed two different configurations of train and test datasets. In the first configuration, the train and test datasets contain all the applications. In the second configuration, the applications are split between the train and test datasets. We showed that the average prediction accuracy of the best LR model is almost the same in both the experimental configurations and its average prediction accuracy is better than RF and NN models. This highlights the consistent accuracy of LR models. The prediction accuracy of RF and NN models is better in the second configuration than the first configuration. It implies that RF and NN models memorize well the domain of inputs but are not able to predict well and hence generalize well for a new set. This is because they use PMC selection techniques that are domain oblivious and that do not take into account the physical significance of the PMCs originating from the

conservation of energy of computing.

We also studied application-specific energy predictive models. We experimentally demonstrated that the use of highly additive PMCs results in notable improvements in the average prediction accuracy of LR models when compared to LR models employing non-additive PMCs. We concluded, therefore, that a high positive correlation with dynamic energy consumption alone is not sufficient to provide good prediction accuracy but should be combined with selection criteria that take into account the physical significance of the PMCs originating from fundamental laws such as energy conservation of computing. Finally, we presented an experimental methodology to select a reliable subset of four PMCs for constructing accurate application-specific *online* models. We showed that the LR models perform better than RF and NN models.

Our results highlight two important points. First, the consistent accuracy of LR models stress the importance of taking into account domain-specific knowledge for model variable selection, in this case, the physical significance of the PMCs originating from the conservation of energy of computing. Second, according to the theory of energy predictive models for computing, any non-linear energy model (in this case, the RF and NN models) employing PMCs only, will be inconsistent and hence inherently inaccurate. A non-linear energy model, in order to be accurate, must employ non-additive model variables in addition to PMCs.

In our future work, we will explore methods to improve the prediction accuracy of energy predictive models that employ high-level model variables such as the utilization rates of compute devices unlike PMCs which are pure counts.

Chapter 6

Conclusion

Modern computing platforms have evolved with increased complexities such as high resource contention and non-uniform memory access. The energy of computing is a serious environmental concern and mitigating it has become an important technological challenge. Information and Communications Technology (ICT) devices and systems are presently consuming about 2000 terawatt-hours (TWh) per year which is about 10% of the global electricity demand. It has been predicted that computing systems and devices will consume up to 50% of global electricity in 2030 with a contribution to greenhouse gas emissions of 23%. Considering the unsustainable future predicted, energy efficiency in ICT is becoming a grand technological challenge and is now a first-class design constraint in all computing settings.

Energy optimization in computing is driven by innovations both at the system-level and application-level. System-level optimization methods aim to maximize the energy efficiency of the environment where the applications are executed using techniques such as DVFS (dynamic voltage and frequency scaling), Dynamic Power Management (DPM), and energy-aware scheduling. Application-level optimization methods use application-level parameters and models to maximize the energy efficiency of the applications. While the mainstream approach is to minimize the energy of the operating environment and is extensively researched, application-level energy optimization is comparatively understudied and forms the focus of this thesis. Accurate measurement

of energy consumption during an application execution is key to energy minimization techniques at the software level. There are three mainstream methods for energy measurement: (a) System-level physical measurements using external power meters, (b) Measurements using on-chip power sensors, and (c) Energy predictive models. In this thesis, we first present a comprehensive study on the accuracy of state-of-the-art energy measurement approaches on multicore CPUs (Chapter 3). We demonstrated poor energy consumption measurements from on-chip sensors when compared with the ground-truth system-level energy measurements using external power meters. We also showed that energy predictive models based on PMCs are plagued by poor accuracy. This thesis first explores the causes of the inaccuracy of linear energy predictive models.

We discovered that existing techniques for the selection of model variables in energy predictive models have not considered one fundamental property of predictor variables that should have been considered in the first place to remove PMCs unfit for modeling energy. We call it the property of *additivity* of PMCs. It is based on the experimental observation that the energy consumption of a serial execution of two applications is the sum of energy consumptions observed for the individual execution of each application. A linear energy predictive model is consistent if and only if its predictor variables are additive in the sense that the vector of predictor variables for a serial execution of two applications is the sum of vectors for the individual execution of each application. It must also have non-negative coefficients and zero intercept when modelling the dynamic energy consumption.

In Chapter 4, we first studied the *additivity* of PMCs offered by two popular tools, *Likwid* and *PAPI*, using a detailed statistical experimental methodology on a modern Intel Haswell multicore server CPU. We showed that many PMCs in Likwid and PAPI are *non-additive* and that some of these PMCs are key predictor variables in energy predictive models thereby bringing into question the reliability and reported prediction accuracy of these models. We showed that a PMC can be non-additive with error as high as 3075% and there are PMCs where the error is over 100%. We discovered that the number of non-additive PMCs rises with an increase in the number of cores employed in the applica-

tion. We consider this to be an inherent trait of modern multicore computing platforms because of severe resource contention and non-uniform memory access (NUMA). We then summarized and generalized the assumptions behind the existing work on PMC-based energy predictive modelling. We used a model-theoretic approach to formulate the assumed properties of the existing models in a mathematical form. We further extended the formalism by adding properties, heretofore unconsidered, that are basic implications of the universal energy conservation law. The new properties are intuitive and have been experimentally validated. The extended formalism defined our theory of *energy of computing*. Using the theory, we proved that a consistent energy predictive model is linear if and only if each PMC variable is *additive*. The implications are unified in a *consistency* test, which contains a suite of properties that include determinism, reproducibility, and additivity to select model variables and constraints for model coefficients.

We experimentally showed that contravening the requirements of the test can worsen the prediction accuracy of linear energy predictive models. We explored the construction of most accurate PMC-based application-specific models on our platform. We studied the prediction accuracy of platform-level and socket-level energy predictive models for data-parallel applications executing on a multi-socket multicore CPU platform where the sockets are independently powered. Our results demonstrated that socket-level models employing socket-level PMCs, which represent the resource utilization of individually powered components, yield a more accurate energy predictive model. We studied the optimization of a parallel scientific application for dynamic energy using IntelRAPL and power meters. We demonstrated that a significant amount of energy (up to 84% for applications used in the experiments) is lost by using IntelRAPL most likely because it does not take into account the properties of the theory of energy predictive models for computing.

Finally, In Chapter 5 we presented the first comprehensive study of techniques for energy predictive modelling using PMCs on modern multicore CPUs. We compared two types of energy predictive models constructed from the same set of experimental data and at two levels, platform, and application. The first type contains linear regression (LR) models employing PMCs se-

lected using a theoretical model of the energy of computing, which is the manifestation of the fundamental physical law of energy conservation. The second type contains sophisticated statistical learning models, random forest (RF), and neural network (NN), that are constructed using PMCs selected based on correlation and principal component analysis. We demonstrated how the accuracy of platform-level LR models can be improved by selecting PMCs based on a property of additivity. We discovered that the CPU-centric operations such as floating-point operations and core micro-operations are dominant contributors towards the dynamic energy consumption and that memory-centric operations make insignificant contributions towards dynamic energy consumption on our platforms. Therefore, we do not use memory PMCs to train the models. We showed that the average prediction accuracies of LR models are better than RF and NN models. We demonstrated that this is because RF and NN models use PMC selection techniques that are domain oblivious and that do not take into account the physical significance of the PMCs originating from the conservation of energy of computing. We concluded from the study of application-specific energy predictive models that a high positive correlation with dynamic energy consumption alone is not sufficient to provide good prediction accuracy but should be combined with selection criteria that take into account the physical significance of the PMCs originating from fundamental laws such as energy conservation of computing.

The potential future work, which could be relevant in the extension of this thesis, includes:

1. Using the theory of energy predictive models for computing, develop and analyze the energy predictive models for a range of CPU architectures such as AMD, ARM, and MIPS, etc.

2. Using the theory of energy predictive models for computing, develop and analyze the energy predictive models for accelerators such as GPUs, Xeon Phis, and FPGAs, etc.

3. Developing and mathematically formulating a theory of energy predictive models for computing for non-linear energy predictive models.

4. Extending the methodology to select the model variables for energy predictions using high-level metrics.

5. Exploring the methods to improve the prediction accuracy of energy predictive models that employ high-level model variables based on the utilization rates of computing devices unlike PMCs, that are pure counts.

6. Studying the accuracy of platform-level and application-specific energy predictive models using pure CPU and memory utilization as model variables, high-level measurements from on-chip power sensors, high-level operating systems, and application parameters such as the number of instructions and problem sizes, etc.

7. Extension *SLOPE-PMC* to provide support for automated collection of PMCs for GPUs and other accelerators.

8. Extension of techniques to accurately and reliably build energy predictive models for workload-parallel applications executing on a multi-socket computing platform.

In closing, our theory of energy predictive models for computing has demonstrated the importance of using the physical significance of model variables for their employment in energy consumption models to achieve accuracy and reliability. We have demonstrated the potential of this novel approach; the challenge now is to ensure that this potential is realised.

Bibliography

[1] A. Andrae and T. Edler, “On global electricity usage of communication technology: Trends to 2030,” *Challenges*, vol. 6, p. 117–157, Apr 2015.

[2] G. E. Moore, “Cramming more components onto integrated circuits, reprinted from electronics, volume 38, number 8, april 19, 1965, pp.114 ff.,” *IEEE Solid-State Circuits Society Newsletter*, vol. 11, no. 3, pp. 33–35, 2006.

[3] R. H. Dennard, F. H. Gaensslen, H. Yu, V. L. Rideout, E. Bassous, and A. R. LeBlanc, “Design of ion-implanted mosfet’s with very small physical dimensions,” *IEEE Journal of Solid-State Circuits*, vol. 9, no. 5, pp. 256–268, 1974.

[4] M. Cierniak, M. J. Zaki, and W. Li, “Compile-time scheduling algorithms for a heterogeneous network of workstations,” *The Computer Journal*, vol. 40, no. 6, pp. 356–372, 1997.

[5] O. Beaumont, V. Boudet, F. Rastello, and Y. Robert, “Matrix multiplication on heterogeneous platforms,” *Parallel and Distributed Systems, IEEE Transactions on*, vol. 12, no. 10, pp. 1033–1051, 2001.

[6] A. Kalinov and A. Lastovetsky, “Heterogeneous distribution of computations solving linear algebra problems on networks of heterogeneous computers,” *J. Parallel Distrib. Comput.*, vol. 61, Apr. 2001.

[7] A. L. Lastovetsky and R. Reddy, “Data partitioning with a realistic performance model of networks of heterogeneous computers,” in *Parallel*

and Distributed Processing Symposium, 2004. Proceedings. 18th International, p. 104, IEEE, 2004.

[8] A. Lastovetsky and R. Reddy, "Data partitioning with a functional performance model of heterogeneous processors," *International Journal of High Performance Computing Applications*, vol. 21, no. 1, pp. 76–90, 2007.

[9] A. Lastovetsky, L. Szustak, and R. Wyrzykowski, "Model-based optimization of EULAG kernel on Intel Xeon Phi through load imbalancing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, pp. 787–797, March 2017.

[10] A. Lastovetsky and R. Reddy, "New model-based methods and algorithms for performance and energy optimization of data parallel applications on homogeneous multicore clusters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 4, pp. 1119–1133, 2017.

[11] OpenBLAS, "OpenBLAS: An optimized BLAS library," 2016.

[12] FFTW, "FFTW: A fast, free c FFT library," 2016.

[13] A. Lastovetsky and R. Reddy, "Data distribution for dense factorization on computers with memory heterogeneity," *Parallel Computing*, vol. 33, Dec. 2007.

[14] A. Ilić, F. Pratas, P. Trancoso, and L. Sousa, "High-performance computing on heterogeneous systems: Database queries on CPU and GPU," *High Performance Scientific Computing with Special Emphasis on Current Capabilities and Future Perspectives*, pp. 202–222, 2010.

[15] D. Clarke, A. Lastovetsky, and V. Rychkov, "Dynamic load balancing of parallel computational iterative routines on highly heterogeneous HPC platforms," *Parallel Processing Letters*, vol. 21, no. 02, pp. 195–217, 2011.

[16] D. Clarke, A. L. Lastovetsky, and V. Rychkov, "Column-based matrix partitioning for parallel matrix multiplication on heterogeneous processors based on functional performance models," in *Euro-Par 2011: Parallel Processing Workshops*, vol. 7155 of *Lecture Notes in Computer Science*, Springer-Verlag, 2012.

[17] X. Liu, Z. Zhong, and K. Xu, "A hybrid solution method for CFD applications on GPU-accelerated hybrid HPC platforms," *Future Generation Computer Systems*, vol. 56, pp. 759–765, 2016.

[18] M. Radmanović, D. Gajić, and R. Stanković, "Efficient computation of galois field expressions on hybrid CPU-GPU platforms.," *Journal of Multiple-Valued Logic & Soft Computing*, vol. 26, 2016.

[19] A. Ilic and L. Sousa, "Simultaneous multi-level divisible load balancing for heterogeneous desktop systems," in *Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th International Symposium on*, pp. 683–690, IEEE, 2012.

[20] J. Colaço, A. Matoga, A. Ilic, N. Roma, P. Tomás, and R. Chaves, "Transparent application acceleration by intelligent scheduling of shared library calls on heterogeneous systems," in *Parallel Processing and Applied Mathematics*, pp. 693–703, Springer, 2013.

[21] V. Cardellini, A. Fanfarillo, and S. Filippone, "Heterogeneous sparse matrix computations on hybrid GPU/CPU platforms.," in *PARCO*, pp. 203–212, 2013.

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

[23] N. Jones, "How to stop data centres from gobbling up the world's electricity," *Nature*, vol. 561, pp. 163–166, 2018.

[24] L. A. Barroso and U. Hölzle, "The case for energy-proportional computing," *Computer*, no. 12, pp. 33–37, 2007.

[25] DOE, “The opportunities and challenges of exascale computing,” 2010.

[26] A. Chakrabarti, S. Parthasarathy, and C. Stewart, “A pareto framework for data analytics on heterogeneous systems: Implications for green energy usage and performance,” in *Parallel Processing (ICPP), 2017 46th International Conference on*, pp. 533–542, IEEE, 2017.

[27] R. Reddy and A. Lastovetsky, “Bi-objective optimization of data-parallel applications on homogeneous multicore clusters for performance and energy,” *IEEE Transactions on Computers*, vol. 64, no. 2, pp. 160–177, 2018.

[28] R. Reddy Manumachu and A. L. Lastovetsky, “Design of self-adaptable data parallel applications on multicore clusters automatically optimized for performance and energy through load distribution,” *Concurrency and Computation: Practice and Experience*, vol. 31, no. 4, p. e4958, 2019.

[29] H. Khaleghzadeh, M. Fahad, A. Shahid, R. Reddy, and A. Lastovetsky, “Bi-objective optimization of data-parallel applications on heterogeneous HPC platforms for performance and energy through workload distribution,” *CoRR*, vol. abs/1907.04080, 2019.

[30] E. Rotem, A. Naveh, A. Ananthakrishnan, E. Weissmann, and D. Rajwan, “Power-Management architecture of the intel microarchitecture Code-Named sandy bridge,” *IEEE Micro*, vol. 32, pp. 20–27, March 2012.

[31] Nvidia, “Nvidia management library: NVML reference manual,” 10 2018.

[32] I. Corporation, “Intel xeon phi coprocessor system software developers guide,” 06 2014.

[33] C. Gough, I. Steiner, and W. Saunders, *Energy Efficient Servers Blueprints for Data Center Optimization*. Apress, 2015.

[34] K. N. Khan, M. Hirki, T. Niemi, J. K. Nurminen, and Z. Ou, “Rapl in action: Experiences in using rapl for power measurements,” *ACM Trans. Model. Perform. Eval. Comput. Syst.*, vol. 3, pp. 9:1–9:26, Mar. 2018.

[35] D. Hackenberg, T. Ilsche, R. Schöne, D. Molka, M. Schmidt, and W. E. Nagel, "Power measurement techniques on standard compute nodes: A quantitative comparison," in *Performance analysis of systems and software (ISPASS), 2013 IEEE international symposium on*, pp. 194–204, IEEE, 2013.

[36] M. Fahad, A. Shahid, R. R. Manumachu, and A. Lastovetsky, "A comparative study of methods for measurement of energy of computing," *Energies*, vol. 12, no. 11, 2019.

[37] PAPI, "Performance application programming interface 5.4.1," 2015.

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

[39] Likwid, "Architecture specific notes for Intel Haswell," 2017.

[40] K. O'Brien, I. Pietri, R. Reddy, A. Lastovetsky, and R. Sakellariou, "A survey of power and energy predictive models in HPC systems and applications," *ACM Computing Surveys*, vol. 50, no. 3, 2017.

[41] W. L. Bircher and L. K. John, "Complete system power estimation using processor performance events," *IEEE Transactions on Computers*, vol. 61, pp. 563–577, Apr. 2012.

[42] P. Gschwandtner, M. Knobloch, B. Mohr, D. Pleiter, and T. Fahringer, "Modeling CPU energy consumption of hpc applications on the IBM POWER7," in *Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on*, pp. 536–543, IEEE, 2014.

[43] J. Haj-Yihia, A. Yasin, Y. B. Asher, and A. Mendelson, "Fine-grain power breakdown of modern out-of-order cores and its implications on skylake-based systems," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 13, no. 4, p. 56, 2016.

[44] K. Singh, M. Bhadauria, and S. A. McKee, "Real time power estimation and thread scheduling via performance counters," *ACM SIGARCH Computer Architecture News*, vol. 37, no. 2, pp. 46–55, 2009.

[45] B. Goel, S. A. McKee, R. Gioiosa, K. Singh, M. Bhadauria, and M. Cesati, "Portable scalable per-core power estimation for intelligent resource management," in *Portable, Scalable, per-Core Power Estimation for Intelligent Resource Management*, Green Computing Conference, 2010 International, 2010-08-16 2010.

[46] C. Lively, X. Wu, V. Taylor, S. Moore, H.-C. Chang, C.-Y. Su, and K. Cameron, "Power-aware predictive models of hybrid (mpi/openmp) scientific applications on multicore systems," *Computer Science-Research and Development*, vol. 27, no. 4, pp. 245–253, 2012.

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

[48] M. Witkowski, A. Oleksiak, T. Piontek, and J. Weglarz, "Practical power consumption estimation for real life HPC applications," *Future Gener. Comput. Syst.*, vol. 29, Jan. 2013.

[49] M. Jarus, A. Oleksiak, T. Piontek, and J. Węglarz, "Runtime power usage estimation of HPC servers for various classes of real-life applications," *Future Generation Computer Systems*, vol. 36, 2014.

[50] X. Wu, V. Taylor, J. Cook, and P. J. Mucci, "Using Performance-Power modeling to improve energy efficiency of HPC applications," *Computer*, vol. 49, no. 10, pp. 20–29, 2016.

[51] M. Chadha, T. Ilsche, M. Bielert, and W. E. Nagel, "A statistical approach to power estimation for x86 processors," in *Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2017 IEEE International*, pp. 1012–1019, IEEE, 2017.

[52] D. Economou, S. Rivoire, C. Kozyrakis, and P. Ranganathan, "Full-system power analysis and modeling for server environments," in *In Proceedings of Workshop on Modeling, Benchmarking, and Simulation*, pp. 70–77, 2006.

[53] J. C. McCullough, Y. Agarwal, J. Chandrashekar, S. Kuppuswamy, A. C. Snoeren, and R. K. Gupta, "Evaluating the effectiveness of model-based power characterization," in *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'11, USENIX Association, 2011.

[54] A. Shahid, M. Fahad, R. Reddy, and A. Lastovetsky, "Additivity: A selection criterion for performance events for reliable energy predictive modeling," *Supercomput. Front. Innov.: Int. J.*, vol. 4, pp. 50–65, Dec. 2017.

[55] N. Muralimanohar, R. Balasubramonian, and N. Jouppi, "Optimizing NUCA organizations and wiring alternatives for large caches with CACTI 6.0," in *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 40, IEEE Computer Society, 2007.

[56] N. Vijaykrishnan, M. Kandemir, M. J. Irwin, H. S. Kim, and W. Ye, "Energy-driven integrated hardware-software optimizations using SimplePower," in *Proceedings of the 27th Annual International Symposium on Computer Architecture*, ISCA '00, ACM, 2000.

[57] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A framework for architectural-level power analysis and optimizations," in *Proceedings of the 27th Annual International Symposium on Computer Architecture*, ISCA '00, ACM, 2000.

[58] D. Brooks, M. Martonosi, J.-D. Wellman, and P. Bose, "Power-performance modeling and tradeoff analysis for a high end microprocessor," in *Proceedings of the First International Workshop on Power-Aware Computer Systems-Revised Papers*, PACS '00, Springer-Verlag, 2001.

[59] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "The McPAT framework for multicore and manycore architectures: Simultaneously modeling power, area, and timing," *ACM Trans. Archit. Code Optim.*, vol. 10, no. 1, 2013.

[60] S. L. Xi, H. Jacobson, P. Bose, G.-Y. Wei, and D. Brooks, "Quantifying sources of error in McPAT and potential impacts on architectural studies," in *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, pp. 577–589, IEEE, 2015.

[61] M. Fahad, A. Shahid, R. R. Manumachu, and A. Lastovetsky, "Accurate energy modelling of hybrid parallel applications on modern heterogeneous computing platforms using system-level measurements," *IEEE Access*, vol. 8, pp. 93793–93829, 2020.

[62] F. Almeida, M. D. Assuncao, J. Barbosa, V. Blanco, I. Brandic, G. Da Costa, M. F. Dolz, A. C. Elster, M. Jarus, H. D. Karatza, *et al.*, "Energy monitoring as an essential building block towards sustainable ultrascale systems," *Sustainable Computing: Informatics and Systems*, vol. 17, pp. 27–42, 2018.

[63] M. Burtscher, I. Zecena, and Z. Zong, "Measuring gpu power with the k20 built-in sensor," in *Proceedings of Workshop on General Purpose Processing Using GPUs*, GPGPU-7, (New York, NY, USA), pp. 28:28–28:36, ACM, 2014.

[64] D. Hackenberg, R. Schöne, T. Ilsche, D. Molka, J. Schuchart, and R. Geyer, "An energy efficiency feature survey of the intel haswell processor," in *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*, pp. 896–904, May 2015.

[65] A. Cabrera, F. Almeida, and V. Blanco, "Eml, an energy measurement library," in *IFIP WG 7.3 Performance 2013 31 st International Symposium on Computer Performance, Modeling, Measurements and Evaluation 2013 Student Poster Abstracts September 24-26, Vienna, Austria*, p. 5, 2013.

[66] A. Cabrera, F. Almeida, J. Arteaga, and V. Blanco, "Energy measurement library (eml) usage and overhead analysis," in *2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pp. 554–558, IEEE, 2015.

[67] I. Corporation, "Intel® manycore platform software stack (Intel MPSS)," 06 2014.

[68] A. M. Devices, "Bios and kernel developer's guide (bkdg) for amd family 15h models 00h-0fh processors," 2012.

[69] S. Roy, A. Rudra, and A. Verma, "An energy complexity model for algorithms," in *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science*, ITCS '13, (New York, NY, USA), pp. 283–304, ACM, 2013.

[70] A. Lewis, S. Ghosh, and N.-F. Tzeng, "Run-time energy consumption estimation based on workload in server systems," in *Proceedings of the 2008 Conference on Power Aware Computing and Systems*, HotPower'08, (Berkeley, CA, USA), pp. 4–4, USENIX Association, 2008.

[71] R. Basmadjian, N. Ali, F. Niedermeier, H. de Meer, and G. Giuliani, "A methodology to predict the power consumption of servers in data centres," in *Proceedings of the 2Nd International Conference on Energy-Efficient Computing and Networking*, e-Energy '11, (New York, NY, USA), pp. 1–10, ACM, 2011.

[72] T. Heath, B. Diniz, B. Horizonte, E. V. Carrera, and R. Bianchini, "Energy conservation in heterogeneous server clusters," in *10th ACM SIGPLAN symposium on Principles and practice of parallel programming (PPoPP)*, pp. 186–195, ACM, 2005.

[73] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *34th Annual International Symposium on Computer architecture*, pp. 13–23, ACM, 2007.

[74] A. Kansal and F. Zhao, "Fine-grained energy profiling for power-aware application design," *ACM SIGMETRICS Performance Evaluation Review*, vol. 36, p. 26, Aug. 2008.

[75] X. Feng, R. Ge, and K. W. Cameron, "Power and energy profiling of scientific applications on distributed systems," in *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, pp. 34–34, IEEE, 2005.

[76] T. Li and L. K. John, "Run-time modeling and estimation of operating system power consumption," *SIGMETRICS Perform. Eval. Rev.*, vol. 31, pp. 160–171, June 2003.

[77] S. Rivoire, P. Ranganathan, and C. Kozyrakis, "A comparison of high-level full-system power models," in *Proceedings of the 2008 Conference on Power Aware Computing and Systems*, HotPower'08, USENIX Association, 2008.

[78] S. Rivoire, "Models and metrics for energy-efficient computer systems.," in *PhD Thesis*, Stanford University, Stanford, California, 2008.

[79] H. Wang, Q. Jing, R. Chen, B. He, Z. Qian, and L. Zhou, "Distributed systems meet economics: pricing in the cloud," in *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, USENIX Association, 2010.

[80] W. Dargie, "A stochastic model for estimating the power consumption of a processor," *IEEE Transactions on Computers*, vol. 64, no. 5, 2015.

[81] G. Jung, M. A. Hiltunen, K. R. Joshi, R. D. Schlichting, and C. Pu, "Mistral: Dynamically managing power, performance, and adaptation cost in cloud infrastructures," in *Distributed Computing Systems (ICDCS), 2010 IEEE 30th International Conference on*, pp. 62–73, IEEE, 2010.

[82] CUPTI, "CUDA profiling tools interface," 2017.

[83] IntelPCM, "Intel performance counter monitor - a better way to measure CPU utilization.," 2012.

[84] P. Wiki, “perf: Linux profiling with performance counters,” 2017.

[85] I. Kadayif, T. Chinoda, M. Kandemir, N. Vijaykirsnan, M. J. Irwin, and A. Sivasubramaniam, “vec: virtual energy counters,” in *Proceedings of the 2001 ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*, pp. 28–31, 2001.

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

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

[88] B. C. Lee and D. M. Brooks, “Accurate and efficient regression modeling for microarchitectural performance and power prediction,” *SIGARCH Comput. Archit. News*, vol. 34, pp. 185–194, Oct. 2006.

[89] M. D. Powell, A. Biswas, J. S. Emer, S. S. Mukherjee, B. R. Sheikh, and S. Yardi, “CAMP: A technique to estimate per-structure power at runtime using a few simple parameters,” in *2009 IEEE 15th International Symposium on High Performance Computer Architecture*, pp. 289–300, Feb 2009.

[90] R. Bertran, M. Gonzalez, X. Martorell, N. Navarro, and E. Ayguade, “Decomposable and responsive power models for multicore processors using performance counters,” in *Proceedings of the 24th ACM International Conference on Supercomputing*, pp. 147–158, ACM, 2010.

[91] H. Hong, Sunpyand Kim, “An integrated GPU power and performance model,” *SIGARCH Comput. Archit. News*, vol. 38, no. 3, 2010.

[92] H. Nagasaka, N. Maruyama, A. Nukada, T. Endo, and S. Matsuoka, “Statistical power modeling of GPU kernels using performance coun-

ters," in *International Green Computing Conference and Workshops (IGCC)*, IEEE, 2010.

[93] Y. S. Shao and D. Brooks, "Energy characterization and instruction-level energy model of Intel's Xeon Phi processor," in *Proceedings of the 2013 International Symposium on Low Power Electronics and Design*, ISLPED '13, IEEE Press, 2013.

[94] Z. Al-Khatib and S. Abdi, "Operand-value-based modeling of dynamic energy consumption of soft processors in FPGA," in *International Symposium on Applied Reconfigurable Computing*, pp. 65–76, Springer, 2015.

[95] V. Bui, B. Norris, K. Huck, L. C. McInnes, L. Li, O. Hernandez, and B. Chapman, "A component infrastructure for performance and power modeling of parallel scientific applications," in *Proceedings of the 2008 compFrame/HPC-GECO Workshop on Component Based High Performance*, CBHPC '08, pp. 6:1–6:11, ACM, 2008.

[96] J. Dongarra, H. Ltaief, P. Luszczek, and V. Weaver, "Energy footprint of advanced dense numerical linear algebra using tile algorithms on multicore architecture," in *The 2nd International Conference on Cloud and Green Computing*, November 2012.

[97] A. Tiwari, M. A. Laurenzano, L. Carrington, and A. Snavely, "Modeling power and energy usage of HPC kernels," in *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International*, pp. 990–998, IEEE, 2012.

[98] A. Cabrera, F. Almeida, V. Blanco, and D. Gimenez, "Analytical modeling of the energy consumption for the high performance linpack," in *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pp. 343–350, IEEE, 2013.

[99] C. Mobius, W. Dargie, and A. Schill, "Power consumption estimation models for processors, virtual machines, and servers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 6, 2014.

[100] M. Dayarathna, Y. Wen, and R. Fan, "Data center energy consumption modeling: A survey," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 732–794, 2016.

[101] R. A. Bridges, N. Imam, and T. M. Mintz, "Understanding gpu power: A survey of profiling, modeling, and simulation methods," *ACM Comput. Surv.*, vol. 49, no. 3, 2016.

[102] F. Almeida, J. Arteaga, V. Blanco, and A. Cabrera, "Energy measurement tools for ultrascale computing: A survey," *Supercomputing frontiers and innovations*, vol. 2, no. 2, pp. 64–76, 2015.

[103] Q. Deng, D. Meisner, A. Bhattacharjee, T. F. Wenisch, and R. Bianchini, "Coscale: Coordinating cpu and memory system dvfs in server systems," in *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 143–154, IEEE Computer Society, 2012.

[104] Z. Lai, K. T. Lam, C.-L. Wang, J. Su, Y. Yan, and W. Zhu, "Latency-aware dynamic voltage and frequency scaling on many-core architectures for data-intensive applications," in *2013 International Conference on Cloud Computing and Big Data*, pp. 78–83, IEEE, 2013.

[105] G. Chen, K. Huang, and A. Knoll, "Energy optimization for real-time multiprocessor system-on-chip with optimal dvfs and dpm combination," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 13, no. 3s, p. 111, 2014.

[106] A. K. Datta and R. Patel, "Cpu scheduling for power/energy management on multicore processors using cache miss and context switch data," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 5, pp. 1190–1199, 2013.

[107] N. B. Rizvandi, J. Taheri, and A. Y. Zomaya, "Some observations on optimal frequency selection in dvfs-based energy consumption minimization," *Journal of Parallel and Distributed Computing*, vol. 71, no. 8, pp. 1154–1164, 2011.

[108] S. Yang, R. A. Shafik, G. V. Merrett, E. Stott, J. M. Levine, J. Davis, and B. M. Al-Hashimi, "Adaptive energy minimization of embedded heterogeneous systems using regression-based learning," in *2015 25th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, pp. 103–110, IEEE, 2015.

[109] F. P. Miller, A. F. Vandome, and J. McBrewster, "Advanced configuration and power interface: Open standard, operating system, power management, cross-platform, intel corporation, microsoft, toshiba,... sleep mode, hibernate (os feature), synonym," 2009.

[110] W. L. Bircher and L. K. John, "Analysis of dynamic power management on multi-core processors," in *Proceedings of the 22nd annual international conference on Supercomputing*, pp. 327–338, ACM, 2008.

[111] K. Huang, L. Santinelli, J.-J. Chen, L. Thiele, and G. C. Buttazzo, "Adaptive power management for real-time event streams," in *Proceedings of the 2010 Asia and South Pacific Design Automation Conference*, pp. 7–12, IEEE Press, 2010.

[112] E.-Y. Chung, L. Benini, and G. De Micheli, "Dynamic power management using adaptive learning tree," in *Proceedings of the 1999 IEEE/ACM international conference on Computer-aided design*, pp. 274–279, IEEE Press, 1999.

[113] A. Beloglazov and R. Buyya, "Energy efficient resource management in virtualized cloud data centers," in *Proceedings of the 2010 10th IEEE/ACM international conference on cluster, cloud and grid computing*, pp. 826–831, IEEE Computer Society, 2010.

[114] W.-K. Lee, S.-W. Lee, and W.-O. Siew, "Hybrid model for dynamic power management," *IEEE Transactions on Consumer Electronics*, vol. 55, no. 2, pp. 656–664, 2009.

[115] L. Niu and G. Quan, "Reducing both dynamic and leakage energy consumption for hard real-time systems," in *Proceedings of the 2004 inter-*

national conference on Compilers, architecture, and synthesis for embedded systems, pp. 140–148, ACM, 2004.

[116] J. Trajkovic, A. V. Veidenbaum, and A. Kejariwal, "Improving sdram access energy efficiency for low-power embedded systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 7, no. 3, p. 24, 2008.

[117] S. Song, C.-Y. Su, R. Ge, A. Vishnu, and K. W. Cameron, "Iso-energy-efficiency: An approach to power-constrained parallel computation," in *2011 IEEE International Parallel & Distributed Processing Symposium*, pp. 128–139, IEEE, 2011.

[118] J. H. Ahn, J. Leverich, R. Schreiber, and N. P. Jouppi, "Multicore dimm: An energy efficient memory module with independently controlled drams," *IEEE Computer Architecture Letters*, vol. 8, no. 1, pp. 5–8, 2008.

[119] A. R. Lebeck, X. Fan, H. Zeng, and C. Ellis, "Power aware page allocation," *ACM Sigplan Notices*, vol. 35, no. 11, pp. 105–116, 2000.

[120] Q. Deng, D. Meisner, L. Ramos, T. F. Wenisch, and R. Bianchini, "Memscale: active low-power modes for main memory," in *ACM SIGARCH Computer Architecture News*, vol. 39, pp. 225–238, ACM, 2011.

[121] J. Lin, H. Zheng, Z. Zhu, E. Gorbatov, H. David, and Z. Zhang, "Software thermal management of dram memory for multicore systems," *ACM SIGMETRICS Performance Evaluation Review*, vol. 36, no. 1, pp. 337–348, 2008.

[122] H. David, C. Fallin, E. Gorbatov, U. R. Hanebutte, and O. Mutlu, "Memory power management via dynamic voltage/frequency scaling," in *Proceedings of the 8th ACM international conference on Autonomic computing*, pp. 31–40, ACM, 2011.

[123] J. Demmel, A. Gearhart, B. Lipshitz, and O. Schwartz, "Perfect strong scaling using no additional energy," in *2013 IEEE 27th International*

Symposium on Parallel and Distributed Processing, pp. 649–660, IEEE, 2013.

[124] A. Cabrera, A. Acosta, F. Almeida, and V. Blanco, "A heuristic technique to improve energy efficiency with dynamic load balancing," *The Journal of Supercomputing*, vol. 75, no. 3, pp. 1610–1624, 2019.

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

[126] F. Alessi, P. Thoman, G. Georgakoudis, T. Fahringer, and D. S. Nikolopoulos, "Application-level energy awareness for openmp," in *International Workshop on OpenMP*, pp. 219–232, Springer, 2015.

[127] V. R. Silva, A. Furtunato, K. Georgiou, K. Eder, and S. Xavier-de Souza, "Energy-optimal configurations for single-node hpc applications," *arXiv preprint arXiv:1805.00998*, 2018.

[128] H. Wang, V. Sathish, R. Singh, M. J. Schulte, and N. S. Kim, "Workload and power budget partitioning for single-chip heterogeneous processors," in *Proceedings of the 21st international conference on Parallel architectures and compilation techniques*, pp. 401–410, ACM, 2012.

[129] A. L. Lastovetsky, L. Szustak, and R. Wyrzykowski, "Model-based optimization of MPDATA on Intel Xeon Phi through load imbalancing," *CoRR*, vol. abs/1507.01265, 2015.

[130] A. Lastovetsky and R. Reddy, "New model-based methods and algorithms for performance and energy optimization of data parallel applications on homogeneous multicore clusters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 4, pp. 1119–1133, 2017.

[131] H. Khaleghzadeh, R. R. Manumachu, and A. Lastovetsky, "A novel data-partitioning algorithm for performance optimization of data-parallel applications on heterogeneous HPC platforms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, pp. 2176–2190, Oct 2018.

[132] HCL, "HCLWattsUp: API for power and energy measurements using WattsUp Pro Meter `http://csgitlab.ucd.ie/hcl/hclwattsup`," 2016.

[133] J. Mair, Z. Huang, and D. Eyers, "Manila: Using a densely populated pmc-space for power modelling within large-scale systems," *Parallel Computing*, vol. 82, pp. 37–56, 2019.

[134] Z. Zhou, J. H. Abawajy, F. Li, Z. Hu, M. U. Chowdhury, A. Alelaiwi, and K. Li, "Fine-grained energy consumption model of servers based on task characteristics in cloud data center," *IEEE access*, vol. 6, pp. 27080–27090, 2018.

[135] A. Shahid, M. Fahad, R. Reddy Manumachu, and A. Lastovetsky, "Energy of computing on multicore cpus: Predictive models and energy conservation law," *arXiv*, 2019.

[136] Intel Optimzed HPCG, "Overview of the intel optimized hpcg `https://software.intel.com/en-us/node/599524`."

[137] A. Waterland, "Stress `https://people.seas.harvard.edu/{~}apw/stress/`," 2001.

[138] M. F. Dolz, J. Kunkel, K. Chasapis, and S. Catalán, "An analytical methodology to derive power models based on hardware and software metrics," *Computer Science-Research and Development*, vol. 31, no. 4, pp. 165–174, 2016.

[139] M. F. Dolz Zaragozá, J. Kunkel, K. Chasapis, and S. Catalán Pallarés, "An analytical methodology to derive power models based on hardware and software metrics," *Computer Science-Research and Development*, 2015.

[140] S. Wang, *Software power analysis and optimization for power-aware multicore systems*. Wayne State University, 2014.

[141] S. J. Eidenbenz, H. N. Djidjev, B. T. Nadiga, and E. J. Park, "Simulation-based and analytical models for energy use prediction," tech. rep., Los Alamos National Laboratory (LANL), 2016.

[142] D. Dauwe, R. Friese, S. Pasricha, A. A. Maciejewski, G. A. Koenig, and H. J. Siegel, "Modeling the effects on power and performance from memory interference of co-located applications in multicore systems," in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, p. 1, The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2014.

[143] X. Wu, H.-C. Chang, S. Moore, V. Taylor, C.-Y. Su, D. Terpstra, C. Lively, K. Cameron, and C. W. Lee, "Mummi: multiple metrics modeling infrastructure for exploring performance and power modeling," in *Proceedings of the Conference on Extreme Science and Engineering Discovery Environment: Gateway to Discovery*, p. 36, ACM, 2013.

[144] X. Wu, C. Lively, V. Taylor, H.-C. Chang, C.-Y. Su, K. Cameron, S. Moore, D. Terpstra, and V. Weaver, "Mummi: multiple metrics modeling infrastructure," in *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2013 14th ACIS International Conference on*, pp. 289–295, IEEE, 2013.

[145] K. Singh, M. Bhadauria, and S. A. McKee, "Real time power estimation and thread scheduling via performance counters," *SIGARCH Comput. Archit. News*, vol. 37, pp. 46–55, July 2009.

[146] A. Shahid, M. Fahad, R. R. Manumachu, and A. Lastovetsky, "Improving the accuracy of energy predictive models for multicore CPUs using additivity of performance monitoring counters," in *Parallel Computing Technologies* (V. Malyshkin, ed.), (Cham), pp. 51–66, Springer International Publishing, 2019.

[147] A. Shahid, M. Fahad, R. Reddy Manumachu, and A. Lastovetsky, "A comparative study of techniques for energy predictive modelling using

performance monitoring counters on modern multicore cpus," *IEEE Access*, 2020.

[148] J.-A. Rico-Gallego, J.-C. Díaz-Martín, and A. L. Lastovetsky, "Extending τ-lop to model concurrent mpi communications in multicore clusters," *Future Generation Computer Systems*, vol. 61, pp. 66–82, 2016.

[149] H. Abdi and L. J. Williams, "Principal component analysis," *Wiley interdisciplinary reviews: computational statistics*, vol. 2, no. 4, pp. 433–459, 2010.

[150] A. Liaw, M. Wiener, *et al.*, "Classification and regression by randomforest," *R news*, vol. 2, no. 3, pp. 18–22, 2002.

[151] F. D. Foresee and M. T. Hagan, "Gauss-Newton approximation to Bayesian learning," in *Proceedings of the 1997 international joint conference on neural networks*, vol. 3, pp. 1930–1935, Piscataway: IEEE, 1997.

[152] J. J. Moré, "The levenberg-marquardt algorithm: implementation and theory," in *Numerical analysis*, pp. 105–116, Springer, 1978.

[153] HCL, "SLOPE-PMC: Towards the automation of pmcs collection for intel based multicore platforms `https://csgitlab.ucd.ie/hcl/SLOPE/tree/master/SLOPE-PMC`," 2017.

[154] HCL, "DE-Meter: Calculate dynamic energy consumption using rapl meter `https://github.com/ArsalanShahid116/DE-METER`," 2019.

[155] A. Lastovetsky, M. Fahad, H. Khaleghzadeh, S. Khokhriakov, R. Reddy, A. Shahid, L. Szustak, and R. Wyrzykowski, "How pre-multicore methods and algorithms perform in multicore era," in *International Conference on High Performance Computing*, pp. 527–539, Springer, 2018.

Appendix A

Methodology for Reliable Energy Measurements

A.1 Rationale Behind Using Dynamic Energy Consumption Instead of Total Energy Consumption

We consider only the dynamic energy consumption in our work for reasons below:

1. Static energy consumption is a constant (or an inherent property) of a platform that can not be optimized. It does not depend on the application configuration.

2. Although the static energy consumption is a major concern in embedded systems, it is becoming less compared to the dynamic energy consumption due to advancements in hardware architecture design in HPC systems.

3. We target applications and platforms where dynamic energy consumption is the dominating energy dissipator.

4. Finally, we believe its inclusion can underestimate the true worth of an

optimization technique that minimizes the dynamic energy consumption. We elucidate using two examples from published results.

- In our first example, consider a model that reports predicted and measured the total energy consumption of a system to be 16,500 J and 18,000 J. It would report the prediction error to be 8.3%. If it is known that the static energy consumption of the system is 9000 J, then the actual prediction error (based on dynamic energy consumption only) would be 16.6% instead.
- In our second example, consider two different energy prediction models (M_A and M_B) with the same prediction errors of 5% for application execution on two different machines (A and B) with same total energy consumption of 10,000 J. One would consider both the models to be equally accurate. But supposing it is known that the dynamic energy proportions for the machines are 30% and 60%. Now, the true prediction errors (using dynamic energy consumption only) for the models would be 16.6% and 8.3%. Therefore, the second model M_B should be considered more accurate than the first.

A.2 Application Programming Interface (API) for Measurements Using External Power Meter Interfaces (HCLWattsUp)

HCLServer1, HCLServer2, and HCLServer03 have a dedicated power meter installed between their input power sockets and wall A/C outlets. The power meter captures the total power consumption of the node. It has a data cable connected to the USB port of the node. A Perl script collects the data from the power meter using the serial USB interface. The execution of this script is non-intrusive and consumes insignificant power.

We use *HCLWattsUp* API function, which gathers the readings from the power meters to determine the average power and energy consumption dur-

ing the execution of an application on a given platform. *HCLWattsUp* API can provide the following four types of measures during the execution of an application:

- *TIME*—The execution time (seconds).
- *DPOWER*—The average dynamic power (watts).
- *TENERGY*—The total energy consumption (joules).
- *DENERGY*— dynamic energy consumption (joules).

We confirm that the overhead due to the API is very minimal and does not have any noticeable influence on the main measurements. It is important to note that the power meter readings are only processed if the measure is not *hcl::TIME*. Therefore, for each measurement, we have two runs. One run for measuring the execution time. And the other for energy consumption. The following example illustrates the use of statistical methods to measure the dynamic energy consumption during the execution of an application.

The API is confined in the *hcl* namespace. Lines 10–12 construct the Wattsup object. The inputs to the constructor are the paths to the scripts and their arguments that read the USB serial devices containing the readings of the power meters.

The principal method of *Wattsup* class is *execute*. The inputs to this method are the type of measure, the path to the executable *executablePath*, the arguments to the executable *executableArgs* and the statistical thresholds (*pIn*) The outputs are the achieved statistical confidence *pOut*, the estimators, the sample mean (*sampleMean*) and the standard deviation (*sd*) calculated during the execution of the executable.

The *execute* method repeatedly invokes the executable until one of the following conditions is satisfied:

- The maximum number of repetitions specified in $maxRepeats$ is exceeded.

```
#include <wattsup.hpp>
int main(int argc, char** argv)
{
    std::string pathsToMeters[2] = {
       "/opt/powertools/bin/wattsup1",
       "/opt/powertools/bin/wattsup2"};
    std::string argsToMeters[2] = {
       "--interval=1",
       "--interval=1"};
    hcl::Wattsup wattsup(
        2, pathsToMeters, argsToMeters
    );
    hcl::Precision pIn = {
        maxRepeats, cl, maxElapsedTime, maxStdError
    };
    hcl::Precision pOut;
    double sampleMean, sd;
    int rc = wattsup.execute(
               hcl::DENERGY, executablePath,
               executableArgs, &pIn, &pOut,
               &sampleMean, &sd
    );
    if (rc == 0)
       std::cerr << "Precision NOT achieved.\n";
    else
       std::cout << "Precision achieved.\n";
    std::cout << "Max repetitions "
              << pOut.reps_max
              << ", Elasped time "
              << pOut.time_max_rep
              << ", Relative error "
              << pOut.eps
              << ", Mean energy "
              << sampleMean
              << ", Standard Deviation "
              << sd
              << std::endl;
    exit(EXIT_SUCCESS);
}
```

Figure A.1: Example illustrating the use of HCLWattsUp API for measuring the dynamic energy consumption

- The sample mean is within $maxStdError$ percent of the confidence interval cl. The confidence interval of the mean is estimated using the Student's t-distribution.
- The maximum allowed time $maxElapsedTime$ specified in seconds has elapsed.

If anyone of the conditions is not satisfied, then a return code of 0 is output suggesting that statistical confidence has not been achieved. If statistical confidence has been achieved, then the number of repetitions performed, the time elapsed and the final relative standard error is returned in the output argument $pOut$. At the same time, the sample mean and standard deviation are returned. For our experiments, we use values of (1000, 95%, 2.5%, 3600) for the parameters ($maxRepeats, cl, maxStdError, maxElapsedTime$) respectively. Since we use Student's t-distribution for the calculation of the confidence interval of the mean, we confirm specifically that the observations follow normal distribution by plotting the density of the observations using the *R* tool.

A.3 Methodology to Obtain a Reliable Data Point

We follow the following strict methodology described below to make sure the experimental results are reliable:

- The server is fully reserved and dedicated to these experiments during their execution. We also made certain that there are no drastic fluctuations in the load due to abnormal events in the server by monitoring its load continuously for a week using the tool *sar*. Insignificant variation in the load was observed during this monitoring period suggesting normal and clean behavior of the server.
- We set the application kernel's CPU affinity mask using SCHED API's system call SCHED_SETAFFINITY() Consider for example MKL-DGEMM application kernel running on HCLServer1. To bind this application kernel, we set its CPU affinity mask to 12 physical CPU cores of Socket 1 and 12 physical CPU cores of Socket 2.

- To make sure that pipelining, cache effects, and so forth, do not happen, the experiments are not executed in a loop and sufficient time (120 s) is allowed to elapse between successive runs. This time is based on observations of the times taken for the memory utilization to revert to base utilization and processor (core) frequencies to come back to the base frequencies.

- To obtain a data point, the application is repeatedly executed until the sample mean lies in the 95% confidence interval and a precision of 0.025 (2.5%) has been achieved. For this purpose, Student's t-test is used assuming that the individual observations are independent and their population follows the normal distribution. We verify the validity of these assumptions by plotting the distributions of observations.

  The function $MeanUsingTtest$, shown in Algorithm 1, describes this step. For each data point, the function is invoked, which repeatedly executes the application app until one of the following three conditions is satisfied:

  1. The maximum number of repetitions ($maxReps$) has been exceeded (Line 3).
  2. The sample mean falls in the confidence interval (or the precision of measurement eps has been achieved) (Lines 15–17).
  3. The elapsed time of the repetitions of application execution has exceeded the maximum time allowed ($maxT$ in seconds) (Lines 18–20).

  So, for each data point, the function $MeanUsingTtest$ is invoked and the sample mean $mean$ is returned at the end of the invocation. The function $Measure$ measures the execution time or the dynamic energy consumption using the HCL's WattsUp library [132] based on the input, $TIME$ or $ENERGY$. The input minimum and the maximum number of repetitions, $minReps$ and $maxReps$, differ based on the problem size solved. For small problem sizes ($32 \leq n \leq 1024$), these values are set to 10,000 and 100,000 respectively. For medium problem sizes ($1024 <$

$n \leq 5120$), these values are set to 100 and 1000. For large problem sizes ($n > 5120$), these values are set to 5 and 50. The values of $maxT$, cl and eps are respectively set to 3600, 0.95 and 0.025. If the precision of measurement is not achieved before the maximum number of repeats has been completed, we increase the number of repetitions and also the maximum elapsed time allowed. However, we observed that condition (2) is always satisfied before the other two in our experiments.

A.3.1 Methodology to Determine the Component-Level Energy Consumption Using HCLWattsUp

We provide here the details of how system-level physical measurements using HCLWattsUp can be used to determine the energy consumption by a component (such as a CPU) during application execution.

We define the group of components running a given application kernel as an *abstract processor*. For example, consider a matrix multiplication application running on a multicore CPU. The abstract processor for this application, which we call *AbsCPU*, comprises of the multicore CPU processor consisting of a certain number of physical cores and DRAM. In this work, we use only such configurations of the application which execute on *AbsCPU* and do not use any other system resources such as solid-state drives (SSDs), network interface cards (NIC) and so forth. Therefore, the change in energy consumption of the system reported by HCLWattsUp reflects solely the contributions from CPU and DRAM. We take several precautions in computing energy measurements to eliminate any potential interference of the computing elements that are not part of the abstract processor *AbsCPU*. To achieve this, we take the following precautions:

1. We ensure the platform is reserved exclusively and fully dedicated to our experiments.

2. We monitor the disk consumption before and during the application run and ensure that there is no I/O performed by the application using tools such as *sar*, *iotop*, and so forth.

Algorithm 1 Function determining the sample mean using Student's *t*-test.

1: **procedure** MeanUsingTtest(
$app, minReps, maxReps,$
$maxT, cl, accuracy,$
$repsOut, clOut, etimeOut, epsOut, mean$)

Input:
The application to execute, app
The minimum number of repetitions, $minReps \in \mathbb{Z}_{>0}$
The maximum number of repetitions, $maxReps \in \mathbb{Z}_{>0}$
The maximum time allowed for the application to run, $maxT \in \mathbb{R}_{>0}$
The required confidence level, $cl \in \mathbb{R}_{>0}$
The required accuracy, $eps \in \mathbb{R}_{>0}$

Output:
The number of experimental runs actually made, $repsOut \in \mathbb{Z}_{>0}$
The confidence level achieved, $clOut \in \mathbb{R}_{>0}$
The accuracy achieved, $epsOut \in \mathbb{R}_{>0}$
The elapsed time, $etimeOut \in \mathbb{R}_{>0}$
The mean, $mean \in \mathbb{R}_{>0}$

2: $reps \leftarrow 0$; $stop \leftarrow 0$; $sum \leftarrow 0$; $etime \leftarrow 0$
3: **while** ($reps < maxReps$) and (!$stop$) **do**
4: $st \leftarrow measure(TIME)$
5: $Execute(app)$
6: $et \leftarrow measure(TIME)$
7: $reps \leftarrow reps + 1$
8: $etime \leftarrow etime + et - st$
9: $ObjArray[reps] \leftarrow et - st$
10: $sum \leftarrow sum + ObjArray[reps]$
11: **if** $reps > minReps$ **then**
12: $clOut \leftarrow$ fabs(gsl_cdf_tdist_Pinv(cl, $reps - 1$))
$\times$ gsl_stats_sd($ObjArray$, 1, $reps$)
/ sqrt($reps$)
13: **if** $clOut \times \frac{reps}{sum} < eps$ **then**
14: $stop \leftarrow 1$
15: **end if**
16: **if** $etime > maxT$ **then**
17: $stop \leftarrow 1$
18: **end if**
19: **end if**
20: **end while**
21: $repsOut \leftarrow reps$; $epsOut \leftarrow clOut \times \frac{reps}{sum}$
22: $etimeOut \leftarrow etime$; $mean \leftarrow \frac{sum}{reps}$
23: **end procedure**

3. We ensure that the problem size used in the execution of an application does not exceed the main memory and that swapping (paging) does not occur.

4. We ensure that the network is not used by the application by monitoring using tools such as *sar*, *atop*, etc.

5. We set the application kernel's CPU affinity mask using SCHED API's system call SCHED_SETAFFINITY() Consider for example MKL DGEMM application kernel running on only abstract processor *A*. To bind this application kernel, we set its CPU affinity mask to 12 physical CPU cores of Socket 1 and 12 physical CPU cores of Socket 2.

6. Fans are also a great contributor to energy consumption. On our platform fans are controlled in two zones: (a) zone 0: CPU or System fans, (b) zone 1: Peripheral zone fans. There are 4 levels to control the speed of fans:

   - Standard: BMC control of both fan zones, with CPU zone based on CPU temp (target speed 50%) and Peripheral zone based on PCH temp (target speed 50%)
   - Optimal: BMC control of the CPU zone (target speed 30%), with Peripheral zone fixed at low speed (fixed 30%)
   - Heavy IO: BMC control of CPU zone (target speed 50%), Peripheral zone fixed at 75%
   - Full: all fans running at 100%

   In all speed levels except the full, the speed is subject to be changed with temperature and consequently, their energy consumption also changes with the change of their speed. Higher the temperature of CPU, for example, higher the fans' speed of zone 0 and higher the energy consumption to cool down. This energy consumption to cool the server down, therefore, is not consistent and is dependent on the fans' speed and consequently can affect the dynamic energy consumption of the given application kernel.

Hence, to rule out the fans' contribution to dynamic energy consumption, we set the fans at full speed before launching the experiments. When set at full speed, the fans run consistently at a fixed speed until we do so to another speed level. Hence, fans consume the same amount of power which is included in the static power of the platform.

7. We monitor the temperature of the platform and speed of the fans (after setting it at full) with help of Intelligent Platform Management Interface (IPMI) sensors, both with and without the application run. We find no considerable difference in temperature and find the speed of fans the same in both scenarios.

Thus, we ensure that the dynamic energy consumption obtained using HCLWattsUp reflects the contribution solely by the *abstract processor* executing the given application kernel.

A.3.2 Methodology to Obtain Dynamic Energy Consumption Using Intel RAPL

We first present a brief on RAPL before introducing our methodology to compare the measurements of dynamic energy consumption by RAPL and HCLWattsUp.

RAPL (Running Average Power Limit) [30] provides a way to monitor and -dynamically-set the power limits on processor and DRAM. So, by controlling the maximum average power, it matches the expected power and cooling budget. RAPL exposes its energy counters through model-specific registers (MSRs) It updates these counters once in every 1 ms. The energy is calculated as a multiple of model-specific energy units. It divides a platform into four domains, which are presented below:

1. *PP0* (Core Devices): Power plane zero includes the energy consumption by all the CPU cores in the socket(s).

2. *PP1* (Uncore Devices): Power plane one includes the power consumption of integrated graphics processing unit – which is not available on

server platforms– uncore components.

3. *DRAM*: Refers to the energy consumption of the main memory.

4. *Package*: Refers to the energy consumption of entire socket including core and uncore: `Package = PP0 + PP1`.

PP0 is removed in the the Haswell E5 generation [33]. For our experiments, we use *Package* and *DRAM* domains to obtain the energy consumption by CPU and DRAM when executing a given application.

To obtain the energy consumption provided by RAPL, we use a well-known package, Intel PCM [83]. We ensure that the RAPL values output by this package is correct by comparing with values given by another well known package, PAPI [37].

To compare the RAPL and HCLWattsUp energy measurements, we use the following workflows of the experiments. The workflow to determine the dynamic energy consumption by the given application using RAPL follows:

1. Using Intel PCM and DE-Meter A.4, we obtain the *base power* of CPUs (core and un-core) and DRAM (when the given application is not running).

2. Using HCLWattsUp API, we obtain the *execution time* of the given application.

3. Using Intel PCM/PAPI, we obtain the *total energy* consumption of the CPUs and DRAM, during the execution of the given application.

4. Finally, we calculate the *dynamic energy* consumption (of CPUs and DRAM) by subtracting the *base energy* from *total energy* consumed during the execution of the given application.

The workflow to determine the dynamic energy consumption using HCLWattsUp follows:

1. Using HCLWattsUp API, we obtain the *base power* of the server (when the given application is not running).

2. Using HCLWattsUp API, we obtain the *execution time* of the application.
3. Using HCLWattsUp API, we obtain the *total energy* consumption of the server, during the execution of the given application.
4. Finally, we calculate the *dynamic energy* consumption by subtracting the *base power* from *total energy* consumed during the execution of the given application.

We make sure that the execution time of the application kernel is the same for dynamic energy calculations by both tools. So, any difference between the energy readings of the tools comes solely from their power readings.

We analyzed 51 energy profiles of different application configurations of the aforementioned applications, using RAPL and HCLWattsUp. Our configuration parameters are: (a) Problem size ($M \times N$) where $M \leq N$, (b) Number of CPU threads or number of CPU cores.

The cost in terms of a number of measurements to determine the dynamic energy consumption of the application using sensors is the same for both tools as we need three (*Base power, Execution Time and Total Energy*) measurements to obtain a single data point of the application dynamic energy profile.

A.4 DE-METER: Calculate Dynamic Energy Consumption Using RAPL Meter

DE-METER is a wrapper using Likwid tool to calculate the dynamic energy consumption of CPU and DRAM using RAPL within a given statistical confidence interval for any given application.

The following are the software requirements for DE-METER.

- Likwid tool
- Pyhton compiler
- Linux Debian or Cent OS

A.4.1 How to Use DE-METER

For getting dynamic energy consumption of any given application using default settings, specify the application in 'run_application.py' script and run using following command:

python ./get_dynamic_energy <application>

Get base power of server using:

python get_base.py <desired_varience (1-100)> <max_iterations> <verbosity (1 || 2)>

python get_base.py 5 100 1 1

Get total power of server running an application using:

python ./get_total_energy.py <desired_varience (1-100)> <max_iterations> <desired_std (e.g. 10)> <verbosity (1 || 2)> <application>

python ./get_total_energy.py 5 100 10 1 <./app>

The source code for DE-METER is available at `https://github.com/ArsalanShahid116/DE-METER`

Appendix B

Methodology for Collection of PMCs

This chapter provides the details about using PMC collection tools on HCLServers and methodology to obtain PMCs reliably.

B.1 List of PMC groups Provided by Likwid

The Likwid tools support likwid-perfctr as a command-line tool for the collection of PMCs. Its usage is shown in Figure B.1.

We used likwid-pin to pin the applications to specific cores on HCLServers for the experiments. The usage for likwid-pin is shown in Figure B.2.

The list of PMC groups provided by Likwid tool [38] on HCLServer2 is shown in the Figure B.3.

B.2 Brief overview of *SLOPE-PMC* and *AdditivityChecker*

SLOPE-PMC is developed on top of Likwid tool to automate the process of PMC collection. It takes an input application and operates in three steps. First, it identifies the available PMCs on a given platform and lists them in a file. In

```
$ likwid-perfctr

likwid-perfctr -- Version 4.3.0
A tool to read out performance counter registers on x86 processors

Options:
-h, --help      Help message
-v, --version   Version information
-V, --verbose  Verbose output, 0 (only errors), 1 (info), 2 (details), 3 (
    developer)
-c <list>       Processor ids to measure (required), e.g. 1,2-4,8
-C <list>       Processor ids to pin threads and measure, e.g. 1,2-4,8
                For information about the <list> syntax, see likwid-pin
-g, --group     Performance group or custom event set string
-H              Get group help (together with -g switch)
-s, --skip      Bitmask with threads to skip
-M <0|1>        Set how MSR registers are accessed, 0=direct, 1=
    accessDaemon
-a              List available performance groups
-e              List available events and counter registers-E <string>
-i, --info      Print CPU info
-T <time>       Switch event sets with given frequency
-f, --force     Force overwrite of registers if they are in use
Modes:-S        Stethoscope mode with duration in s, ms or us, e.g 20ms
-t <time>       Timeline mode with frequency in s, ms or us, e.g. 300ms
                The output format (to stderr) is:
-m, --marker    Use Marker API inside code
Output options:
-o, --output    Store output to file. (Optional: Apply text filter
    according to filename suffix)
-O              Output easily parseable CSV instead of fancy tables
--stats         Always print statistics table

Examples:
List all performance groups:
likwid-perfctr -a
List all events and counters:
likwid-perfctr -e
List all events and suitable counters for events with 'L2' in them:
likwid-perfctr -E L2
Run command on CPU 2 and measure performance group TEST:
likwid-perfctr -C 2 -g TEST ./a.out
```

Figure B.1: likwid-perfctr *options and usage*

```
$ likwid-pin -h

-h, --help     Help message
-v, --version  Version information
-V, --verbose  Verbose output, 0 (only errors), 1 (info), 2 (details), 3 (
    developer)
-i          Set NUMA interleave policy with all involved numa nodes
-m          Set NUMA membind policy with all involved numa nodes
-S, --sweep    Sweep memory and LLC of involved NUMA nodes
-c <list>      Comma separated processor IDs or expression
-s, --skip     Bitmask with threads to skip
-p          Print available domains with mapping on physical IDs
            If used together with -p option outputs a physical processor
    IDs.
-d <string>    Delimiter used for using -p to output physical processor
    list, default is comma.
-q, --quiet    Silent without output

// Command to use likwid-pin to pin myApp to 5 threads

$ likwid-pin -c 0,2,4-6  <myApp> <app-parameters>
```

Figure B.2: List of PMC groups provided by Likwid tool on HCLServer2

```
$ likwid-perfctr -a

 Group name     Description
-----------     ----------------------------------------
     BRANCH     Branch prediction miss rate/ratio
     CACHES     Cache bandwidth in MBytes/s
       CBOX     CBOX related data and metrics
      CLOCK     Power and Energy consumption
       DATA     Load to store ratio
     ENERGY     Power and Energy consumption
FALSE_SHARE     False sharing
  FLOPS_AVX     Packed AVX MFLOP/s
         HA     Main memory bandwidth in MBytes/s
                     seen from Home agent
     ICACHE     Instruction cache miss rate/ratio
         L2     L2 cache bandwidth in MBytes/s
    L2CACHE     L2 cache miss rate/ratio
         L3     L3 cache bandwidth in MBytes/s
    L3CACHE     L3 cache miss rate/ratio
        MEM     Main memory bandwidth in MBytes/s
       NUMA     Local and remote memory accesses
        QPI     QPI Link Layer data
   RECOVERY     Recovery duration
       SBOX     Ring Transfer bandwidth
   TLB_DATA     L2 data TLB miss rate/ratio
  TLB_INSTR     L1 Instruction TLB miss rate/ratio
       UOPS     UOPs execution info
  UOPS_EXEC     UOPs execution
 UOPS_ISSUE     UOPs issueing
UOPS_RETIRE     UOPs retirement
CYCLE_ACTIVITY Cycle Activities
```

Figure B.3: List of PMC groups provided by Likwid tool on HCLServer2

the second step, the input application is executed several times as in a single invocation of an application only 4 PMCs can be collected. To ensure reliable results, we also take an average of each PMC count using multiple executions (at least 3) of an application. In the final step, the PMCs are extracted with labels in a stats file. Figure B.4 summarizes the work-flow of *SLOPE-PMC*.

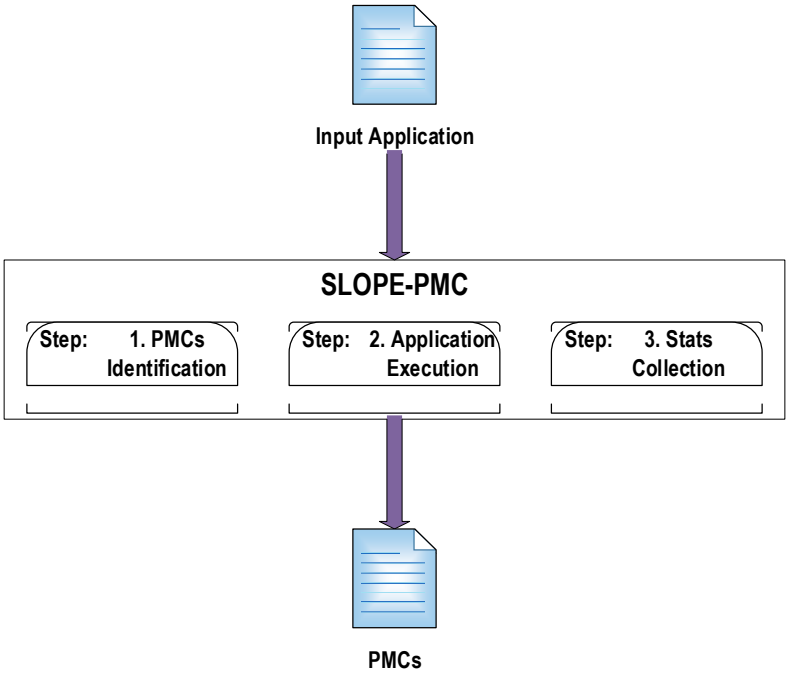


Figure B.4: SLOPE-PMC*: Towards the automation of PMC collection on Modern Computing Platforms*

Figure B.5 describes the *AdditivityChecker* where it takes as an input: 1). PMCs of two base applications *A* and *B*, and a compound application (*AB*) composed of base applications and 2). user-specified tolerance in percentage. It returns a list of *additive* and *non-additive* PMCs along with their percentage errors.

The source code for SLOPE-PMC is available online at: https://github.com/ArsalanShahid116/SLOPE-PMC

The source code for AdditivityChecker is available online at: `http://csgitlab.ucd.ie/hcl/SLOPE/tree/master/AdditivityChecker`

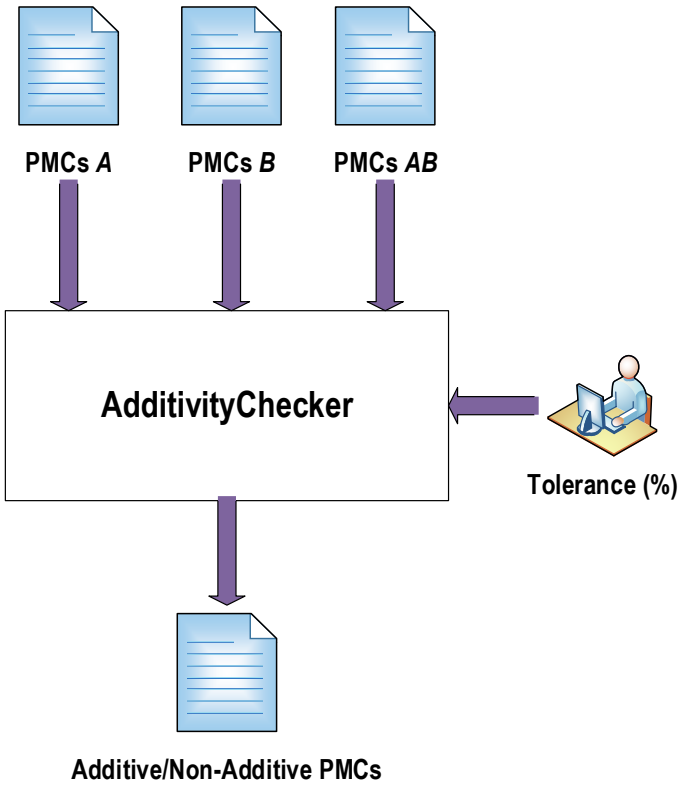


Figure B.5: AdditivityChecker*: Test PMCs for* Additivity

Appendix C

Methodology to Obtain Likwid and PAPI PMCs

C.1 LIKWID PMCs

In this section, we explain the experimental methodology to obtain Likwid PMCs.

A sample Likwid command-line invocation is shown below where *EVENTS* represents one or more PMCs, which are collected during the execution of the given application *APP*:

likwid-perfctr -f -C S0:0-11@S1:12-23 -g EVENTS APP

Here, the application (APP) during its execution is pinned to physical cores (0-11, 12-23) in our platform. Since Likwid do not provide option to bind application to memory, we have used *numactl*, i.e., a command-line linux tool, with option *–membind* to pin our applications to memory blocks (for our platform *numactl* gives 2 memory blocks, 0 and 1). The list of comma-separated PMCs is specified in *EVENTS*. For example, the following command:

likwid-perfctr -f -C S0:0-11@S1:12-23
-g ICACHE_ACCESSES:PMC0,ICACHE_MISSES:PMC1
numactl –membind=0,1 APP

determines the counts for two PMCs, *ICACHE_ACCESSES:PMC0* and *ICACHE MISSES:PMC1*.

The collection of all PMCs requires significant programming efforts and execution time because only a limited number of PMCs can be obtained in a single application run due to the limited number of registers dedicated to collecting PMCs. In addition, to ensure the reliability of our results, we follow a detailed statistical methodology where a sample mean of a PMC is used. It is calculated by executing the application repeatedly until it lies in the 95% confidence interval and a precision of 0.050 (5.0%) has been achieved. For this purpose, Student's t-test is used assuming that the individual observations are independent and their population follows the normal distribution. We verify the validity of these assumptions by plotting the distributions of observations.

Likwid provides 167 PMCs for our platform. In order to collect all of them for an application, we have to run the application 53 times. We wrote a software tool to automate this collection process, *SLOPE-PMC-LIKWID* [153].

Before we apply the *additivity* test, we remove few PMCs such as IIO_CREDIT (related to I/O and QPI), and $OFFCORE\_RESPONSE$ since they exhibit zero counts. We also remove PMCs having very low count (less than 10). The resulting dataset contained 151 performance events, which are then inputted to the *additivity* test.

C.2 PAPI PMCs

In this section, we explain the experimental methodology to obtain PAPI PMCs.

We check the available PAPI PMCs for our Intel Haswell platform using the command-line invocation, $'papi\_avail - a'$. We found that a total of 53 PMCs are available. The number of PMCs that can be gathered in a single application run varies. While gathering a set of 4 PMCs is common, there are a few event sets, which can contain up to 2 or 3 PMCs. Therefore, we found that an application has to be executed 14 times in order to collect all the PMCs for the application on our platform.

We wrote a software tool to automate the process of collection of PMCs, SLOPE-PMC-PAPI [153]. It is to be noted that for ensuring the reliability of our experimental results, we follow the same statistical methodology that was

followed for determining Likwid PMCs.

Appendix D

Calibration of WattsUp Pro power-meters

The dynamic energy consumption during the application execution is measured using a *WattsUp Pro* power meter on both servers (HCLServer1 and HCLServer2) and obtained programmatically via the HCLWattsUp interface [132]. The power meter is periodically calibrated using an ANSI C12.20 revenue-grade power meter, Yokogawa WT210. In this chapter, we explain our methodology and some results to calibrate our power-meters.

We compare the WattsUp Pro power-meter power measurements with Yokogawa using three methods that are explained as follows:

1. **Naked-eye visual monitoring**

    - We first attach the WattsUp Pro power-meters to both servers.
    - Once the servers are switched on and are in stable condition, we monitor the WattsUp pro LCDs and note the power readings in watts for both servers.
    - we carefully plugged off the WattsUp Pro power meters and plug the servers via Yokogawa power meter.
    - Once the servers are switched on and are in stable condition, we monitor the Yokogawa LCDs and note the power readings in watts for both servers.

- On comparison, we find a difference of 2 watts and 3 watts for HCLServer1 and HCLServer2, respectively.

2. **Monitoring server base power**

   - We connect both power meters to both servers one by one and once the servers are stable, we programmatically obtain the power readings from both power meters.
   - For WattsUp Pro, a Perl script provides the power readings with a granularity of 1 second. Similarly, Yokogawa comes with its own software and allows us to read power readings at the granularity of 1 second.
   - We measure and record the base powers of both servers for 3.5 hours using both power meters. Figure D.1(a) and D.1(b) compare the idle power profiles of HCLServer1 and HCLserver2 using both power meters, respectively. It can be seen that both profiles are almost the same. However, HCLServer1 has more power variations and HCLServer2 is considerably stable in terms of base power. For HCLServer2, there are power spikes after every half an hour for a couple of seconds. This is because of a daemon service being triggered by the OS.
   - Table D.1 show the minimum, average, and maximum power consumption for both servers and power meters. If we compare the average, WattsUp Pro gives 1-2 watts less power consumption than Yokogawa.

3. **Measurement of total energy consumption for two scientific applications**

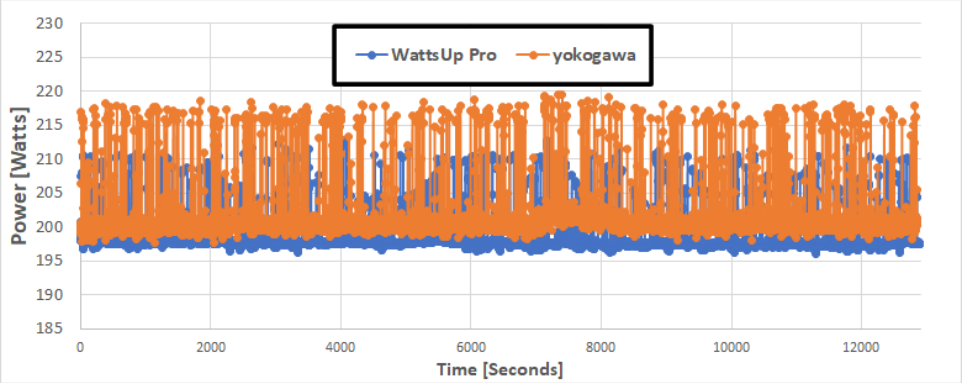
   - We choose two scientific applications: 1) DGEMM and 2) FFT from Intel MKL.
   - We execute DGEMM for problem sizes 4096$\times$4096 to 30720$\times$30720 with a constant step size of 1024 on HCLServer1. We then execute DGEMM for problem sizes 15360$\times$15360 to

30720×30720 with a constant step size of 1024 on HCLServer2. We build the total energy consumption profiles using both power meters for these application executions.

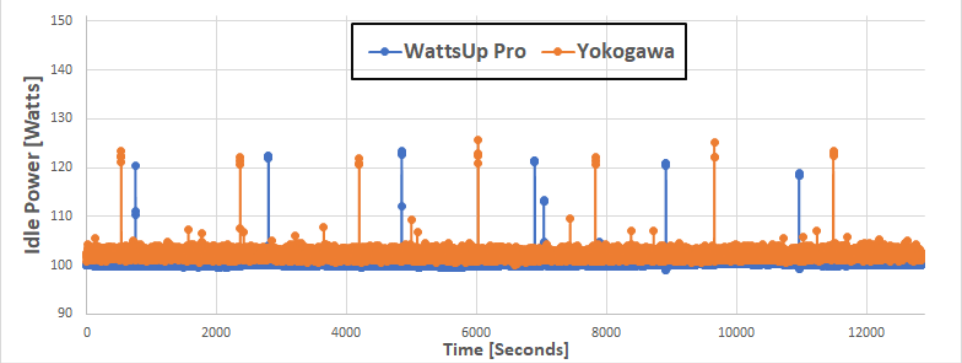
- We execute FFT for problem sizes 4096×4096 to 30720×30720 with a constant step size of 1024 on HCLServer1. We then execute FFT for problem sizes 8384×8384 to 62880×62880 with a constant step size of 2096 on HCLServer2. We build the total energy consumption profiles using both power meters for these application executions.
- Figure D.2 and D.3 show the total energy consumption profiles for DGEMM and FFT on both servers, respectively.
- Table D.2 show the relative error in percentage for total energy consumptions obtained using HCLServer1 and HCLServer2. It can be seen that the average measurement error for DGEMM is 4.95% and 9.25% on HCLServer2 and HCLServer1, respectively. For FFT, the average measurement error is 6% and 7.4% on HCLServer2 and HCLServer1, respectively.

Table D.1: Minimum, maximum and average of idle power using WattsUp Pro and Yokogawa PowerMeter on HCLServer1 and HCLServer2

	HCLServer1		HCLServer2	
	WattsUp Pro	Yokogawa	WattsUp Pro	Yokogawa
Min	196	197.72	99.1	99.93
Max	212.8	219.47	123.3	125.6
Average	198.2	201.0	100.03	102.06



(a)

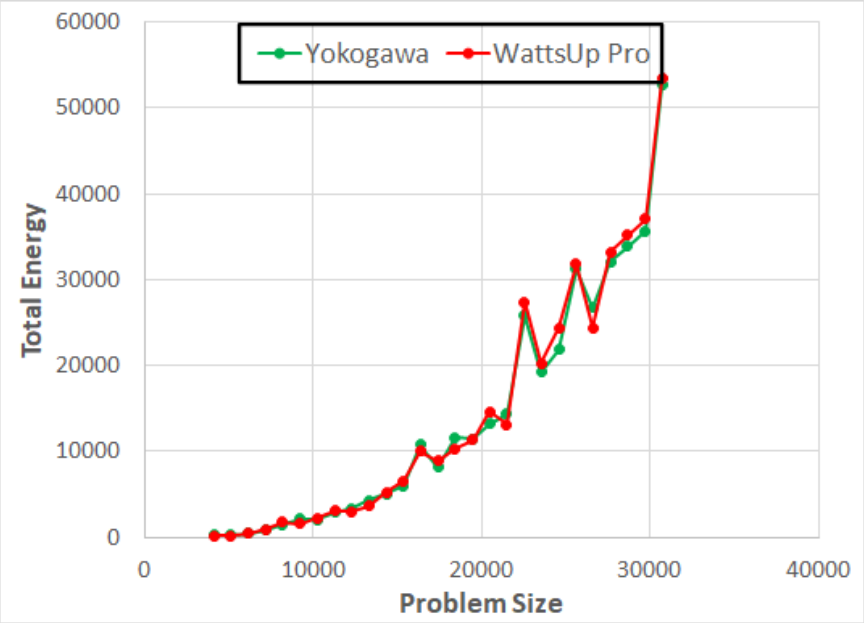


(b)

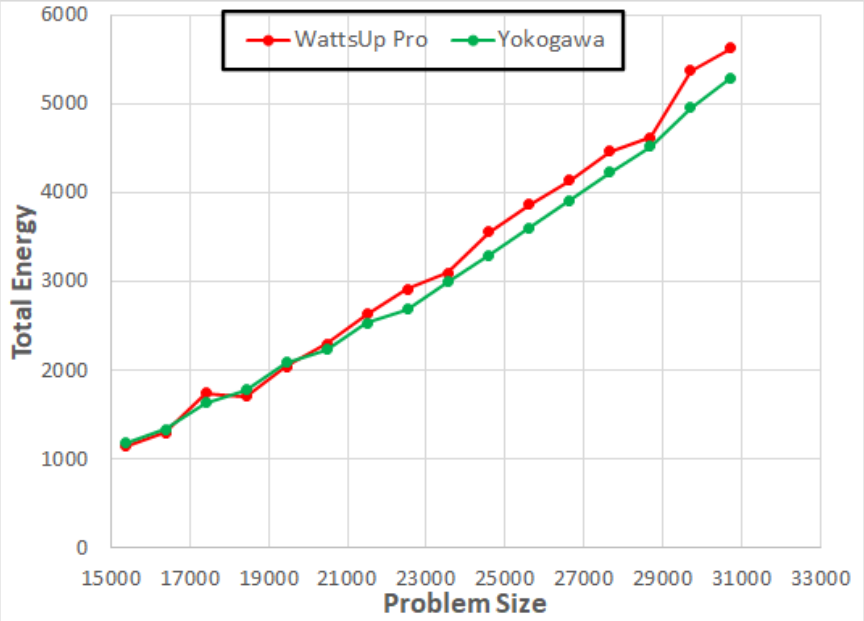
Figure D.1: Calibration test for idle power using WattsUp Pro and Yokogawa PowerMeter on (a) HCLServer1 and (b) HCLServer2

Table D.2: Comparison of minimum, average, and maximum measurement errors for DGEMM and FFT on HCLServer1 and HCLServer2 using WattsUp Pro and Yokogawa

	HCLServer2 Errors [%] (Min, Avg, Max)	HCLServer1 Errors [%] (Min, Avg, Max)
DGEMM	(2.01, 4.95, 8.16)	(0.58, 9.25, 33.18)
FFT	(0.11, 6.02, 15.84)	(0.20, 7.41, 16.90)

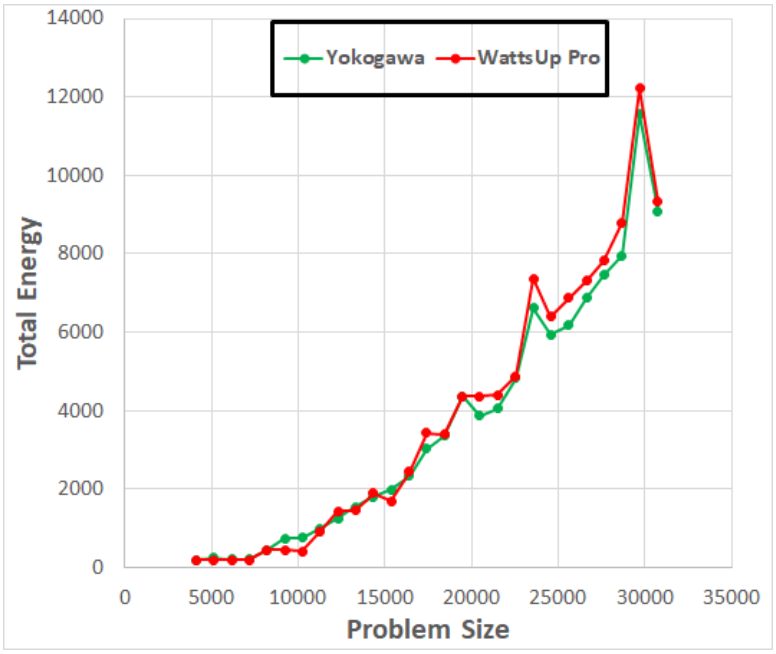


(a)

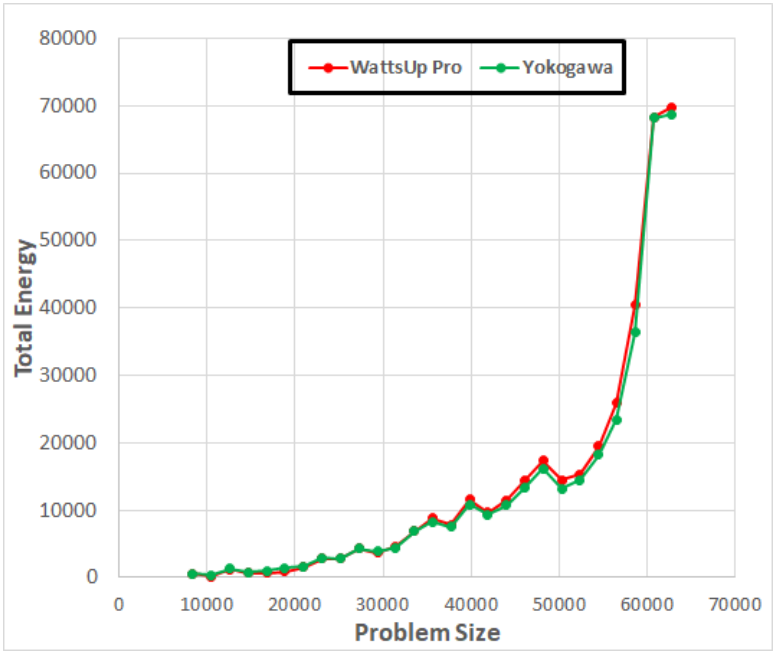


(b)

Figure D.2: Comparison of total power for Intel MKL DGEMM using WattsUp Pro and Yokogawa PowerMeter on (a) HCLServer1 and (b) HCLServer2



(a)



(b)

Figure D.3: Comparison of total power for Intel MKL FFT using WattsUp Pro and Yokogawa PowerMeter on (a) HCLServer1 and (b) HCLServer2

Appendix E

Employment of PMCs Selected Using Consistency Test in RF and NN

E.0.1 Platform-level Models

We employ the PMCs C1-C6 to build six RF and six NN models. The models use the same train and test set as we used to build LR models using approach A. Table E.1 and E.2 show the minimum, average, and maximum prediction accuracies of RF and NN models for the test sets, respectively. It can be seen that the prediction accuracy of RF and NN models is poor as compared to LR models. We observe a decrease in average prediction error from RF1 to RF4 and NN1 to NN4. The minimum average prediction errors for RF and NN models are obtained for RF4 (27.18%) and NN4 (26.06%), respectively. However, we observe an RF and NN based model perform worse when only one or two PMCs are employed as model variables for dynamic energy prediction. This is because a model fails to train well with only two PMCs.

E.0.2 Application-level Models

We build two random forest models, {RF-A,RF-NA}, and two neural network models, {NN-A,NN-NA}. The models {RF-A,NN-A} are trained using PMCs

Table E.1: Random forest (RF) regression-based energy predictive models (RF1-RF6) with their minimum, average, and maximum prediction errors.

Model	**PMCs**	**Relative prediction errors [%] (min, avg, max)**
RF1	$C_1, C_2, C_3, C_4, C_5, C_6$	(3.18, 38.2, 199.2)
RF2	C_1, C_2, C_3, C_4, C_5	(3.5, 33.4, 186.6)
RF3	C_1, C_2, C_3, C_4	(2.5, 30.02, 180.1)
RF4	C_1, C_2, C_3	(2.4, 27.18, 152.3)
RF5	C_1, C_2	(3.1, 43.4, 174.4)
RF6	C_1	(2.9, 57.7, 172.1)

Table E.2: Neural Networks based energy predictive models (NN1-NN6) with their minimum, average, and maximum prediction errors.

Model	**PMCs**	**Relative prediction errors [%] (min, avg, max)**
NN1	$C_1, C_2, C_3, C_4, C_5, C_6$	(2.1, 30.31, 192.3)
NN2	C_1, C_2, C_3, C_4, C_5	(2.02, 29.32, 201.2)
NN3	C_1, C_2, C_3, C_4	(1.9, 29.14, 160.1)
NN4	C_1, C_2, C_3	(2.2, 26.06, 180.3)
NN5	C_1, C_2	(8.5, 64.21, 192.6)
NN6	C_1	(8.8, 65.05, 201.6)

belonging to PA, and the models RF-NA,NN-NA are trained using PMCs belonging to PNA. Table E.3 show the relative and proportional prediction errors of the models. It can be seen that the models based on PA have better average prediction accuracies than the models based on PNA.

Table E.3: Prediction accuracies of LR models using nine PMCs.

Model	**PMCs**	**Relative Errors p in [%] (Min, Avg, Max)**	**Proportional Errors μ (Min, Avg, Max)**
RF-A	*PA*	(.001, 28.39, 132.2)	(1, 1.41, 4.42)
RF-NA	*PNA*	(0.031, 35.90, 1721)	(1.002, 3.71, 13.12)
NN-A	*PA*	(0.031, 14.31, 94.5)	(1, 1.43, 3.93)
NN-NA	*PNA*	(0.601, 19.82, 166.7)	(1.001, 4.21, 15.61)

We use PA and PNA to build two sets of four most energy correlated PMCs. The first set PA4, {$X1$, $X2$, $X4$, $X8$}, is constructed using PA and the second set PNA4, {$Y1$, $Y3$, $Y8$, $Y9$}, using PNA. We build two random forest models, {RF-A4,RF-NA4}, and two neural network models, {NN-A4,NN-NA4}. The models {RF-A4,NN-A4} are trained using PMCs belonging to PA4, and the models {RF-NA4,NN-NA4} are trained using PMCs belonging to PNA4. The training and test data-sets are the same as before.

Table E.4 shows the relative and proportional prediction errors of the models. Model NN-A4 has the least average of p and μ of 11.46% and 1.12. We can see that models {RF-NA4,NN-NA4} build using highly correlated but *non-additive* PMCs do not demonstrate much improvement in average prediction accuracies when compared to models {RF-NA,NN-NA} based on nine non-additive PMCs.

Table E.4: Prediction accuracies of LR, RF, and NN models using four PMCs.

Model	**PMCs**	**Relative Errors p in [%] (Min, Avg, Max)**	**Proportional Errors μ (Min, Avg, Max)**
RF-A4	PA4	(0.005, 22.73, 207.7)	(1, 1.21, 4.38)
RF-NA4	PNA4	(0.035, 38.06, 1628)	(1.021, 3.01, 8.21)
NN-A4	PA4	(0.003, 11.46, 152.2)	(1.001, 1.12, 3.58)
NN-NA4	PNA4	(0.016, 21.32, 227.5)	(1.03, 3.82, 9.37)

Appendix F

Employing High-Level Metrics as Model Variables in Energy Predictive Models: A Preliminary Study

In this thesis, we previously showed that all the PMCs on modern computing platforms are non-additive, in general, (for an input tolerance of 5%) and hence a model purely based on PMCs is pruned to prediction errors. In this chapter, we first identify the High-Level Metrics (HLM) from applications and hardware components and evaluate them to be used as model variables using the consistency test presented in Chapter 5 of this thesis. The hardware HLM includes the utilization of CPU and memory and the on-chip sensor energy measurements. The application HLM representing the activities of the application's execution includes floating-point operations (FLOP) and the total number of instructions. We demonstrate that employing high-level metrics as model variables to predict the dynamic energy consumption of the applications provides significant improvements in the prediction accuracies of the models. We denote the metrics that pass the consistency test as prime *HLM*.

We divide our experiments into three groups, i.e., group 1, group 2, and group 3. The experiments in groups 1 and 2 are performed on HCLServer1

and HCLServer2. However, group 3 only uses HCLServer1.

In group 1, we study in detail the accuracy of application-specific energy predictive models using three popular techniques: (a). Linear Regression (LR), (b). Random Forest (RF), and (c). Neural Networks (NN). The experiments are conducted on HCLServer1 and HCLServer2 using two highly optimized scientific applications from Intel Math Kernel Library (MKL): (1). DGEMM and (2). FFT. We build dynamic energy predictive models using pure utilization parameters, PMCs, and HLMs and analyze them in terms of their prediction errors.

In group 2, we study platform-level energy predictive models based on LR, RF, and NN. We first study the accuracy of energy predictive models employing pure utilizations, PMCs, and HLMs for a small set of applications. We further expand the data set with different types of applications (including memory-bound, compute-bound, optimized, and unoptimized kernels) using a vast range of problem sizes and then to analyze the prediction accuracy of the models.

Finally, in group 3, we conduct an experimental study to explore the techniques to build accurate and reliable energy predictive models for workload parallel applications running on a dual-socket multicore server (HCLServer1). We employ socket-level and platform-level PMCs and HLMs in energy models using LR, RF, and NN and study their prediction accuracies.

The main original contributions of this work are:

- A study exploring the High-Level Metrics (HLMs) that reflect the complete energy-consuming activities of the applications executing on a computing platform. We provide a detailed experimental methodology to extract and select the HLMs on modern multicore CPU platforms.

- An experimental study demonstrating the accuracy of application-specific energy predictive models based on LR, RF, and NN techniques using pure utilization parameters, PMCs, and HLMs. We show that the use of pure CPU and memory utilization as predictor variables in a model does not provide a good prediction accuracy, but should be combined with other high-level application metrics and on-chip sensor readings to

build a model with considerable accuracy. We further show that a model based on HLMs performs significantly better in terms of average prediction accuracy than a model employing only additive PMCs. We show that HLMs better reflect the dynamic energy-consuming activities of a processor executing an application.

- A study demonstrating the effectiveness of using HLMs as model variables in platform-level energy predictive models using LR, RF, and NN.
- A study exploring the effective techniques for accurate and reliable energy predictive models for workload-parallel applications executing on a dual-socket multicore CPU platform. We experimentally demonstrate that the use of socket-level HLMs as predictor variables results in the most accurate model; when compared with platform-level PMC-based, platform-level HLM-based, and socket-level PMC-based models. Therefore, socket-level HLMs better reflect the contributions of individually powered components (sockets) to the total dynamic energy consumption.

We organize the rest of this chapter as follows. The following section presents in-detail the methodology to select the predictor variables along with our experimental setups. We then present our experimental results, analysis, and discussions. The final chapter presents the summary as conclusions.

F.1 Selection Procedure for Model Variables

All the model variables are evaluated through the consistency test presented in Chapter 4.

F.1.1 Experimental Setup

Our experimental platforms include the latest Intel Haswell (HCLServer1) and Intel Skylake (HCLServer2) multicore CPU servers, whose specifications are

given in the Table 3.1. The detailed experimental workflow is illustrated in Figure 5.1.

Our application test suite is composed up of scientific applications that are highly optimized such as DGEMM and FFT from the Intel math kernel library (MKL), NASA benchmarking suite (NAS Parallel), and Intel optimized HPCG. These applications consist up of highly compute-bound and memory-bound kernels. To ensure the diversity of applications, we also include non-scientific applications such as *stress* in our application suite. Table 3.2 lists the applications along with their description.

For each application run on our platforms, we record the measurements for the following:

1. Dynamic energy consumption.
2. Execution time.
3. PMCs.
4. HLMs.

The dynamic energy consumption during the application execution is measured using a *WattsUp Pro* power meter and obtained programmatically via the HCLWattsUp API [132]. The power meter is periodically calibrated using an ANSI C12.20 revenue-grade power meter, Yokogawa WT210. PMCs and HLMs are obtained using the Likwid tool and Linux Perf.

Steps to Ensure Reliable Experiments

To ensure that our results are reliable, we follow a detailed statistical methodology that is summarized in Section 4.3.4.

We now apply the consistency test to select PMCs and HLMs. HLMs are the high-level application or hardware metrics that are the functional combination of individual PMCs and that reflect the complete activity of application execution on a platform.

F.1.2 Selection of Performance Monitoring Counters (PMCs)

We use *Likwid* [38],[39] to obtain the PMCs. It offers 164 and 323 PMCs on HCLServer1 and HCLServer2 (Table 3.1), respectively. The collection of all the PMCs is a tedious job and requires a lot of time because of the restriction to obtain a limited number of PMCs (3-4) for application execution. This restriction is because of the availability of a limited number of hardware registers to store them. We found that each application must be executed about 53 and 99 times to collect all the PMCs on HCLServer1 and HCLServer2, respectively.

We apply the first stage of consistency test, that is, to check if the PMCs are deterministic and reproducible using the following two steps:

1. We eliminate PMCs with counts less than or equal to 10. The eliminated PMCs have no significance on modelling energy consumption of our platform because we found them to be non-reproducible. We also remove several PMCs that count equal to zero. The reduced set contains 151 and 298 PMCs on Intel Haswell and Intel Skylake, respectively.

2. We compare the counts obtained for PMCs obtained using Likwid, PAPI, and Linux Perf. We find a difference in counts for several PMCs obtained using different tools. We eliminate these PMCs from our set. This elimination of PMCs results in 115 and 224 PMCs on Intel Haswell and Intel Skylake platform. We call the PMCs that pass the first stage as *prime PMCs*.

Literature (section 2.3.5) shows that the dominant groups from which PMCs are mainly selected for energy predictive models are cache, branch instructions, micro-operations (μops), and main memory activities. We make sure that the selected PMCs are from the list of prime PMCs and the mentioned dominant groups, and apply the second stage of consistency test.

Additivity of PMCs

After studying the additivity of PMCs, we conclude that all the PMCs fail the *additivity* test for a vast set of applications with a specified tolerance of 5% on current multicore platforms. However, the additivity test holds if applied on the PMCs of individual applications. Therefore, PMCs are not useful for platform-level energy predictive models but can be employed in application-specific models.

F.1.3 Selection of High-Level Metrics (HLMs)

We pick the high-level metrics that reflect the overall resource consumption and application activities of the platform when running an application. These metrics are given as below:

- Average CPU and memory utilization [(CPU, Memory) Util $\times$ Time]: To obtain the utilizations, we follow the following steps
  - Using an automated script, we collect the average CPU and memory utilization in percentage for the platform using Linux *ps* tool.
  - The script reads the CPU and memory utilization after every 0.25 seconds during application execution.
  - The overall CPU utilization at a particular instance is the average utilization of the individual cores employed in the execution of the application.
  - We apply the trapezoidal rule on the utilization profile for an application to obtain the average.
  - The average CPU and memory utilization are finally multiplied with the application execution time.
- Cache misses [C_{misses}]: C_{misses} represents the number of memory accesses that could not be served by any of the caches. We obtain their count using the *Linux Perf* tool.

- Memory accesses [$M_{accesses}$]: $M_{accesses}$ represents the total number of main memory accesses for application execution. We obtain their count using the Linux Perf tool.

- Floating-point operations [$FLOP_{total}$]: $FLOP_{total}$ represents the total number of floating-point operations for an application run. We use the Likwid tool to obtain their count.

- Instructions [Ins_{total}]: Ins_{total} represents the total number of instructions during execution of an application. We use the Likwid tool to obtain their count.

- Dynamic energy using RAPL [DE_{RAPL}]: Since dynamic energy consumption measurements for applications using RAPL are inaccurate [36], we investigate if RAPL readings can be used as a model parameter. We use an automated tool called *DE-Meter* [154] to obtain the dynamic energy consumption within specified statistical confidence (95% for our experiments) for an application run.

We evaluated each metric using the consistency test on HCLServer1 and HCLServer2.

First, we pass each metric from the first stage to check if they are deterministic and reproducible by executing applications in our test suite (Table 3.2) using different problem sizes on HCLServer1 and HCLServer2. We also write *assembly* level programs and theoretically determine the number of $FLOPS_{total}$ and Ins_{total} for them. We run the assembly programs on both experimental servers and record all the metric counts using Likwid and Linux Perf. The product of CPU and memory utilization with execution time and DE_{RAPL} using DE-Meter is deterministic and reproducible. We observe that $FLOPS_{total}$ and Ins_{total} obtained experimentally are equal to the theoretically calculated values. Hence, they are deterministic as well. However, we observe a difference in counts for $M_{accesses}$ and C_{misses} by using both tools. The maximum observed difference in counts is as high as 80% for $M_{accesses}$ and 75% for C_{misses}.

We then study the additivity of all the metrics except $M_{accesses}$ and C_{misses} (since they fail the first stage of consistency test).

We take the same application set used to study the additivity of PMCs (consisting of 60 and 40 compound applications composed from the base applications on both servers). The additivity test reveals that the metrics are highly additive (with errors less than 4%) for all the applications. We refer to the high-level metrics that pass both stages of consistency test as prime HLMs.

To summarize, prime HLMs can be employed as model variables in any platform-level linear energy predictive model. They include the following five parameters: 1). average CPU utilization$\times$Time, 2). average memory utilization$\times$Time, 3). $FLOPS_{total}$, 4). Ins_{total}, and 5). DE_{RAPL}.

F.2 Experiments and Analysis

The energy predictive models are build using three popular techniques: 1). Linear Regression (LR), 2). Random Forest (RF), and 3). Neural Networks (NN). These techniques are explained in section 5.3.4. The rest of this section is divided into the following three groups:

1. **Group 1:** We first study the accuracy of application-specific energy predictive models on HCLServer1 and HCLServer2 using pure utilization parameters, PMCs, and prime HLMs.

2. **Group 2:** We then study the platform-level energy predictive models on HCLServer1 and HCLServer2 using pure utilization parameters, PMCs, and prime HLMs. We divide the experiments in this section into two classes, class A and class B. In class A, we explore the prediction accuracies of LR, RF, and NN based models for a limited set of applications (DGEMM and FFT). In class B, we analyze models that employ data-sets from a variety of applications with a vast range of problem sizes.

3. **Group 3:** Finally, we study the prediction accuracies of PMC and HLM based models for workload-parallel applications on HCLServer1 (a mul-

ticore and dual-socket server). We analyze the prediction accuracies for models employing platform-wide and socket-wide parameters.

F.2.1 Group 1: Accuracy of Application-Specific Energy Predictive Models Using Pure Utilization Parameters, PMCs, and HLMs

We select two highly optimized scientific applications: 2-dimensional Fast Fourier Transform (FFT) and Dense Matrix-Multiplication application (DGEMM), from Intel Math Kernel Library (MKL).

We first outline a summary of the experimental steps in this section as below:

- We build two data-sets to study the *additivity* of raw PMCs for FFT and DGEMM containing the *compound* and *base* applications. Using the *additivity* test errors, we select the most *additive* PMCs that are common for both applications.
- We build a vast data-set containing dynamic energy consumptions, *additive* PMCs and the HLMs to build energy predictive models.
- We employ the utilization parameters, highly *additive* PMCs and prime HLMs in LR, RF, and NN techniques as predictor variables.
- Finally, we analyze the prediction accuracy for all models.

Procedure to Select Model Variables Using Consistency Test on HCLServer1 and HCLServer2

We now summarize the methodology to select the PMCs and HLMs to be employed in the models in the following steps:

- We build a dataset of 50 base applications using different problem sizes for DGEMM and FFT to apply the additivity test. The range of problem sizes for DGEMM is 6500×6500 to 20000×20000, and for FFT is $22400 \times$

22400 to 29000×29000. We select this range because of reasonable execution time ($>$ 3 seconds) of the applications on our platforms.

- For each application in a dataset, we measure the following: PMCs, HLMs, dynamic energy consumption, and the execution time. We also build a dataset of 30 *compound* applications from the serial execution of base applications. The additivity test based on the two datasets reveals that several PMCs are highly additive and are common for both applications. All the HLMs for both applications are highly additive and energy correlated with errors less than 0.5%.
- From the additivity test results on HCLServer1 and HCLServer2, we select PMCs that are commonly additive with additivity test errors of less than 0.5%. In total there are nine PMCs for both servers with an error equal to 0.5%. The PMCs are represented as set $S01A = \{A1, A2, A3, A4, A5, A6, A7, A8, A9\}$ and set $S02B = \{B1, B2, B3, B4, B5, B6, B7, B8, B9\}$ for HCLServer1 and HCLServer2, respectively.
- We calculate the correlation for all PMCs in *S01A* and *S02B* with dynamic energy consumption. The PMCs and their correlation factor with dynamic energy consumption are given in Table F.1.
- We also build two subsets with four most energy correlated PMCs from S01A and S01B and label them as *S01A-Corr* and *S02B-Corr*. S01A-Corr and S02B-Corr consist up of $A1$, $A5$, $A8$, $A9$ and $B1$, $B2$, $B4$, $B8$, respectively.

Energy Predictive Models for DGEMM and FFT

On both experimental platforms (Table 3.1), we build a dataset containing dynamic energy consumption, execution time, PMCs (Table F.1) and HLMs representing DGEMM and FFT for a range of problem sizes. We measure the dynamic energy consumption using *HCLWattsUP* API. The PMCs are collected using Likwid tool for each application, whereas, the HLMs are collected using

Table F.1: Selected additive PMCs and their correlations with dynamic energy consumption on, (a). HCLServer1 and (b). HCLServer2. 0 to 1 represents correlation factors of 0% to 100%, respectively.

	HCLServer1 PMCs	**Correlation**
$A1$	IDQ_MITE_UOPS	0.993
$A2$	CPU_CLOCK_UNHALTED_REF_XCLK	0.801
$A3$	OFFCORE_REQUESTS_ALL_DATA_RD	0.921
$A4$	L2_RQSTS_ALL_DEMAND_DATA_RD	0.502
$A5$	UOPS_ISSUED_TOTAL_CYCLES	0.962
$A6$	UOPS_EXECUTED_PORT_PORT_0	0.932
$A7$	UOPS_RETIRED_CORE_TOTAL_CYCLES	0.917
$A8$	L2_RQSTS_MISS	0.990
$A9$	UOPS_EXECUTED_PORT_PORT_6	0.992

(a)

	HCLServer2 PMCs	**Correlation**
$B1$	UOPS_RETIRED_CYCLES_GE_4_UOPS_EXEC	0.992
$B2$	FP_ARITH_INST_RETIRED_DOUBLE	0.993
$B3$	MEM_INST_RETIRED_ALL_STORES	0.870
$B4$	UOPS_EXECUTED_CORE	0.993
$B5$	UOPS_DISPATCHED_PORT_PORT_4	0.870
$B6$	IDQ_DSB_CYCLES_6_UOPS	0.981
$B7$	IDQ_ALL_DSB_CYCLES_5_UOPS	0.972
$B8$	IDQ_ALL_CYCLES_6_UOPS	0.993
$B9$	MEM_LOAD_RETIRED_L3_MISS	-0.112

(b)

Linux Perf, Likwid, and Linux *ps* tool. The number of data points in the data-set and range of problem sizes for both applications is given in Table F.2. The dataset is also split into two subsets for training and testing the models.

Table F.2: Data-set for application specific models on HCLServer1 and HCLServer2

HCLServer1					
Application	**Range of Problem Sizes**	**Step Size**	**Total Data Points**	**Training Set**	**Testing Set**
DGEMM	12000×12000 to 24736×24736	64	200	150	50
FFT	40000×40000 to 44992×44992	64	79	59	20
HCLServer2					
DGEMM	6400×6400 to 38400×38400	64	401	300	101
FFT	22400×22400 to 41536×41536	64	300	225	75

We build models for MKL-FFT and MKL-DGEMM using LR, RF, and NN techniques employing the predictor variables form the training sets given in Table F.2 for HCLServer1 and HCLServer2. The models are evaluated using the test dataset. These models are divided into four categories as given below:

1. *Category A*: Energy Predictive models for FFT executing on HCLServer1 using LR, RF, and NN.

    - LRM-S01-UxT-FFT, RFM-S01-UxT-FFT, and NNM-S01-UxT-FFT use pure CPU and memory utilization as predictor variables.
    - LRM-S01A-FFT, RFM-S01A-FFT, and NNM-S01A-FFT use a highly additive PMC set (S01A) as predictor variables.
    - LRM-S01A-Corr-FFT, RFM-S01A-Corr-FFT, and NNM-S01A-Corr-FFT use the top four high positively correlated PMCs (S01A-Corr) as predictor variables.
    - LRM-S01-FFT-HLM, RFM-S01-FFT-HLM, and NNM-S01-FFT-HLM use prime HLMs as predictor variables.

2. *Category B*: Energy Predictive models for DGEMM executing on HCLServer1 using LR, RF, and NN.

    - LRM-S01-UxT-DGEMM, RFM-S01-UxT-DGEMM, and NNM-S01-UxT-DGEMM use pure CPU and memory utilization as predictor variables.

- LRM-S01A-DGEMM, RFM-S01A-DGEMM, and NNM-S01A-DGEMM use highly additive PMC set (S01A) as predictor variables.
- LRM-S01A-Corr-DGEMM, RFM-S01A-Corr-DGEMM, and NNM-S01A-Corr-DGEMM use top four high positively correlated PMCs (S01A-Corr) as predictor variables.
- LRM-S01-DGEMM-HLM, RFM-S01-DGEMM-HLM, and NNM-S01-DGEMM-HLM use prime HLMs as predictor variables.

3. *Category C*: Energy Predictive models for FFT executing on HCLServer2 using LR, RF, and NN.

   - LRM-S02-UxT-FFT, RFM-S02-UxT-FFT, and NNM-S02-UxT-FFT use pure CPU and memory utilization as predictor variables.
   - LRM-S02B-FFT, RFM-S02B-FFT, and NNM-S02B-FFT use a highly additive PMC set (S01A) as predictor variables.
   - LRM-S02B-Corr-FFT, RFM-S02B-Corr-FFT, and NNM-S02B-Corr-FFT use top four high positively correlated PMCs (S01A-Corr) as predictor variables.
   - LRM-S02-FFT-HLM, RFM-S02-FFT-HLM, and NNM-S02-FFT-HLM use prime HLMs as predictor variables.

4. *Category D*: Energy Predictive models for DGEMM executing on HCLServer2 using LR, RF, and NN.

   - LRM-S02-UxT-DGEMM, RFM-S02-UxT-DGEMM, and NNM-S02-UxT-DGEMM use pure CPU and memory utilization as predictor variables.
   - LRM-S02B-DGEMM, RFM-S02B-DGEMM, and NNM-S02B-DGEMM use highly additive PMC set (S01A) as predictor variables.

- LRM-S02B-Corr-DGEMM, RFM-S02B-Corr-DGEMM, and NNM-S02B-Corr-DGEMM use top four high positively correlated PMCs (S01A-Corr) as predictor variables.
- LRM-S02-DGEMM-HLM, RFM-S02-DGEMM-HLM, and NNM-S02-DGEMM-HLM use prime HLMs as predictor variables.

Table F.3, F.4, F.5, and F.6 shows the minimum, average, and maximum percentage prediction errors for the models in category A, B, C, and D, respectively.

NNM-S01-FFT-HLM, RFM-S01-DGEMM-HLM, RFM-S02-FFT-HLM, and RFM-S02-DGEMM-HLM result in minimum average prediction errors of 5.1%, 6.9%, 1.1%, and 2.8% for models in category A, B, C, and D, respectively.

Table F.3: Prediction Accuracies for Application-Specific Models in Category A

Model	**Predictor variables**	**Prediction Errors (%) [min, avg, max]**
LRM-S01-UxT-FFT	(CPU,Memory) Util $\times$ Time	(2.72, 35.23, 89.41)
RFM-S01-UxT-FFT	(CPU,Memory) Util $\times$ Time	(2.12, 23.31, 50.25)
NNM-S01-UxT-FFT	(CPU,Memory) Util $\times$ Time	(1.92, 15.41, 32.12)
LRM-S01A-FFT	S01A	(1.71, 16.22, 44.91)
RFM-S01A-FFT	S01A	(0.02, 13.21, 45.85)
NNM-S01A-FFT	S01A	(0.01, 9.71, 32.20)
LRM-S01A-Corr-FFT	S01A-Corr	(2.15, 14.23, 42.15)
RFM-S01A-Corr-FFT	S01A-Corr	(0.25, 9.52, 28.25)
NNM-S01A-Corr-FFT	S01A-Corr	(0.02, 8.01, 30.10)
LRM-S01-FFT-HLM	HLMs	(1.95, 9.22, 33.95)
RFM-S01-FFT-HLM	HLMs	(0.013, 6.17, 25)
NNM-S01-FFT-HLM	HLMs	(0.05, 5.17, 25.00)

Discussions

Following are the salient observations from the results:

- The models employing only utilization parameters provide poor prediction accuracy for all model categories. The average prediction accuracy

Table F.4: Prediction Accuracies for Application-Specific Models in Category B

Model	Predictor variables	Prediction Errors (%) [min, avg, max]
LRM-S01-UxT-DGEMM	(CPU,Memory) Util $\times$ Time	(2.21, 37.71, 53.23)
RFM-S01-UxT-DGEMM	(CPU,Memory) Util $\times$ Time	(1.33, 21.20, 46.12)
NNM-S01-UxT-DGEMM	(CPU,Memory) Util $\times$ Time	(1.66, 23.40, 41.31)
LRM-S01A-DGEMM	S01A	(1.21, 20.11, 89.01)
RFM-S01A-DGEMM	S01A	(0.22, 9.12, 42.10)
NNM-S01A-DGEMM	S01A	(0.11, 15.42, 71.01)
LRM-S01A-Corr-DGEMM	S01A-Corr	(0.01, 15.21, 82.27)
RFM-S01A-Corr-DGEMM	S01A-Corr	(0.12, 8.92, 45.14)
NNM-S01A-Corr-DGEMM	S01A-Corr	(0.01, 13.07, 72.25)
LRM-S01-DGEMM-HLM	HLMs	(0.18, 10.98, 51.77)
RFM-S01-DGEMM-HLM	HLMs	(0.08, 6.92, 34.13)
NNM-S01-DGEMM-HLM	HLMs	(0.13, 11.07, 52.55)

Table F.5: Prediction Accuracies for Application-Specific Models in Category C

Model	Predictor variables	Prediction Errors (%) [min, avg, max]
LRM-S02-UxT-FFT	(CPU,Memory) Util $\times$ Time	(0.92, 50.21, 88.16)
RFM-S02-UxT-FFT	(CPU,Memory) Util $\times$ Time	(1.20, 9.23, 32.54)
NNM-S02-UxT-FFT	(CPU,Memory) Util $\times$ Time	(0.43, 19.74, 61.23)
LRM-S02B-FFT	S02B	(0.447, 36.31, 182.2)
RFM-S02B-FFT	S02B	(0.069, 4.97, 42.81)
NNM-S02B-FFT	S02B	(0.077, 9.328, 73.85)
LRM-S02B-Corr-FFT	S02B-Corr	(0.042, 25.12, 190.15)
RFM-S02B-Corr-FFT	S02B-Corr	(0.002, 2.02, 37.21)
NNM-S02B-Corr-FFT	S02B-Corr	(0.024, 6.012, 67.15)
LRM-S02-FFT-HLM	HLMs	(0.03, 19.07, 123,2)
RFM-S02-FFT-HLM	HLMs	(0.22, 1.16, 16.6)
NNM-S02-FFT-HLM	HLMs	(0.003, 3.47, 64.64)

Table F.6: Prediction Accuracies for Application-Specific Models in Category D

Model	**Predictor variables**	**Prediction Errors (%) [min, avg, max]**
LRM-S02-UxT-DGEMM	(CPU,Memory) Util $\times$ Time	(2.12, 31.33, 53.02)
RFM-S02-UxT-DGEMM	(CPU,Memory) Util $\times$ Time	(0.12, 8.13, 23.23)
NNM-S02-UxT-DGEMM	(CPU,Memory) Util $\times$ Time	(1.31, 21.32, 102.32)
LRM-S02B-DGEMM	S02B	(0.094, 22.62, 125.48)
RFM-S02B-DGEMM	S02B	(0.008, 4.89, 63.55)
NNM-S02B-DGEMM	S02B	(0.007, 11.25, 131.23)
LRM-S02B-Corr-DGEMM	S02B-Corr	(0.004, 16.12, 87.25)
RFM-S02B-Corr-DGEMM	S02B-Corr	(0.012, 3.12, 50.12)
NNM-S02B-Corr-DGEMM	S02B-Corr	(0.014, 10.22, 130.21)
LRM-S02-DGEMM-HLM	HLMs	(0.45, 9.40, 63.72)
RFM-S02-DGEMM-HLM	HLMs	(0.09, 2.82, 63.34)
NNM-S02-DGEMM-HLM	HLMs	(0.05, 5.41, 124.57)

for RF and NN based models are comparatively better than LR based models for DGEMM and FFT applications executing on HCLServer1 and HCLServer2.

- The average prediction accuracy for models employing additive PMCs (S01A and S01B) is better as compared to models using only pure utilization parameters as predictor variables. The accuracy further improves for the models employing the top four most positively correlated PMCs (S01A-Corr and S02B-Corr). We find that the models using highly additive and the most positively correlated PMCs yield better prediction accuracy for application-specific models. In our previous work [146], we also show that the correlation if applied to non-additive PMCs does not improve prediction accuracy. We find that the models employing less than four PMCs for DGEMM and FFT does not improve the prediction accuracy. Therefore, to build an accurate energy predictive model, one must employ at least four PMCs.

- We also build models using more than four PMCs as model variables. We find that the average prediction accuracy for models using five or six most additive and positively correlated PMCs can be better than the one

employing four PMCs. But, since the modern computing platforms only support 3-4 hardware registers for storing PMCs, a five or six parameters model can not be employed for online energy predictions.

- The best prediction accuracy has been achieved for models that employ prime HLMs as predictor variables. This is because all prime HLMs are 1). highly additive, 2). positively correlated, and 3). they represent the complete energy-consuming activities for the applications' execution on our platforms.

F.2.2 Group 2: Improving the Accuracy of Platform-Level Energy Predictive Models

In this section, we study the accuracy of platform-level energy predictive models for a set of applications (Table 3.2). We divide our experiments into two classes:

1. Class A: We study the improvements in the prediction accuracy of energy predictive models using HLMs when compared with utilization and prime PMC based models for a limited set of applications (i.e., DGEMM and FFT).
2. Class B: We analyze the prediction accuracy of energy predictive models employing pure utilization parameters and prime HLMs for a dataset obtained using a variety of applications executing a vast range of problem sizes on HCLServer1 and HCLServer2.

Class A: Analysis of Prediction Accuracies of Energy Predicted Models for a Limited Set of Applications

The experiments in this class are run on HCLServer1 and HCLServer2 (Table 3.1). Since, we choose commonly *additive* PMCs and HLMs for MKL-DGEMM and MKL-FFT for experiments on application-specific models (Section F.2.1), we combine the dataset for both applications. We build the following two sets of models using the extended dataset:

- Set A: LRM-MMFT-S01-UxT, RFM-MMFT-S01-UxT, NNM-MMFT-S01-UxT, LRM-MMFT-S01A, RFM-MMFT-S01A, NNM-MMFT-S01A, LRM-MMFT-S01A-Corr, RFM-MMFT-S01A-Corr, NNM-MMFT-S01A-Corr, LRM-MMFT-S01-HLM, RFM-MMFT-S01-HLM, NNM-MMFT-S01-HLM are the models employing pure CPU and memory utilization, additive PMCs (S01A and S01A-Corr) or prime HLMs on HCLServer1 using LR, RF, and NN, respectively.

- Set B: LRM-MMFT-S02-UxT, RFM-MMFT-S02-UxT, NNM-MMFT-S02-UxT, LRM-MMFT-S02B, RFM-MMFT-S02B, NNM-MMFT-S02B, LRM-MMFT-S02B-Corr, RFM-MMFT-S02B-Corr, NNM-MMFT-S02B-Corr, LRM-MMFT-S02-HLM, RFM-MMFT-S02-HLM, NNM-MMFT-S02-HLM are the models employing pure CPU and memory utilization, additive PMCs (S02B and S02B-Corr) or prime HLMs on HCLServer2 using LR, RF, and NN, respectively.

The training and testing sets for the models on HCLServer1 and HCLServer2 are 153 and 66, and, 490 and 211, respectively. Table F.7 shows the minimum, average, and maximum percentage prediction errors for the models built-in class A.

Following are the salient observations from the results:

- RFM-MMFT-S01-HLM and NNM-MMFT-S02-HLM result in minimum average prediction errors of 4.6% and 7.2% for HCLServer1 and HCLServer2, respectively. We discover that for a small set of applications, the RF and NN based models can lead to better prediction accuracy.

- Each model parameter represents the energy-consuming activity for an application run on a platform. Based on the nature of the application, the impact of a model parameter may change towards their contribution to the overall energy consumption. We find the overall average prediction accuracy for an application-specific model is better than the models with the combined dataset for two applications.

Table F.7: Prediction Accuracies for Energy Predictive Models in set A and set B.

Model	Predictor variables	Prediction Errors (%) [min, avg, max]
	Set A Models	
LRM-MMFT-S01-UxT	(CPU,Memory) Util $\times$ Time	(2.32, 22.39, 121.31)
RFM-MMFT-S01-UxT	(CPU,Memory) Util $\times$ Time	(0.42, 12.35, 82.12)
NNM-MMFT-S01-UxT	(CPU,Memory) Util $\times$ Time	(0.49, 19.82, 81.31)
LRM-MMFT-S01A	S01A	(0.014, 18.62, 125.48)
RFM-MMFT-S01A	S01A	(0.028, 9.89, 63.55)
NNM-MMFT-S01A	S01A	(0.041, 12.25, 131.23)
LRM-MMFT-S01A-Corr	S01A-Corr	(0.014, 16.12, 87.25)
RFM-MMFT-S01A-Corr	S01A-Corr	(0.042, 6.12, 50.12)
NNM-MMFT-S01A-Corr	S01A-Corr	(0.090, 10.22, 130.21)
LRM-MMFT-S01-HLM	HLMs	(0.25, 10.40, 63.72)
RFM-MMFT-S01-HLM	HLMs	(0.39, 4.62, 69.54)
NNM-MMFT-S01-HLM	HLMs	(0.01, 6.41, 124.57)
	Set B Models	
LRM-MMFT-S02-UxT	(CPU,Memory) Util $\times$ Time	(1.23, 39.21, 132.23)
RFM-MMFT-S02-UxT	(CPU,Memory) Util $\times$ Time	(1.05, 35.41, 160.21)
NNM-MMFT-S02-UxT	(CPU,Memory) Util $\times$ Time	(0.89, 20.16, 150.94)
LRM-MMFT-S02B	S02B	(0.005, 35.32, 225.5)
RFM-MMFT-S02B	S02B	(.0001, 29.39, 157.4)
NNM-MMFT-S02B	S02B	(0.001, 15.43, 104.2)
LRM-MMFT-S02B-Corr	S02B-Corr	(0.024, 25.12, 87.25)
RFM-MMFT-S02B-Corr	S02B-Corr	(0.005, 22.73, 207.7)
NNM-MMFT-S02B-Corr	S02B-Corr	(0.003, 11.46, 152.2)
LRM-MMFT-S02-HLM	HLMs	(0.20, 17.27, 112.90)
RFM-MMFT-S02-HLM	HLMs	(0.01, 16.32, 115.34)
NNM-MMFT-S02-HLM	HLMs	(0.03, 7.21, 144.57)

- The prediction accuracy of additive PMC-based models is better than the ones using only utilization parameters and predictor variables. The average prediction errors of the PMC-based models employing most positively energy correlated PMCs (S01A-Corr and S02B-Corr) significantly drop when compared with models using all additive PMCs (S01A and S02B).

***Class B:* Analysis of Prediction Accuracies of Energy Predictive Models for a Broad Set of Applications Employing Pure Utilization Parameters and HLMs**

We build a dataset of 586 points and 1008 points on HCLServer1 and HCLServer2 using the applications in our testsuite (Table 3.2). The input parameters for the applications used to build the dataset are as follows:

- MKL FFT: Problem Size = 40000×40000 to 44992×44992 on HCLServer1 and 22400×22400 to 41536×41536 on HCLServer2 with step size of 64, verbosity = 0, Iteration = 1.
- MKL DGEMM: Problem Size = 12000×12000 to 24736×24736 for HCLServer1 and 6400×6400 to 38400×38400 on HCLServer2 with step size of 64, verbosity = 0, Iteration = 1.
- Intel HPCG: Problem Size = 40×40×40 to 240×240×240 with step size of 8, Iterations = 1, Threads = 48 (HCLServer1) and 44 (HCLServer2).
- NAS OMP 3D FT: Problem Size = 256×256×128, Iterations 95 to 395 with step size of 5, Threads 48 (HCLServer1) and 44 (HCLServer2).
- NAS OMP 3D LU: Problem Size = 70×70×70 to 139×139×139 with step size of 1, Iterations = 250, Threads = 48 (HCLServer1) and 44 (HCLServer2).
- NAS OMP 3D SP: Problem Size = 84×84×84 to 139×139×139 with step size of 1, Iterations = 100, dt = 0.0015000, Threads = 48 (HCLServer1) and 44 (HCLServer2).

- NAS OMP 3D BT: Problem Size = 128×128×128 to 190×190×190 with step size of 1, Iterations = 200, dt = 0.0008000, Threads = 48 (HCLServer1) and 44 (HCLServer2).
- stress: Problem Size = 4 seconds to 45 seconds with a step size of 1.

For each application configuration, we measure the dynamic energy consumption, execution time, and HLMs. 410 and 705 points have been used to train and 176 and 303 points for testing the models on HCLServer1 and HCLServer2, respectively.

We build six platform-level models using pure CPU and memory utilization. These models are LRM-PL-S01-UxT, RFM-PL-S01-UxT, NNM-PL-S01-UxT, for HCLServer1, and, LRM-PL-S02-UxT, RFM-PL-S02-UxT, NNM-PL-S02-UxT for HCLServer2. Similarly, we build six platform-level models that use prime HLMs. These models are LRM-PL-S01-HLM, RFM-PL-S01-HLM, and NNM-PL-S01-HLM for HCLServer1, and, LRM-PL-S02-HLM, RFM-PL-S02-HLM, and NNM-PL-S02-HLM for HCLServer2. Table F.8 show the prediction accuracies for platform level models.

Table F.8: Prediction accuracies for platform level energy predictive models.

Model	**Predictor variables**	**Prediction Errors (%) [min, avg, max]**
LRM-PL-S01-UxT	(CPU,Memory) Util × Time	(0.11, 37.35, 140.05)
RFM-PL-S01-UxT	(CPU,Memory) Util × Time	(0.32, 25.17, 130.85)
NNM-PL-S01-UxT	(CPU,Memory) Util × Time	(0.09, 21.63, 152.90)
LRM-PL-S02-UxT	(CPU,Memory) Util × Time	(0.37, 40.73, 197.02)
RFM-PL-S02-UxT	(CPU,Memory) Util × Time	(0.12, 31.68, 99.65)
NNM-PL-S02-UxT	(CPU,Memory) Util × Time	(0.19, 24.04, 121.65)
LRM-PL-S01-HLM	HLMs	(0.06, 14.36, 50.75)
RFM-PL-S01-HLM	HLMs	(0.058, 7.91, 55.15)
NNM-PL-S01-HLM	HLMs	(0.05, 7.74, 52.10)
LRM-PL-S02-HLM	HLMs	(.003, 20.69, 157.49)
RFM-PL-S02-HLM	HLMs	(0.00, 11.38, 91.23)
NNM-PL-S02-HLM	HLMs	(0.06, 13.54, 97.08)

Discussions

Following are the salient observations from the results:

- The minimum average prediction errors for platform level models on HCLServer1 and HCLServer2 are 7.7% (for NNM-PL-S01-HLM) and 11.3% (for RFM-PL-S02-HLM), respectively.
- Although the utilization parameters (average CPU and memory utilization for the application execution on a platform) are additive and highly correlated, their use in models do not provide a better prediction accuracy. This is because they do not represent all the energy-consuming activities of processor components during the application run. However, the use of all prime HLMs (that include utilization parameters) yields more accurate energy predictive models as they represent all the energy-consuming activities for our application set on our platform.

F.2.3 Group 3: Energy Predictive Models for Workload Parallel Applications on a Dual Socket Multicore CPU Platform

In this section, we present an experimental study to explore the techniques for accurate and reliable energy predictive models for data or workload parallel applications on a dual-socket (socket0 and socket1) multicore CPU platform, i.e., HCLServer1 (Table 3.1). The models employ platform-wide (PW) and socket-wide (SW) parameters (PMCs or HLMs) as predictor variables. We define PW parameters as features obtained for the applications' activity on the whole platform. However, the SW parameters track the individual resource utilization of the applications' activity on individually powered components (i.e., sockets). The application set includes DGEMM and FFT applications from Intel MKL. We design three sets of experiments using the following workload configurations of applications running on both sockets: 1). same-kernel and same-workload, 2). same-kernel and different-workload, 3). different-kernel and different-workload. These configurations are explained below:

- Same-kernel and same-workload (SKSW): In this set of experiments, we execute two DGEMM applications with the same problem sizes on both sockets (one kernel on socket1 and one on socket2) in parallel. The

range of problem sizes varies from 9728×9728 to 33792×33792, with a constant step size of 512.

- Same-kernel and different-workload (SKDW): This set of experiments is divided into the following two subsets:

  1. We first execute DGEMM with two different ranges of workload or problem sizes. Let us call them as range A and range B. Range A and range B uses workload size from 10000×10000 to 15000×15000 and 15000×15000 to 20000×20000, respectively, with a constant step size of 64. To obtain data points, we run a workload load size from Range A on socket0 and another workload size from range B and run on socket1 in parallel.
  2. The second subset of experiments include the execution of two different workload sizes for FFT application on each socket in parallel. The ranges selected for this application are 20000×20000 to 22500×22500 and 22500×22500 to 25000×25000, respectively, with a constant step size of 64.

- Different-kernel and different-workload (DKDW): In this set of experiments, we execute a range of problem sizes for FFT and DGEMM on socket0 and socket1 in parallel. Problem size range for FFT vary from 20000×20000 to 22432×22432, and for DGEMM, 10000×10000 to 12400×12400. The step size is 64 for both the ranges.

In each experimental set, we measure the following for all workload configurations to build a data-set:

- The dynamic energy consumption of the data-parallel applications executing on both sockets of HCLServer1 using HCLWattsUp API.
- The execution time of each workload configuration.
- We obtain two sets of top three most additive and high positively correlated PMCs (A1,A8,A9) from Table F.1a by using Likwid tool. We

label the PMC sets as 1) PW-PMCs and 2). SW-PMCs. The PW-PMCs contain the overall activity count of the platform for the workload parallel applications while executing on both sockets. However, SW-PMCs contain the activity count for individual workload sizes of the applications while executing on each socket. We label the PW-PMCs as P-PMC-A1,P-PMC-A8,P-PMC-A9 and SW-PMCs as S0-PMC-A1,S1-PMC-A1,S0-PMC-A8,S1-PMC-A8,S0-PMC-A9,S1-PMC-A9.

- Finally, we collect the HLMs for all workload configurations of applications involved in our experimental sets. We label the HLM sets as 1) PW-HLMs and 2). SW-HLMs. PW-HLMs contains the overall activity counts and SW-HLMs contains the individual activity counts (at a socket level) for the applications' execution.

The datasets for all the experimental sets are divided into training and testing sets. For SKSW, we used 33 and 14 points to train and test the models. For SKDW, 58 and 29 points are used to train, and, 20 and 10 points are used to test the DGEMM and FFT models, respectively. Finally, for DKDW, we used 30 and 10 points to train and test the energy predictive models.

For each experimental set, we build the following predictive models using PW-PMCs, SW-PMCs, PW-HLMs, and SW-HLMs:

- Same-kernel and same-workload models: LR-SKSW-PW-PMCs, RF-SKSW-PW-PMCs, NN-SKSW-PW-PMCs, LR-SKSW-SW-PMCs, RF-SKSW-SW-PMCs, NN-SKSW-SW-PMCs, LR-SKSW-PW-HLMs, RF-SKSW-PW-HLMs, NN-SKSW-PW-HLMs, LR-SKSW-SW-HLMs, RF-SKSW-SW-HLMs, NN-SKSW-SW-HLMs.

- Same-kernel and different workload models for DGEMM (SKDWMM): LR-SKDWMM-PW-PMCs, RF-SKDWMM-PW-PMCs, NN-SKDWMM-PW-PMCs, LR-SKDWMM-SW-PMCs, RF-SKDWMM-SW-PMCs, NN-SKDWMM-SW-PMCs, LR-SKDWMM-PW-HLMs, RF-SKDWMM-PW-HLMs, NN-SKDWMM-PW-HLMs, LR-SKDWMM-SW-HLMs, RF-SKDWMM-SW-HLMs, NN-SKDWMM-SW-HLMs.

- Same-kernel and different workload models for FFT (SKDWFT): LR-SKDWFT-PW-PMCs, RF-SKDWFT-PW-PMCs, NN-SKDWFT-PW-PMCs, LR-SKDWFT-SW-PMCs, RF-SKDWFT-SW-PMCs, NN-SKDWFT-SW-PMCs, LR-SKDWFT-PW-HLMs, RF-SKDWFT-PW-HLMs, NN-SKDWFT-PW-HLMs, LR-SKDWFT-SW-HLMs, RF-SKDWFT-SW-HLMs, NN-SKDWFT-SW-HLMs.
- Different-kernel and different workload models: LR-DKDW-PW-PMCs, RF-DKDW-PW-PMCs, NN-DKDW-PW-PMCs, LR-DKDW-SW-PMCs, RF-DKDW-SW-PMCs, NN-DKDW-SW-PMCs, LR-DKDW-PW-HLMs, RF-DKDW-PW-HLMs, NN-DKDW-PW-HLMs, LR-DKDW-SW-HLMs, RF-DKDW-SW-HLMs, NN-DKDW-SW-HLMs.

Table F.9: Prediction accuracies for platform-wide and socket-wide PMC-Based models for data-parallel applications

Platform-Level PMC-Based Models	**Prediction Errors (%) [Min, Avg, Max]**	**Socket-Level PMC-Based Models**	**Prediction Errors (%) [Min, Avg, Max]**
LR-SKSW-PW-PMCs	(0.02, 4.2, 12.13)	LR-SKSW-SW-PMCs	(0.38, 3.23, 10.12)
RF-SKSW-PW-PMCs	(0.34, 3.9, 7.42)	RF-SKSW-SW-PMCs	(0.92, 3.34, 6.42)
NN-SKSW-PW-PMCs	(63.23, 72.34, 80.23)	NN-SKSW-SW-PMCs	(0.73, 2.64, 5.13)
LR-SKDWMM-PW-PMCs	(0.59, 3.5, 10.29)	LR-SKDWMM-SW-PMCs	(0.67, 2.98, 7.6)
RF-SKDWMM-PW-PMCs	(0.01, 3, 6.8)	RF-SKDWMM-SW-PMCs	(0.15, 2.99, 8.41)
NN-SKDWMM-PW-PMCs	(62.03, 81.95, 96.63)	NN-SKDWMM-SW-PMCs	(0.14, 1.2, 5.6)
LR-SKDWFT-PW-PMCs	(1.18, 15.56, 31.81)	LR-SKDWFT-SW-PMCs	(2.64, 13.87, 24.81)
RF-SKDWFT-PW-PMCs	(1.21, 16.70, 32.09)	RF-SKDWFT-SW-PMCs	(0.19, 13.49, 29.35)
NN-SKDWFT-PW-PMCs	(46.08, 61.14, 79.42)	NN-SKDWFT-SW-PMCs	(0.05, 3.34, 10.86)
LR-DKDW-PW-PMCs	(0.48, 7.10, 19.29)	LR-DKDW-SW-PMCs	(2.44, 6.61, 13.5)
RF-DKDW-PW-PMCs	(0.12, 8.37, 19.13)	RF-DKDW-SW-PMCs	(0.13, 5.55, 15.52)
NN-DKDW-PW-PMCs	(35.48, 82.26, 126.23)	NN-DKDW-SW-PMCs	(0.01, 2.75, 5.73)

Table F.9 and F.10 show the prediction accuracy of energy predictive models for workload parallel applications using PMCs and HLMs, respectively.

Discussions

Following are the salient observations from the results:

- The average prediction accuracies for PMC and HLM based models employing SW parameters are significantly better than the ones employing

Table F.10: Prediction accuracies for platform-level and socket-wide HLM-Based models for data-parallel applications

Platform-Level HLM-Based Models	Prediction Errors (%) [Min, Avg, Max]	Socket-Level HLM-Based Models	Prediction Errors (%) [Min, Avg, Max]
LR-SKSW-PW-HLMs	(0.39, 2.31, 8.34)	LR-SKSW-SW-HLMs	(0.02, 1.72, 4.52)
RF-SKSW-PW-HLMs	(0.22, 2.77, 8.12)	RF-SKSW-SW-HLMs	(0.02, 1.10, 5.26)
NN-SKSW-PW-HLMs	(0.31, 3.21, 7.21)	NN-SKSW-SW-HLMs	(0.02, 1.12, 4.95)
LR-SKDWMM-PW-HLMs	(0.15, 2.86, 6.54)	LR-SKDWMM-SW-HLMs	(0.11, 2.49, 5.72)
RF-SKDWMM-PW-HLMs	(0.07, 2.50, 7.47)	RF-SKDWMM-SW-HLMs	(0.10, 2.45, 7.06)
NN-SKDWMM-PW-HLMs	(0.06, 1.88, 6.80)	NN-SKDWMM-SW-HLMs	(0.06, 1.16, 3.28)
LR-SKDWFT-PW-HLMs	(3.24, 13.89, 21.28)	LR-SKDWFT-SW-HLMs	(3.83, 9.63, 12.24)
RF-SKDWFT-PW-HLMs	(0.32, 12.31, 31.49)	RF-SKDWFT-SW-HLMs	(1.97, 8.29, 17.48)
NN-SKDWFT-PW-HLMs	(0.79, 8.79, 28.73)	NN-SKDWFT-SW-HLMs	(0.05, 3.34, 10.86)
LR-DKDW-PW-HLMs	(0.67, 5.02, 7.99)	LR-DKDW-SW-HLMs	(0.04, 3.05, 7.34)
RF-DKDW-PW-HLMs	(0.26, 6.48, 16.64)	RF-DKDW-SW-HLMs	(0.63, 5.24, 12.48)
NN-DKDW-PW-HLMs	(0.29, 3.15, 10.41)	NN-DKDW-SW-HLMs	(0.001, 2.09, 6.25)

PW parameters. The sockets are independent power-consuming components of a processor and a parallel application running on two sockets consumes the computing resources differently. To explain better, a DGEMM or FFT application solving a small workload size may finish earlier on one socket than the one solving a comparatively larger workload size on the other socket. Therefore, the resource consumption becomes asymmetric. We conclude that using socket-wide parameters in an energy predictive model is better equipped to capture the contributions of individually powered computing components (for example, sockets) employed in the execution of an application. As a result, the models yield better average prediction accuracy.

- The neural network models using platform-wide PMCs (NN-SKSW-PW-PMCs, NN-SKDWMM-PW-PMCs, NN-SKDWFT-PW-PMCs, and NN-DKDW-PW-PMCs) perform poorly in terms of average prediction accuracies. This is because of the neural network under-fits for our given dataset. However, for the models employing socket-wide PMCs (NN-SKSW-SW-PMCs, NN-SKDWMM-SW-PMCs, NN-SKDWFT-SW-PMCs, and NN-DKDW-SW-PMCs), the neural network trains well and provides the least average prediction error for all the applications in the experimental sets.

- The average prediction accuracy of the models employing PW HLMs is

almost equal or even better (for configurations such as LR-SKSW, RF-SKSW, etc.) than the models employing SW PMC. This shows the effectiveness of using HLMs that captures all the energy-consuming activities in a processor during an application run.

- The minimum average prediction errors are obtained using SW HLMs in an energy predictive model. This is because the SW HLMs better capture the individual energy-consuming activities of application execution on a platform.

We, therefore, conclude that for a data-parallel application, the average prediction accuracy of energy predictive models improve when the PMCs and HLM representing the activity count of individually powered components (such as sockets) are used as predictor variables.

F.3 Summary

In this chapter, we first explored the high-level metrics (HLMs) that pass the consistency test and that represents all the energy-consuming activities during the execution of an application on a platform. The HLMs include high-level application parameters and on-chip sensor readings. We presented a detailed experimental methodology to extract and select the energy monitoring counters (HLMs) on modern multicore CPU platforms.

In an experimental study, we used the pure utilization parameters, PMCs, and HLMs to demonstrate the accuracy of application-specific energy predictive models based on linear regression (LR), random forest (RF), and neural network (NN). We showed that the use of pure CPU and memory utilization as predictor variables in a model worsens the prediction accuracy. The accuracy of additive PMC based models is better than pure utilization based models. However, the results demonstrated that HLM-based models (containing high-level application metrics and on-chip sensor readings) are the most accurate in terms of average prediction accuracy. We showed that the HLMs better reflect the dynamic energy-consuming activities of a processor executing an application.

We further presented a study demonstrating the effectiveness of HLMs for platform-level energy predictive models using LR, RF, and NN. We demonstrated that an HLM based model performs better than a pure utilization based model in terms of average prediction accuracy.

Finally, we presented an experimental study to explore the effective techniques for accurate and reliable energy predictive models for workload-parallel applications executing on a dual-socket multicore CPU platform. We demonstrated that the use of socket-level HLMs as predictor variables results in the most accurate model; when compared with platform-level PMC-based, platform-level HLM-based, and socket-level PMC-based models. We concluded that the socket-level HLMs better reflect the contributions of individually powered components (sockets) to the total dynamic energy consumption.

Appendix G

Study of Dynamic Energy Predictive Modelling for Data-Parallel Applications on Dual-socket Multicore CPU platform

In this chapter, we present an experimental study of energy predictive models that employ platform-level PMCs and socket-level PMCs for data-parallel applications on a dual-socket multicore CPU platform. We use the dual-socket (socket0 and socket1) HCLServer1 (Table 3.1) for our experiments. The application set includes DGEMM and FFT applications from Intel MKL. We design three sets of experiments described below:

- *Set A*: The first set of experiments includes DGEMM with two different ranges of workload/problem sizes (say, range A and range B). Range A and range B use workload size from 10000 $\times$ 10000 to 15000 $\times$ 15000 and 15000 $\times$ 15000 to 20000 $\times$ 20000, respectively, with a constant step size of 64. One workload is selected from Range A and run on socket0 and one workload is selected from range B and run on socket1 in parallel. We call this workload distribution as same kernel and different

workload configuration.

- *Set B*: The second set of experiments includes FFT application. The ranges selected for this application are 20000 $\times$ 20000 to 22500 $\times$ 22500 and 22500 $\times$ 22500 to 25000 $\times$ 25000, respectively, with a constant step size of 64.

- *Set C*: The third set of experiments are run using a range of problem sizes for FFT and DGEMM executing on socket0 and socket1 in parallel. The range of problem sizes for FFT include 20000 $\times$ 20000 to 22432 $\times$ 22432 and for DGEMM, 10000 $\times$ 10000 to 12400 $\times$ 12400. A constant step size of 64 is used for both the ranges. We call this workload distribution as a different kernel and different workload configuration.

For each set of experiments, we measure the following for all the workload configurations:

1. The dynamic energy consumption of the data-parallel applications executing on both sockets of HCLServer1 using HCLWattsUp API.

2. The execution time of each workload configuration. We make sure that for each applications' run, the execution time is over three seconds.

3. Using Likwid tool, we collect two sets of top three most additive and positively correlated PMCs (X1, X5, X6) from Table 4.5. We label the PMC sets as: 1) platform-level (*PW*) and 2). socket-level (*SW*) PMCs. The PW PMCs contain the overall count of the activity when the parallel application is executing on both sockets. However, SW PMCs holds the activity count for individual workload size executing on each socket. We label the PMCs in PW as {PWX1,PWX2,PWX3} and SW as {S0X1,S1X1,S0X2,S1X2,S0X3,S1X3}.

The datasets for all the experimental sets are divided into training and testing sets. 58, 29, and 30 data points are used to train and 20, 10, and 10 data points are used to test the models from set A, set B, and set C, respectively.

We build two predictive models for each experimental set using PW and SW PMCs, which are given below:

- Set A Models: {A-PW, A-SW}.
- Set B Models: {B-PW, B-SW}.
- Set C Models: {C-PW, C-SW}.

Table G.1 shows the prediction accuracies of set A, set B, and set C models. For each set of models, the two socket-level models have different coefficients suggesting asymmetric use of the resources of the multicore processor.

G.0.1 Discussion

Following are the salient observations from the results:

- The minimum average prediction error of 2.98% is obtained for A-SW. The high average prediction accuracy means that PMCs employed in the model best represents the energy-consuming activities for DGEMM application on our platform.
- The average prediction accuracies for models employing socket-level PMCs are better than those for models with platform-level PMCs. This is because a parallel application running on two sockets (that are independent power-consuming components of a processor) consumes the resources differently. An application executing a small workload may finish sooner on one socket than the one executing a larger workload on the other socket. In other words, the resource consumption is asymmetric. We conclude that using socket-level PMCs in an energy predictive model captures well the individual contributions of power-consuming components (sockets) executing an application. Hence, it improves the average prediction accuracy of a model.

We conclude therefore that for a data-parallel application, the average prediction accuracy of energy predictive models improves when the PMCs representing the activity count of individually powered components (such as sockets) are used as predictor variables.

Table G.1: Prediction accuracies for platform-level and socket-level models for data-parallel applications.

Platform-level Models	**Prediction Errors (%) [Min, Avg, Max]**	**Socket-level Models**	**Prediction Errors (%) [Min, Avg, Max]**
A-PW	(0.59, 3.5, 10.29)	A-SW	(0.67, 2.98, 7.6)
B-PW	(1.18, 15.56, 31.81)	B-SW	(2.64, 13.87, 24.81)
C-PW	(0.48, 7.10, 19.29)	C-SW	(2.44, 6.61, 13.5)

Appendix H

Experimental Observations Demotivating the Use of Intel RAPL for Energy Optimization

We first explain the experimental observations that lead us to investigate if inaccurate energy measurements using Intel RAPL can affect application-level energy optimizations.

On HCLServer1, we conduct the experiments using the Intel MKL FFT application. We execute the application for the problem sizes 8960×8960 to 35712×35712 with a constant step size of 128. We measure the energy consumption using Intel RAPL and HCLWattsUp API. Figure H.1 shows the energy profiles. The comparison results show that the average prediction error is 35.9%. Although the results show RAPL values to be inaccurate, we further analyzed if the RAPL readings follow the trend as system-level measurements. The results show that RAPL does not follow the trend for several points in the energy profile. The bold dots in the zoomed graph of Figure H.1 shows the instances where RAPL readings deviate from HCLWattsUp readings. These variations are because of severe resource contention and non-uniform memory access [155].

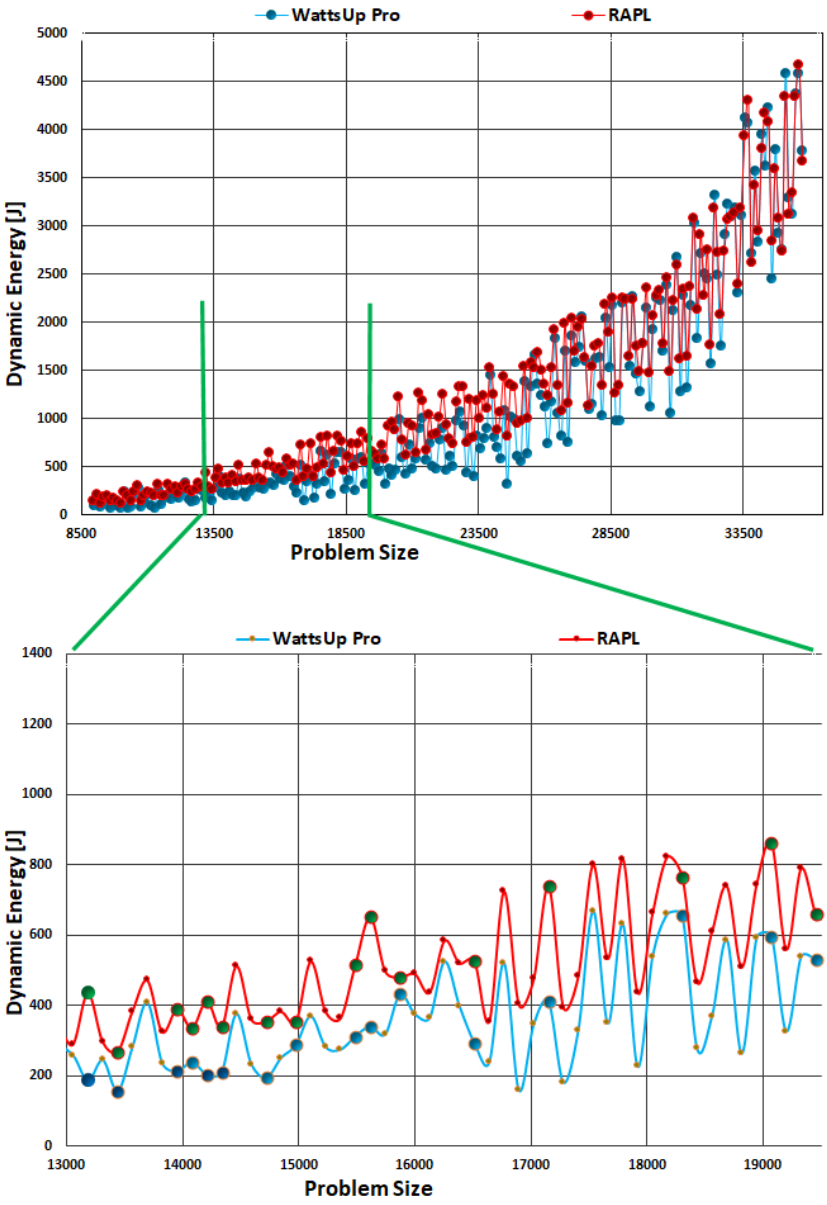


Figure H.1: Dynamic energy consumption of RAPL and HCLWattsUp on HCLServer1.

Acronyms

CMOS Complementary Metal-oxide-semiconductor. 46

DGEMM Double-precision General Matrix Multiplication. xviii, 5, 43, 54, 59, 60, 100, 104, 107, 180, 231

DVFS Dynamic Voltage and Frequency Scaling. 8, 9, 12, 45

FFT Fast Fourier Transform. xviii, 43, 54, 59, 60, 100, 104, 107, 231

FPGA Field Programmable Gate Array. 2, 41, 151

GPU Graphics Processing Unit. 6, 11, 14, 16, 28, 36, 37, 41, 46, 54, 66, 151

HPC High Performance Computing. 6, 15, 23, 29, 42–44, 66

ICT Information and Communication Technologies. v, 5, 148

LLC Last Level Cache. 9, 21, 40, 44, 89

NUMA Non-Uniform Memory Access. 4, 9, 21, 22, 26, 44, 52, 89, 92, 111, 112, 150

PCA Principal Component Analysis. 15

PMC Performance Monitoring Counter. xiii, 12, 16, 35–37, 91, 191

QPI Quick Path Interconnect. 21, 40, 91, 192

Xeon Phi Intel Xeon Phi. 2, 6, 11, 41, 151