Fall 2017

Modeling Energy Consumption of High-Performance Applications on Heterogeneous Computing Platforms

Gary D. Lawson Jr.
Old Dominion University

MODELING ENERGY CONSUMPTION OF HIGH-PERFORMANCE APPLICATIONS ON HETEROGENEOUS COMPUTING PLATFORMS

by

Gary D. Lawson Jr.
B.S. June 2010, Old Dominion University
M.S. December 2011, Old Dominion University

A Dissertation Submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

MODELING AND SIMULATION

OLD DOMINION UNIVERSITY
October 2017

Approved by:

Masha Sosonkina (Director)

Yuzhong Shen (Director)

Duc Nguyen (Member)

Manwo Ng (Member)

Tal Ezer (Member)

ABSTRACT

MODELING ENERGY CONSUMPTION OF HIGH-PERFORMANCE APPLICATIONS ON HETEROGNEOUS COMPUTING PLATFORMS

Gary D. Lawson Jr.
Old Dominion University, 2017
Director: Masha Sosonkina
Director: Yuzhong Shen

Achieving Exascale computing is one of the current leading challenges in High Performance Computing (HPC). Obtaining this next level of performance will allow more complex simulations to be run on larger datasets and offer researchers better tools for data processing and analysis. In the dawn of Big Data, the need for supercomputers will only increase. However, these systems are costly to maintain because power is expensive. Thus, a better understanding of power and energy consumption is required such that future hardware can benefit.

Available power models accurately capture the relationship to the number of cores and clock-rate, however the relationship between workload and power is less understood. Thus, investigation and analysis of power measurements has been a focal point in this work with the aim to improve the general understanding of energy consumption in the context of HPC.

This dissertation investigates power and energy consumption of many different parallel applications on several hardware platforms while varying a number of execution characteristics. Multicore and manycore hardware devices are investigated in homogeneous and heterogeneous computing environments. Further, common techniques for reducing power and energy consumption are employed to each of these devices.

Well-known power and performance models have been combined to form the

Execution-Phase model, which may be used to quantify energy contributions based on execution phase and has been used to predict energy consumption to within 10%. However, due to limitations in the measurement procedure, a less intrusive approach is required.

The Empirical Mode Decomposition (EMD) and Hilbert-Huang Transform analysis technique has been applied in innovative ways to model, analyze, and visualize power and energy measurements. EMD is widely used in other research areas, including earthquake, brain-wave, speech recognition, and sea-level rise analysis and this is the first it has been applied to power traces to analyze the complex interactions occurring within HPC systems.

Probability distributions may be used to represent power and energy traces, thereby providing an alternative means of predicting energy consumption while retaining the fact that power is not constant over time. Further, these distributions may be used to define the cost of a workload for a given computing platform.

This dissertation is dedicated to my mother
for her passion, strength, and love for family and education
and for teaching me to read and write.

ACKNOWLEDGMENTS

Throughout my dissertation, I have received support and encouragement from numerous individuals. First, a special thanks to my committee members: Dr. Masha Sosonkina, Dr. Yuzhong Shen, Dr. Duc Nguyen, Dr. Manwo Ng, and Dr. Tal Ezer. Then a special thanks to my family, fiancée, friends, and colleagues.

I would like to thank Dr. Nguyen for introducing me to parallel computing, Dr. Ng for introducing transportation modeling, and Dr. Ezer for introducing the EMD time-series analysis method. Thank each of you for investing time and energy into my education, and for your support and contributions to my dissertation and education.

I would like to extend a special thanks to Dr. Yuzhong Shen who has been my assiduous mentor throughout my entire graduate school career, both through my thesis and this dissertation. His course in computer graphics and game design inspired me to pursue graduate school. He has patiently devoted many years to my education by providing direction, inspiration, support, and valuable advice. Thank you.

I would like to extend a special thanks to my mentor, Dr. Masha Sosonkina, for her untiring devotion to me and my studies in high performance computing, and for the many opportunities provided to me to conduct my research and further my dissertation. Her course in HPC was captivating, and inspired me to pursue this dissertation. Thank you for your guidance, instruction, patience, and dedication.

I would also like to thank Dana Hammond and Robert Baurle for the opportunity to work with NASA in HPC, and to learn from experts in the field. Last but certainly not least, I would like to thank my friends, family, and colleagues.

To my parents, thank you for raising me to be independent and hard-working, and for pushing me to pursue my dreams. Their patience is unparalleled and I thank them for their wise advice, support, and love. To my brother and first best friend, thank you for all the laughs over the years and for always being there for me. To Elizabeth, thank you for your strength, love, and support; you truly are my other half. Finally, I want to thank my friends and colleagues for the many conversations, research related and especially otherwise, and for many more to come.

To all who have contributed to this dissertation and my education, sincerely, thank you.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Achieving Exascale computing is one of the current leading challenges in High Performance Computing (HPC). Obtaining this next level of performance is important because it allows more complex simulations to be run on larger datasets, current simulations may be run faster, and offers researchers better tools for analysis. HPC systems are commonly used for government, industry, and academic research projects. In the dawn of Big Data, the need for supercomputers will only increase as more data becomes readily available, and more conclusions can be made from these datasets[1].

1.1 Theoretical Formulations

Current HPC systems are rated according to the Top500 list, which ranks computing platforms by performance [99]. As of June 2017, the Sunway TaihuLight of China is the top ranked platform with 93 petaflops performance. Titan, the top ranked US platform installed in 2012, is ranked fourth with 17.6 petaflops [106].

The two systems differ significantly in architecture. Sunway is built with 256 processors 1.45GHz per node, each with 64KB scratchpad memory (16KB instruction), and communicate via a network on chip [105]. Titan is built using heterogeneous nodes, each contains a 16-core AMD Opteron CPU with 32 GB DDR3 ECC memory and an Nvidia Tesla K20X GPU with 2,688 CUDA cores 732 MHz and 6GB GDDR5 ECC memory. Nodes are interconnected using a 3D torus Gemini network.

[1] IEEE Transactions and Journals style is used in this dissertation for formatting figures, tables, and references.

The complexity of the two systems makes it difficult to pinpoint how specific design differences impact the final performance result. For example, consider investigating the differences in data transfer and power between the two systems.

- How much better is the scratchpad memory versus a cache memory hierarchy?
- What impact does the network on a chip have on data transfer performance?
- How well does each system perform with applications requiring more data than cache can hold?
- *Where is all the power going?*

To answer such questions, models may be used to better understand the system.

Power is one of the obstacles preventing Exascale performance. The Department of Energy imposed a 20MW power cap for all US based systems [53]. Titan consumes 8.2 MW of power, and Sunway consumes 15.4 MW – thus for the 1,000-fold increase in performance, power usage can at most double which is not feasible given current technology. Thus, a better understanding of power consumption is required such that future hardware can benefit. Furthermore, power usage is of great concern because typical systems require 60% of total power for cooling, and only 40% goes towards performance, and power is expensive. Thus, reducing power draw is important to reduce the long-term costs of maintaining the system.

1.2 Purpose

It is well-known that the most basic functions of a processor are performing computations, i.e., arithmetic operations, and moving data, whether it be from cache to registers, DRAM to cache, hard disk to DRAM, or DRAM to network. At some point, the

processor must perform these basic operations, and the processor must consume power to do so.

With newer generations of processors, these functions are expected to improve; both the operation of computing and moving data, and the power consumed for these operations. And with power becoming more adaptive for different components, i.e. core, DRAM, uncore, peripherals, etc., power traces will be more important to the analysis of executions because they will show more definitively how power is used for performance. This trend is already present between three generations of Xeon CPU's, as will be shown in this dissertation, and is expected to continue into Exascale computing.

Power has gained attention in the literature over the past 5 years, however it is still difficult to predict. While available models accurately capture the trend of power draw while varying cores or clock-rate, the relationship between workload and power is not modeled and must be measured. Workload, here, is not just a measure of floating-point operations because data must be moved for computation to occur, and data movement incurs significant penalties as the distance increases. Even cache misses present opportunities for inefficiency. Although, this may be hidden by allowing multiple threads to access the same core, however, this method only works when there is dedicated memory for these threads such that context switches occur with low latency. Bottom line, the relationship between power, performance, and workload needs to be better understood to improve future hardware.

Typically, power is summarized as an average. This is good when only a single value is needed to represent a complex system for comparison to other systems. However, the measurements show a distribution; and in order to better predict power draw, this distribution needs to be evaluated and modeled. The purpose of this work is to investigate power draw and

energy consumption for modern multicore and manycore platforms, whether heterogeneous or homogeneous, in order to more accurately predict power and energy consumption while varying execution characteristics, including workload.

1.3 Problem

Power draw, and thus energy consumption, is the leading limitation to Exascale performance. Although such a machine could be built in the present, the power draw of such a system would exceed 20MW, and thus be in violation of the DOE standard in place. However, more importantly, beyond 20MW a computer center will face significant costs — cooling, maintenance, etc. Thus, the problem remains, *how to achieve more efficient power and energy consumption without sacrificing performance?*

The relationship of power is well known to voltage, current, and the characteristics of interest to performance, such as clock-rate and cores, however less is known about the relationship between power and workload. There is a need for a new analytical model to describe the relationship between power and workload.

1.4 Method and Procedure

This dissertation investigates power and energy consumption on a number of hardware platforms with many different parallel applications while varying a number of execution characteristics. Homogeneous and heterogeneous executions are considered, as well as techniques for reducing energy consumption (such as dynamic voltage and frequency scaling) on CPU and Xeon Phi accelerators.

Available power and performance models are also investigated in this work, and applied to these applications and hardware combinations. A combination of these models was united to create the Execution Phase model, that models the relationship in power and performance for heterogeneous executions based on dividing the execution into computation or communication phases.

The Empirical Mode Decomposition and Hilbert-Huang Transform analysis technique is used to analyze power traces. Execution characteristics such as the number of cores, clock-rate, number of nodes, thread mapping, and device configuration are varied to capture as many different variations of execution, and therefore power draw, as available for analysis. The approach has been used to visualize power traces using the relation of energy, frequency, and time; frequency here relating to the physical system and not clock-rate (GHz). The approach has also been used to analyze segmented power measurements, and model the general trend of an execution. It is shown in this work that the EMD method is commutative, and may be applied to a sum of time-series, or individual time-series representing the same system (e.g. multiple sockets, nodes).

Probability distributions are used in this work to represent power and energy traces, thereby providing an alternative means of modeling power and energy consumption. The distribution models retain the fact that power is not constant over time, and also retains the fact that average power is an excellent approximation for most workloads and systems. Also, they may be used to define the explicit costs of a workload for a given computing platform.

The remainder of the dissertation has been separated into the following chapters. Chapter 2 presents the review of relevant literature that has motivated this work, an

introduction to the parallel applications and hardware used throughout this work, and the measurement tools required for power measurements. Chapter 3 presents the Empirical Mode Decomposition and Hilbert-Huang Transform analysis method, and the applicability of this method to power traces. Chapter 4 presents investigations on power and energy for hardware applications with Intel Xeon Phi, and discusses thread mapping strategies, applicability to DVFS for heterogeneous executions, and power limiting on the Xeon Phi. Chapter 5 presents the execution phase model where computation and data movement are modeled according to well-known power and performance models in order to predict energy for specific phases and devices. Chapter 6 presents methods for predicting energy while varying workload, and including EMD and probability distributions. Chapter 7 presents an analysis for power traces obtained on multiple sockets and nodes. Finally, Chapter 8 concludes this dissertation.

CHAPTER 2

BACKGROUND

This chapter provides the background and motivation for the remainder of this work. A review of the literature has been performed; performance, power, and energy investigations and models are reviewed. The models are then formally defined. The parallel applications, computing platforms, and measurement software and procedures used throughout this work are then introduced.

2.1 Literature Review

This section presents an analysis of literature for hardware-software modeling with a focus placed on Intel Xeon Phi co-processor and processor. This review is motivated by the following factors:

- Future systems are expected to be power constrained, which makes power capping an upper-bound on application performance,
- Future systems are susceptible to dark silicon — system resources must be turned off because of power constraints,
- Time, Power, and Energy modeling improves the understanding of hardware-software interactions, which are used to improve resource utilization, overall performance, and energy-efficiency,
- Scalability modeling is crucial for developing future Exascale systems and applications because data movement is also an upper-bound on computational performance.

2.1.1 Xeon Phi Performance, Power, and Energy Investigations

Computational throughput was the dominating performance bottleneck through the rise of Petascale computing platforms. However, as systems surpass Petascale and advanced towards Exascale performance, data movement has become an overwhelming bottleneck. The trend in computing platforms has migrated to heterogeneous computing platforms with processor and accelerators on each node. Accelerators require steep power draw requirements, but the advantage is a device with 50+ small, low clock-rate processors for highly-parallel computational workloads.

This section presents a comparison of papers investigating performance, power, and/or energy for the Intel Xeon Phi. Comparisons are made between competitive hardware (GPU and CPU where applicable), and Xeon Phi usage modes: offload, native, and symmetric. The offload usage mode uses a host + accelerator strategy, where computational tasks are "offloaded" to the Xeon Phi co-processor over the PCI bus. An application run only on the Xeon Phi is deemed "native" mode. Symmetric mode treats the CPU and Xeon Phi as separate nodes, thus MPI tasks are distributed between devices.

In most of the literature, the GPU outperforms the Xeon Phi co-processor in compute performance and energy-efficiency, as noted in: [6], [7], [37], [68], [70], and [98]. Only in the case of sparse matrix multiplication does the Xeon Phi outperform the GPU as found in [85]. Comparing the Xeon Phi to the CPU, however, most works find the Xeon Phi superior. This finding has been noted in the following 12 works: [4], [6], [7], [37], [39], [52], [69], [78], [83], [84], [85], and [98].

Although the Xeon Phi doesn't outperform the GPU for all applications, additional factors may influence users to choose the Xeon Phi over the GPU. The Xeon Phi uses the x86 architecture, which makes it compatible with x86 instruction sets – newer devices support all legacy instruction sets. Support for legacy compilers is important, since many codes have been created and maintained since the 80's and 90's; this is especially true for government projects. The GPU uses the CUDA programming model which requires code refactoring. The newer Xeon Phi ("Knights Landing") is available as a processor and is more energy-efficient than the prior generations of Xeon Phi.

Comparing Xeon Phi usage modes, the literature shows a trend towards native and offload execution. In [4], [6], [70], [83], and [108], the offload execution mode has been found to outperform the native execution mode. The opposite is found in [7], [54], [77], and [83]. Note the authors in [83] presented two evaluations of performance – one on the NASA Parallel Benchmarks (NPB) and one on the Weather Research and Forecasting real-world application. The NPB performed best under the native execution mode, and the Weather Research and Forecasting application performed best with offload.

Symmetric execution is rarely investigated ([72], [83], [78]), and has not been found to be better than native execution. Further, symmetric mode execution is sensitive to load balancing issues and data movement bottlenecks over the PCI bus which limits the usefulness of this mode. Offload execution also suffers from data movement over the PCI bus and from load balancing between the host and accelerator. Native mode execution is limited by the capabilities of a single Xeon Phi which only has 8-16GB of DRAM as a co-processor. The processor supports 16 GB of multi-channel DRAM, as well as conventional DDR3 DRAM off-chip.

In general, Xeon Phi performance depends on vectorization and cache performance, which is especially critical on this device since it only has 2 levels of cache, and each MB of L2 is shared between two cores. The smaller cache is also found on the newer Xeon Phi processor, which can be cumbersome for applications that do not optimize cache performance.

2.1.2 Modeling Multicore and Heterogeneous Computing Platforms

Aside from testing all permutations of the execution space, few methods exist to determine the optimal configuration (cores, clock-rate, etc.) for a given hardware-software combination. One such method is auto-tuning [47, 48], where many different compiled versions of a code are tested according to a search algorithm to find the best version (configuration). Although there are many flavors of auto-tuning, and often the results are very promising, there is one dominating drawback of the method: all permutations must be tested to measure performance. This leads to using time, power, and energy models for configuration space exploration. The difficulty here is that execution performance is not easily quantified into available models.

Analytical models are derived as an abstraction of the system in the form of a set of equations [71]. In this work, analytical models are further divided into the following categories: performance, energy, and scalability.

2.1.2.1 Scalability

Scalability has been an important area of research since the beginning of parallel computing. A difficult challenge in scalability is to determine the efficiency of an algorithm

on different hardware platforms, or when varying parameters such as the number of cores or problem size. Currently a method does not exist to determine the scalability of an application beyond what is measured.

Isoefficiency is a metric for measuring scalability that relates problem size to the number of cores required to maintain efficiency. Efficiency is the ratio of speedup vs the number of processors used. An introduction to the basics on isoefficiency and scalability can be found in [31, 80].

True heterogeneous models are few and far between, however the authors in [49] present an extension to Amdahl's Law to investigate the trade-off between energy and performance for heterogeneous systems. This model incorporates serial and parallel phases of execution and relative architecture complexity to compare architectures, which is an important concept in scalability.

The authors in [102] present a novel approach to determining the best configuration for energy-efficient computation for a given application and hardware pair. The work uses scalability concepts, such as speedup, concurrency, and work defined by the serial and parallel portions of the application to devise an empirical model for scalability. Although the model is excellent for selecting a best configuration for the hardware-software combination tested, the model parameters cannot be used to predict usage on another hardware platform.

Additional details on iosefficiency may be found on Georg Hager's personal blog [34] which discusses the Z-plot presented by Thomas Zeiser. In short, performance is dependent on the number of cores, clock-rate, and performance on a single core. The

findings presented here are consistent with the Execution-Cache-Model developed by Georg Hager and others, to be discussed shortly.

2.1.2.2 Performance

Performance models generally focus on defining computational throughput and/or data movement (communication) overhead as a result of parallelizing a sequential application. One of the earliest models is a communication performance model—known as "LogP"—for parallel architectures and applications which was proposed at the dawn of parallel computing [19]. LogP uses communication latency, memory transfer overhead, the reciprocal of per-processor communication bandwidth, and the number of available processor/memory modules to calculate the application performance. Following in the footsteps of the LogP model, the "roofline" model [107] has also been proposed as a general way to model parallel application runtime performance. It describes the relationship between the data movement and computational throughput, which helps to identify performance bottlenecks with respect to the theoretical performance of the hardware.

Many works investigate communication performance [74, 3, 33, 81, 92], because communication is an overhead of parallel execution. Sequential codes do not have communication because all of the data is readily available for computation, however sequential execution is too slow for real-world use. Therefore, communication overhead is a necessary penalty and these models aim to identify performance degradation due to communication. Computational throughput is linearly dependent on clock-rate, as found in [16].

This model is easily applied to any type of application (kernel, proxy-app, and real-world); however, the method is not applicable to all hardware — specifically the Xeon Phi. The time on- and off-chip model [16] requires clock-rate to be varied in order to determine the ratio of time on- and off-chip. This is useful because this ratio can be used to determine the compute- or memory-boundedness of an application without analysis of the source code. However, hardware such as the Xeon Phi does not allow user-defined control over the voltage/clock-rate states and so this model is not easily applied. This is especially true for the Knights Landing generation of Xeon Phi where clock-rate varies significantly with execution and is not controllable by the user.

2.1.2.3 Power and Energy

The models discussed thus far do not consider the combined effects of performance and power on the energy consumption of a software-hardware combination. Building upon the roofline model, [14, 15] include power and energy contributions of the parallel architecture. Introduced in [42], the roofline model has been extended to incorporate cache-memory performance in addition to data transfers between LLC and DRAM. This improvement to the model allows for more fine-grained power and performance investigations, although operational intensity must be known.

Instruction-level modeling [87] is another way for characterizing the hardware, but is not easily extended to real-world applications. Instruction-level models are very specific to a particular hardware device, in this case the Xeon Phi, but this specificity makes the model impractical for comparing hardware platforms or even analyzing large-scale

application behavior. These models are best used for theoretical foundations and benchmarking hardware performance.

The execution-cache-memory (ECM) model [35, 38] extends the Roofline model to incorporate performance degradation due to scaling clock-rate, and maintains the upper-bound on performance due to data movement. In addition, the model provides a new take on power where clock-rate has a quadratic relationship with power draw.

2.2 Time and Power Model Definitions

Several models are considered in this work: linear regression power, the Roofline model for execution time, and the ECM model for both time and power. The models are defined here for reference throughout the document.

2.2.1 Roofline

In the Roofline performance model, time is described as the maximum between computation and data movement (between DRAM and LLC), and is defined as:

$$T = \max(N_{flop} \times T_{flop},\ N_{mop} \times T_{mop}), \quad (1)$$

where T is total execution time, N_{flop} is the total number of floating-point operations, T_{flop} is the time per flop, N_{mop} is the total number of memory operations, and T_{mop} is the time per memory operation. Typically, the model is applied to micro-benchmarks which are custom built for a hardware architecture to stress-test performance. Applying the model to a larger application can be more difficult if the number of FLOPs and MOPs is not well defined.

2.2.2 Linear Power Model

The power model assumes a linear relationship between workload, cores, clock-rate, and power draw, and is defined as

$$P = P_{static} + kcf^3 \tag{2}$$

where P is the total power, P_{static} is the static power draw, and dynamic power is defined by the workload constant k, number of cores c, and clock-rate f . Clock-rate is cubed because power is proportional the product of dynamic capacitance, voltage squared, and clock-rate; however some assumptions can be made. The influence of voltage and clock-rate on power draw are proportional, hence power is defined as clock-rate cubed [111] and dynamic capacitance is factored into the workload constant k in the linear model.

2.2.3 Execution-Cache-Memory

The Execution-Cache-Memory (ECM) energy model is defined using novel performance and power models. The time model assumes a linear relationship between performance exists between floating-point operations, cores and clock-rate defined as:

$$T = \min\left(cN_0\left(1 + \frac{f_0 - f}{f_0}\right), N_{max}\right)^{-1}, \tag{3}$$

where N_0 is the performance in FLOPs for one core, N_{max} is the maximum achievable performance given all bottlenecks, c is the number of cores, f is the current clock-rate, and f_0 is the baseline clock-rate.

The power model assumes a quadratic relationship to clock-rate, a linear relationship to the number of cores (independent of static power), and is defined as:

$$P = W_0 + c(W_1 f + W_2 f^2), \tag{4}$$

where W0 is static power draw, W1 is the coefficient for the linear term of power draw, and W2 is the quadratic term. Dynamic power, defined by the linear and quadratic terms, scales with the number of cores.

2.3 Parallel Applications

Parallel applications are commonly used in academic, government, and industry research. Each application requires specific resources which varies how the software utilizes the hardware, therefore it is of interest to test many different applications to better understand hardware power draw. Below, the following parallel applications are introduced: GAMESS, CoMD, and NPB.

2.3.1 GAMESS

The General Atomic and Molecular Electronic Structure System (GAMESS) [30, 86] is a widely used quantum chemistry package capable of performing molecular structure and property calculations by a rich variety of ab initio methods finding an (approximate) solution of the Schrödinger equation for a given molecular system. An approximate (uncorrelated) solution is initially found using the Hartree-Fock (HF) method via an iterative self-consistent field (SCF) approach, and then is improved using various electron-correlated methods, such as second-order Møller-Plesset perturbation theory (MP2).

To reduce the computational complexity for large molecular systems, a fragmentation approach, such as Fragment Molecular Orbital (FMO) method [29], is used, which divides the system into fragments and applies a quantum chemical method to each

fragment, followed by the consideration of fragment interactions. The inputs used in this work are calculated using the MP2 method. Specifically, they are *20w*, a cluster of 20 water molecules; *1L2Y*, a synthetic protein tryptophan cage; *S256*, a 1-trichloromethylsilatrane (TCMS) molecule with 6-31G(d) basis set (265 basis functions), and *S301*, a TCMS molecule with 6-31G(d,p) basis set (301 basis functions). The inputs 20w and 1L2Y also use FMO approximations of short-range interactions up to trimers (when triples of fragments considered as a single fragment). OpenMP is not available in GAMESS, so half of the total MPI (Message Passing Interface) tasks are dedicated to computation and the remaining half to data movement via the generalized Distributed Data Interface (GDDI) [27].

2.3.2 CoMD

Co-design Molecular Dynamics (CoMD) is a proxy application developed as part of the Department of Energy co-design research effort [22] at the Extreme Materials at Extreme Scale (ExMatEx) center. CoMD is compute-intensive, where approximately 85– 90% of the execution time is spent computing forces. In this work, both force kernels are used: the more accurate Embedded Atom Model (EAM) force kernel for short-range material response simulations, such as uncharged metallic materials [23], and the less accurate Lennard-Jones (LJ) force kernel. The LJ force kernel consists of one compute loop, whereas EAM consists of three compute loops and a small halo data exchange between the second and third loop.

Problem size is expressed as the number of atoms along an axis of the material; the default material is copper. In this work, each axis is equivalent (in atoms) which defines

the material shape is a cube. A problem size of 40 equates to $4 \times 40^3 = 256{,}000$ atoms. CoMD is available as a hybrid of MPI and OpenMP, thus each may be measured separately or in combination.

2.3.3 NPB

The NAS Parallel Benchmarks [76] is a collection of programs used to evaluate the performance of parallel supercomputers, which was derived from computational fluid dynamics applications. This work considers all its five kernels: EP (embarrassingly parallel), CG (conjugate gradient), FT (discrete 3D fast Fourier Transform), IS (integer sort), and MG (multi-grid solver on a sequence of meshes). Note that EP is compute-intensive, CG and MG are memory-intensive (see [96]), IS uses random memory access patterns, and FT performs all-to-all communication. Additionally, four pseudo-applications have been tested: BT (block tri-diagonal solver), SP (scalar penta-diagonal solver), and LU (lower-upper Gauss-Seidel solver), and UA (an unstructured adaptive mesh which imposes dynamic and irregular memory accesses). The NPB applications are available for MPI or OpenMP, although few are also offered as a hybrid. Problem sizes are defined by “class”, ranging from S, W, A, B, C, D, and E as specified in [75].

2.4 Computing Platforms

The computing platforms used in this work are organized as follows: Borges, Bolt, Turing, Marquez, and Rulfo. The computing architectures include Intel Sandy-Bride, Ivy-Bridge, and Haswell CPU’s and Intel Xeon Phi KNC and KNL generations. The CPU hardware specifications are provided in Table I for Borges, Bolt, Turing, and Marquez.

TABLE I

HARDWARE CHARACTERISTICS OF THE PARALLEL COMPUTING PLATFORMS

	Borges	Bolt	Turing	Marquez
Microarchitecture	Sandy	Sandy	Ivy	Haswell
Model	E5-2650	E5-1650	E5-2670 v2	E5-2630 v3
Nodes	1	3	10	1
Sockets (p Node)	2	1	2	2
Cores (p Socket)	8	6	10	8
Clock-Rate (GHz)	2.0-1.2	3.2-1.2	2.5-1.2	2.4-1.2
LL Cache (MB)	32	12	25	20.5
DRAM (GB)	64	64	64	64
TDP (Watts)	95	130	115	85
Location	ODU	Ames Lab	ODU - HPC	ODU

TABLE II

INTEL XEON PHI HARDWARE SPECIFICATIONS FOR KNC AND KNL

	Knights-Corner	Knights-Landing
Device	Co-processor	Processor
Model	5110p	7210
Cores	60	64
Threads (p Core)	4	4
Clock-Rate (GHz)	1.053	1.3-1.0
LL Cache (MB)	30	32
DRAM (GB)	8	16
VPU (bits)	512	512 x2
FMA (ops/cyc)	2	2
TDP (Watts)	245	215

The Intel Xeon Phi hardware specifications are provided in Table II. The Borges, Bolt, and Turing systems are equipped with 2 KNC per node, in addition to the CPU specified in Table I. For Bolt, the compute nodes are QDR-connected with Infiniband. For Turing, nodes are FDR-connected with Infiniband. To avoid confusion with the EMD/HHT analysis method, which calculates a physical frequency, clock-rate is used to reference the operating frequency of any hardware platform in this work.

2.5 Power Measurement

The Sandia National Labs PowerAPI [55] is used to measure energy via the Linux Power Capping Framework (LPCF) [2] plugin which reads energy from the Running Average Power Limit (RAPL) [103, 20] counters. The PowerAPI uses the hardware locality (hwloc) API [79, 9] to detect the underlying hardware and is very portable.

Power measurements are collected every 5ms, and measurements are collected for five seconds before and after the application is executed to establish the idle power draw. Although most of computing platforms support up to 1ms resolution for sampling power using RAPL, it has been found empirically that 1ms sampling is unreliable on most systems. A more modest 5ms sampling rate is used in this work; however, the sampling rate is closer to 10ms when the system is loaded. Sampling rate is reported as 5ms throughout this work. In the case of the KNC, a 20ms sampling rate is used because this is the minimum achievable sampling rate for the device (and RAPL is not available). Power is read using the micmgmt API or by reading the power file hosted on the device at: `/sys/class/micras/power`. KNL supports RAPL and the LPCF, and is compatible with the PowerAPI.

For the Intel Xeon Phi, user-defined clock-rate scaling is not available; instead, the clock-rate may be changed indirectly by setting power limit thresholds for the device. The Xeon Phi System Management Controller (SMC) varies operating clock-rate as power surpasses the designated thresholds. Specifically, the Xeon Phi uses two power threshold values—low and high—each with a designated time window. By default, the low power threshold is set to the TDP with a time window of 100ms and the high threshold at 120% of the TDP and a time window of 10ms. When power exceeds the low threshold for the duration of the time window, clock-rate is decreased until power consumption is less than that of the threshold. When power exceeds the high threshold for the duration of the time window, the thermal throttling mechanism is engaged, which forces the device to the lowest operating clock-rate of around 500 MHz, as seen experimentally. More on Xeon Phi power limiting can be found in the datasheet [43].

CHAPTER 3

EMPIRICAL MODE DECOMPOSITION

The Empirical Mode Decomposition and Hilbert-Huang Transform (EMD/HHT) method [40, 109] is used for non-parametric non-stationary time-series analysis and calculates instantaneous amplitude and frequency, and is applied to real-world systems to uncover underlying physical interactions. This method has been already successfully applied in a variety of fields, such as medicine, finance, engineering, and more recently in geosciences — analysis of sea level data [26] and climate change studies [25]. The main advantage of EMD/HHT over standard spectral methods is that it detects oscillating modes with time-dependent amplitudes and frequencies, so it is useful for analyzing irregular data with unknown frequencies. On the other hand, the interpretation of the EMD/HHT results is not straightforward since individual modes do not necessarily represent particular execution characteristics.

EMD/HHT has been adopted to analyze an execution as a whole as opposed to its division into phases based on specific resources used in each phase. Phase refers to a computation or data-movement type operation, such as RAM to cache data transfers or communication on the node or over the network; the phases often overlap to optimize performance. Such a division was considered in [66] in order to model each phase differently, which has proven to be difficult in general for correlating phases with power readings. See Chapter 5 for additional details on the phase method.

3.1 Method

EMD is used to decompose a power trace into oscillating intrinsic mode functions

(IMF) and a residual trend. An IMF is a function that satisfies two criteria [40]. First, the number of extrema and number of zero-crossings must be equal or differ by no more than one. Second, the mean value of the envelope defined by the local maxima and minima is zero. IMF's are recovered from the time-series until none remain — the resulting time-series is the residual trend. This may be described as:

$$h(t) = \sum_{i=1}^{N} c_i(t) + r(t) \tag{5}$$

where $h(t)$ is the original time-series, in this work the power trace, $c_i(t)$ is the i-th IMF of a total of N IMF's, and $r(t)$ is the residual.

EMD extracts IMFs through a process called sifting. To sift, the minimum and maximum extrema of the time-series are used to calculate the average; the difference between the average and time-series is then treated as the time-series for the next sift. This process continuously refines the data set until the standard deviation of the resulting time-series is less than 0.2 (see [40]). Once this standard deviation is obtained the resulting time-series is accepted as an IMF and is subsequently removed from the original time-series. This process is repeated until the residual is found from which no other IMFs may be obtained. One potential use for the residual trend is to construct a non-linear model to relate power and time-to-solution, as proposed by the authors [60]. Note that the total number of IMFs is an output of EMD and depends on the trace characteristics. For instance, more IMF modes are found in longer traces because low-frequency oscillations are more likely to be detected.

HHT is then applied to each IMF, except the residual, to calculate instantaneous frequency: the time derivative of the oscillation phase for any time-step of the signal [40].

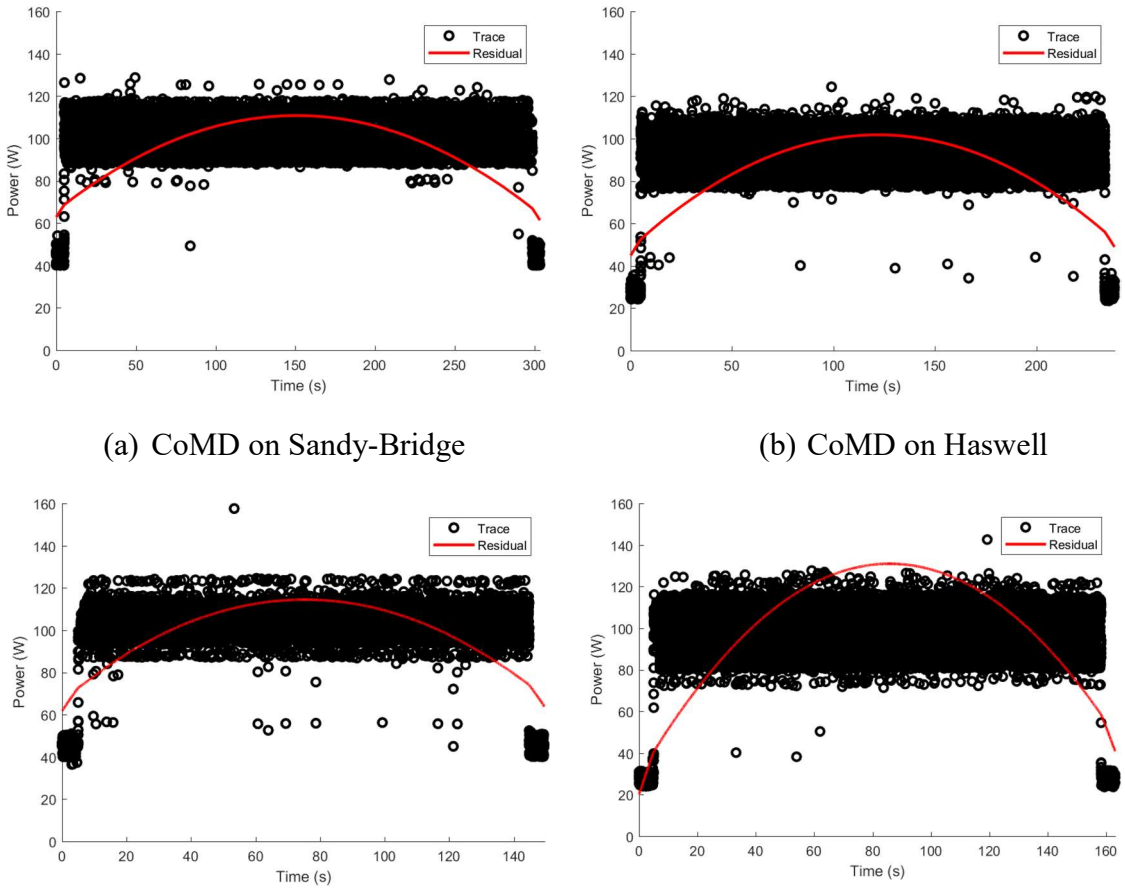
The maximum frequency that may be obtained using HHT is determined by the sampling rate r in the expression $1/5r$, where 5 is the minimum number of data points required to accurately define instantaneous frequency [40]. Modern HPC systems are able to sample power at a maximum rate of 1ms, but to ensure consistency between measurements across computing platforms, a more modest sampling rate of 5ms is used throughout this work. At 5ms, the maximum frequency obtainable by EMD is 40Hz. Sampling rate significantly impacts the utility of the EMD/HHT method.

The implementation of the EMD/HHT method used here is based on the original one from [40, 109], as adapted in [25, 26]. Source code for EMD/HHT analysis in Matlab is available at [24]. A talk given by Donghoh Kim is an excellent aid for understanding the EMD procedure, see [51].

3.2 EMD/HHT on Power Traces

Figure 1 presents power traces for CoMD and CG executed on the Borges and Marquez computing platforms, Sandy-bridge and Haswell respectively. Power samples are shown as black circles and the residual trend of the trace is shown as a solid red curve. The residual has been obtained using the EMD procedure above and will be of importance later in this chapter. The traces include measurements for idle (static) power draw and active (dynamic) power draw, and some comparisons between systems and applications may be established.

First note that static power for the Sandy-Bridge system is between 40-50W, whereas the Haswell system is between 20-30W. Already the Haswell system has an energy advantage over Sandy. Notice also that CoMD runs faster on Haswell than Sandy,



(a) CoMD on Sandy-Bridge

(b) CoMD on Haswell

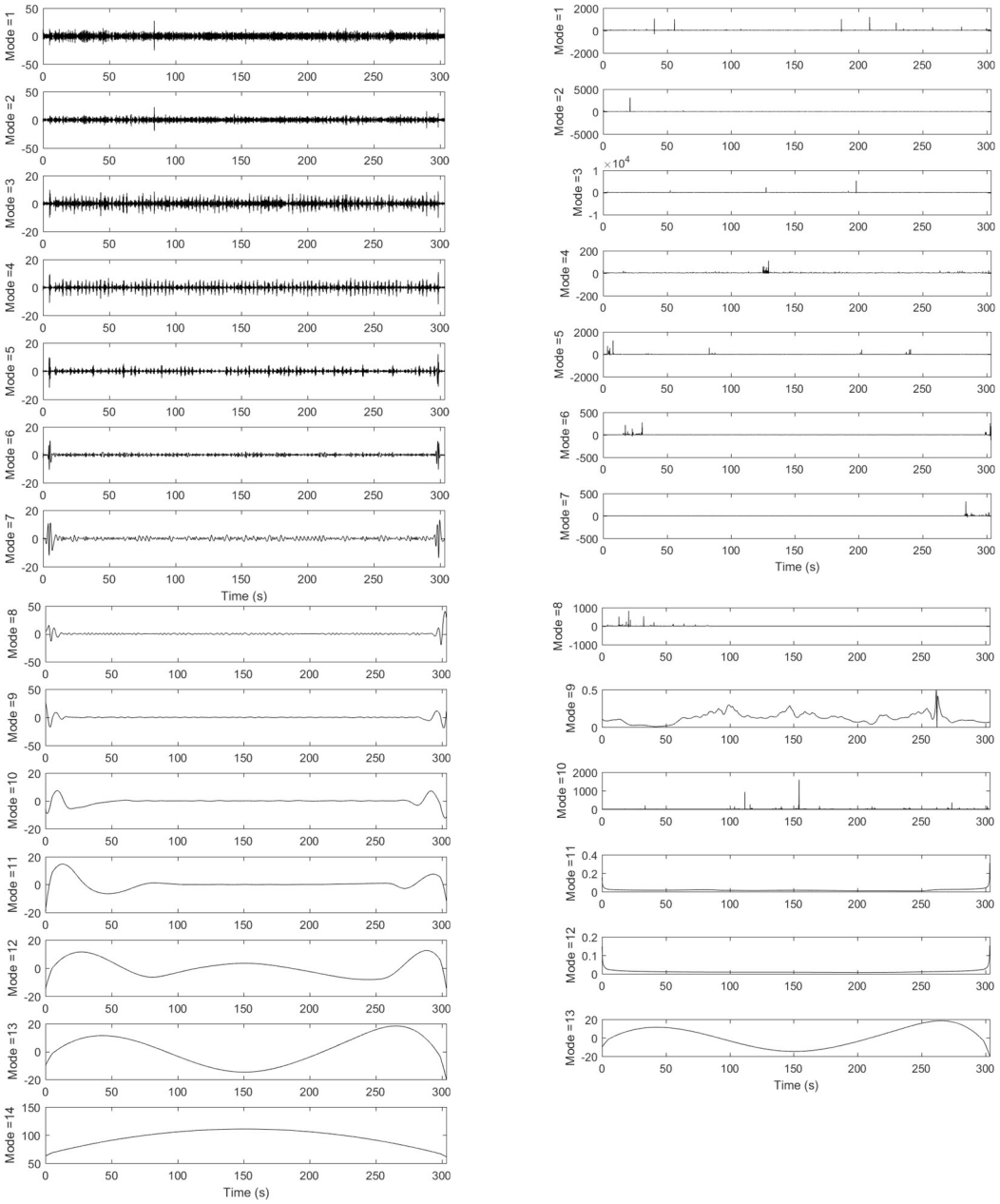
(c) CG on Sandy-Bridge

(d) CG on Haswell

Fig. 1. Original power traces with EMD residual for CoMD and CG on Sandy-Bridge (Borges) and Haswell (Marquez) computing platforms.

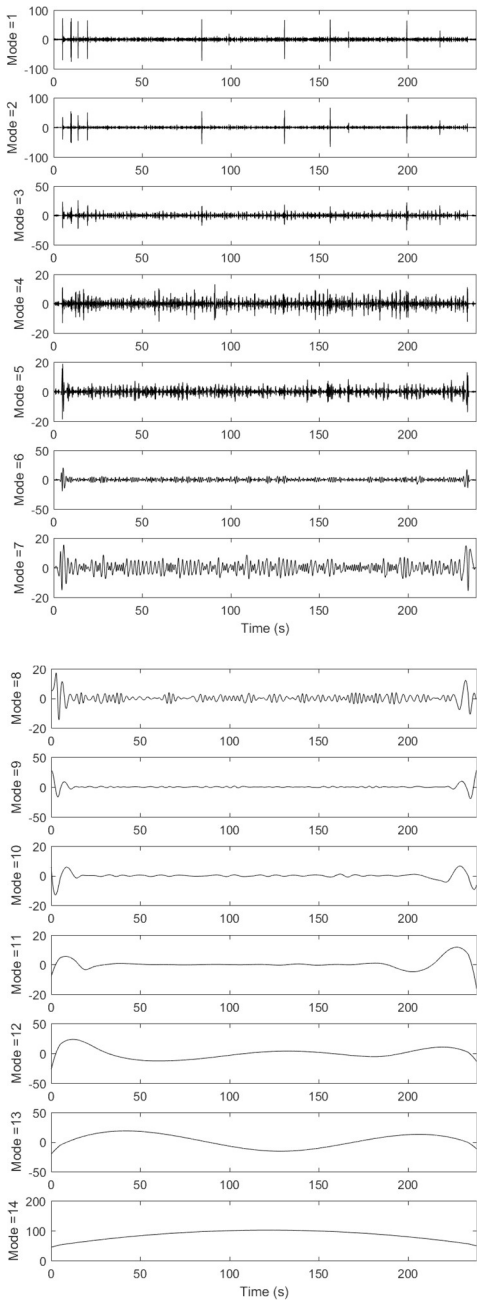
but CG runs faster on Sandy. Interestingly, all applications use roughly 80-120W while active and power samples are observed over the range with few outliers. Outliers near idle power draw show moments when the execution encountered an idle period, possibly due to a severe latency penalty in data movement.

The power traces in Fig. 1 serve as input to the Empirical Mode Decomposition (EMD) and Hilbert-Huang Transform (HHT) analysis method. Figure 2 presents the

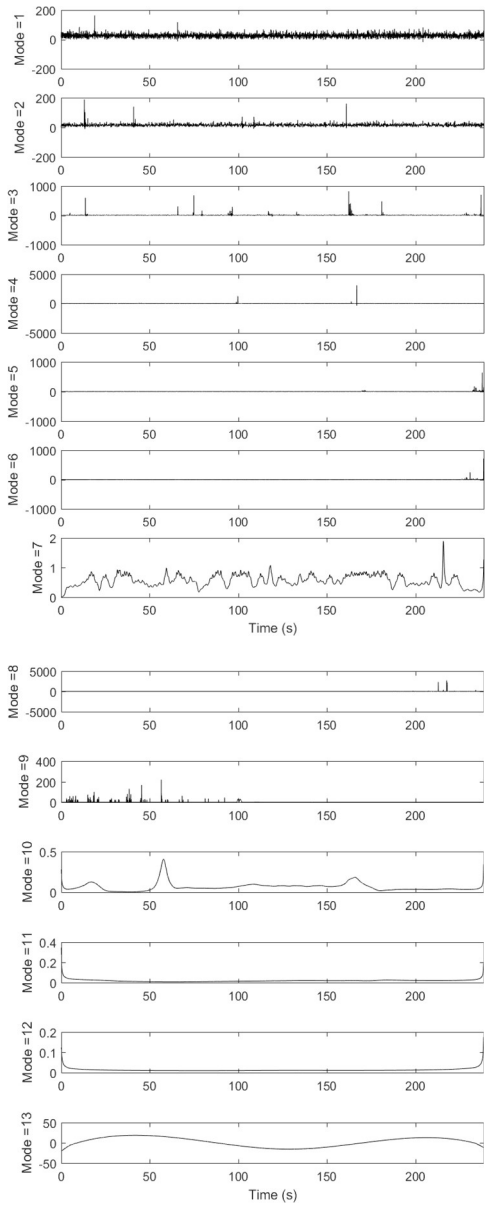


Amplitude for CoMD on Sandy-Bridge Frequency for CoMD on Sandy-Bridge

Fig. 2. Intrinsic mode functions for CoMD on the Borges platform. The power trace decomposed into 13 modes and the residual trend, with amplitude (Watts) on the left and frequency (Hertz) on the right.

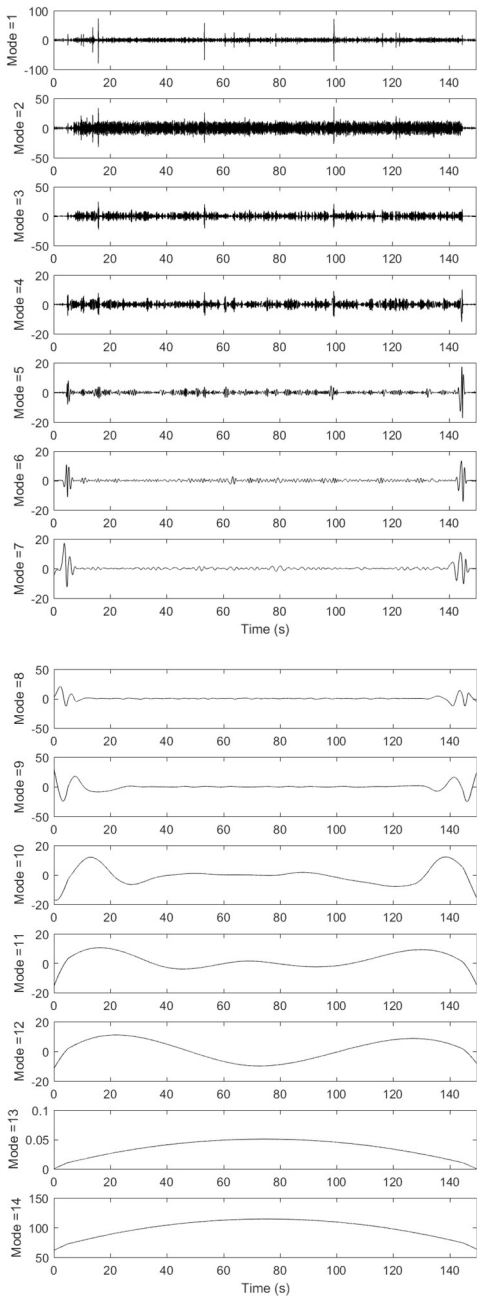


Amplitude for CoMD on Haswell

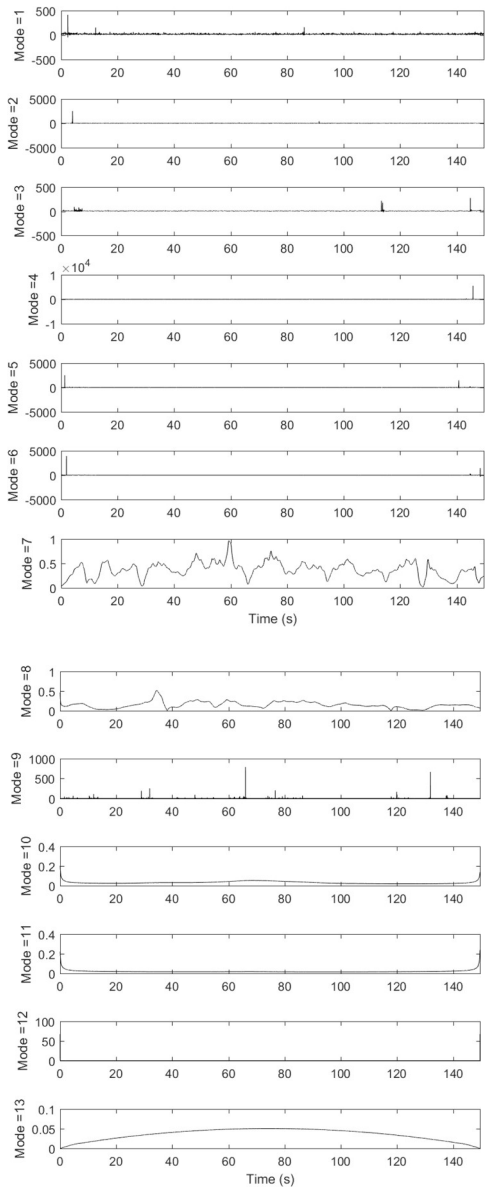


Frequency for CoMD on Haswell

Fig. 3. Intrinsic mode functions for CoMD on the Marquez platform. The power trace decomposed into 13 modes and the residual trend, with amplitude (Watts) on the left and frequency (Hertz) on the right.

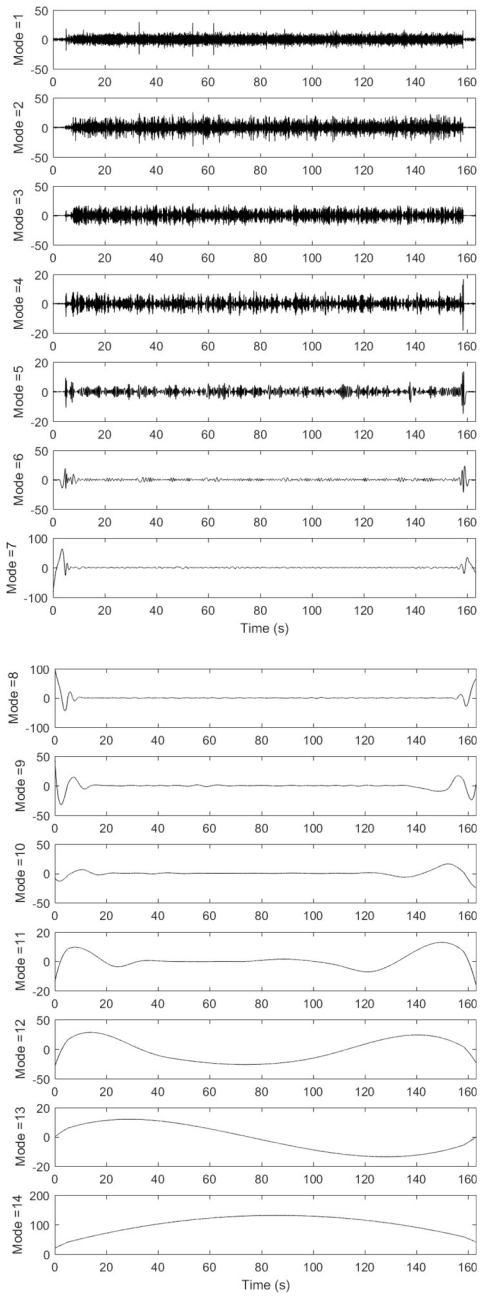


Amplitude for CG on Sandy-Bridge

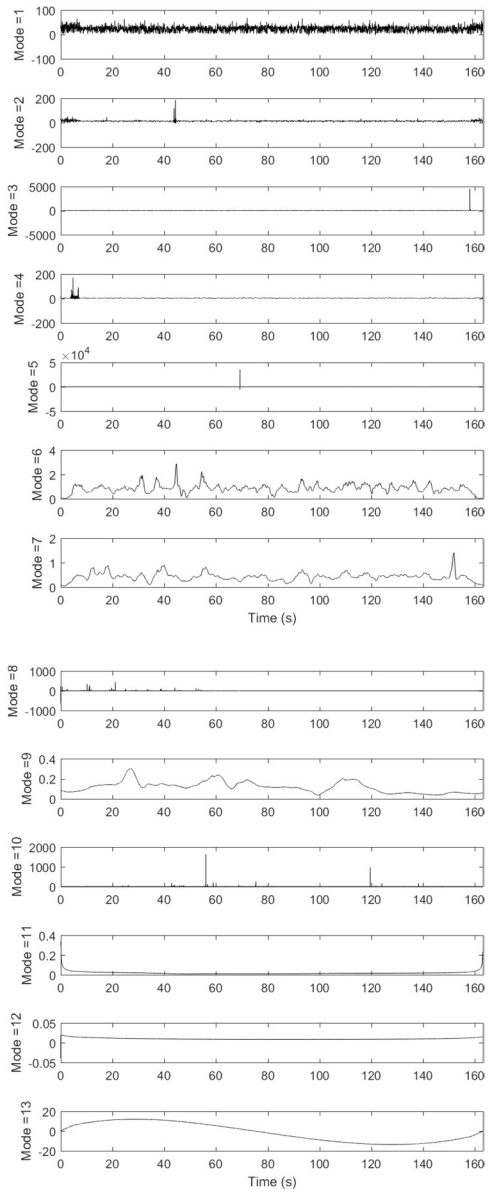


Frequency for CG on Sandy-Bridge

Fig. 4. Intrinsic mode functions for CG on the Borges platform. The power trace decomposed into 13 modes and the residual trend, with amplitude (Watts) on the left and frequency (Hertz) on the right.



Amplitude for CG on Haswell



Frequency for CG on Haswell

Fig. 5. Intrinsic mode functions for CG on the Marquez platform. The power trace decomposed into 13 modes and the residual trend, with amplitude (Watts) on the left and frequency (Hertz) on the right.

intrinsic mode functions for the power trace of CoMD on Borges, shown in Fig. 1a. Figure 3 presents the IMF's for CoMD on Marquez, shown in Fig. 1b. Figure 4 presents the IMF's for CG on Borges, shown in Fig. 1c. Figure 5 presents the IMF's for CG on Marquez, shown in Fig. 1d.

The sifting process removes the highest frequency oscillations from the source time-series first, hence the low modes of amplitude correspond to the high-frequency oscillations and as the number of modes increases the frequency of oscillations decreases. This pattern may be observed in the amplitude of IMF's for all four traces. Note here that the frequency of oscillations in amplitude, as described here, is different from the instantaneous frequency calculated using HHT. Instantaneous frequency describes the frequency of the system, in this case the execution (application running on a computing platform).

By observing the patterns in the amplitude of the IMF's, some features become visible. Most obvious is the start and end of execution which can be identified in most of the IMF modes for any given trace. In the lower modes, the high-frequency oscillations do not begin until execution starts which is to be expected. The hardware platform is idle until a workload is executed on the system, and this is reflected well by EMD.

Beyond identifying the bounds of execution with respect to the trace, it is difficult to discern much from the IMFs. It is evident that the IMF's relate to workload and performance characteristics, but it is not immediately discernible how the IMF's relate to computation or data movement. Consider IMF modes 1-7 for each of the four traces and observe the frequency of oscillations in amplitude while varying hardware platforms and applications. There are different patterns corresponding to different applications/platforms

which likely correlate computation or data movement, since these are the most basic workloads of any application. The difficulty is in establishing the correlation, since it is not feasible to instrument a large-scale application with output markers to identify computation and communication phases.

Due to the difference in power sampling rate and clock-rate, it is difficult to relate instantaneous frequency to the power trace. However, certain modes show interesting results; consider modes 6, 7, & 9 for CG on the Haswell platform (Fig. 5). Similar patterns in frequency may be observed in the other traces.

3.3 Ensemble Empirical Mode Decomposition

It may happen, however, that an intermittent mode cannot manifest under the standard deviation constraint and "contaminates" the residual trend with a spurious IMF. To alleviate this problem and obtain a more reliable shape of the residual, the Ensemble EMD (EEMD) method [109] may be applied.

EEMD works by introducing white noise to the time-series to exhaust the sifting process. While in EMD, the sifting processes the original time-series once to extract each IMF, in EEMD, white noise and the sifting process are applied to the time-series multiple times, such that the white noise is averaged out and only the trace itself remains. As explained in [109], IMFs obtained from different series of white noise have no correlation with each other, and therefore the means of each IMF (of white noise) will cancel out. This way, EEMD may avoid the residual contamination, as seen, e.g., in Fig. 6, which illustrates the difference between residuals found using EMD and EEMD.

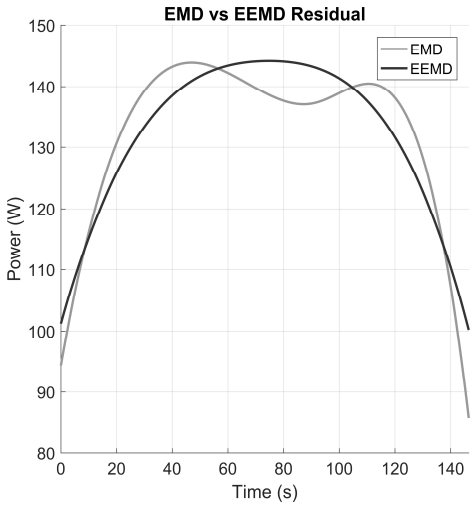
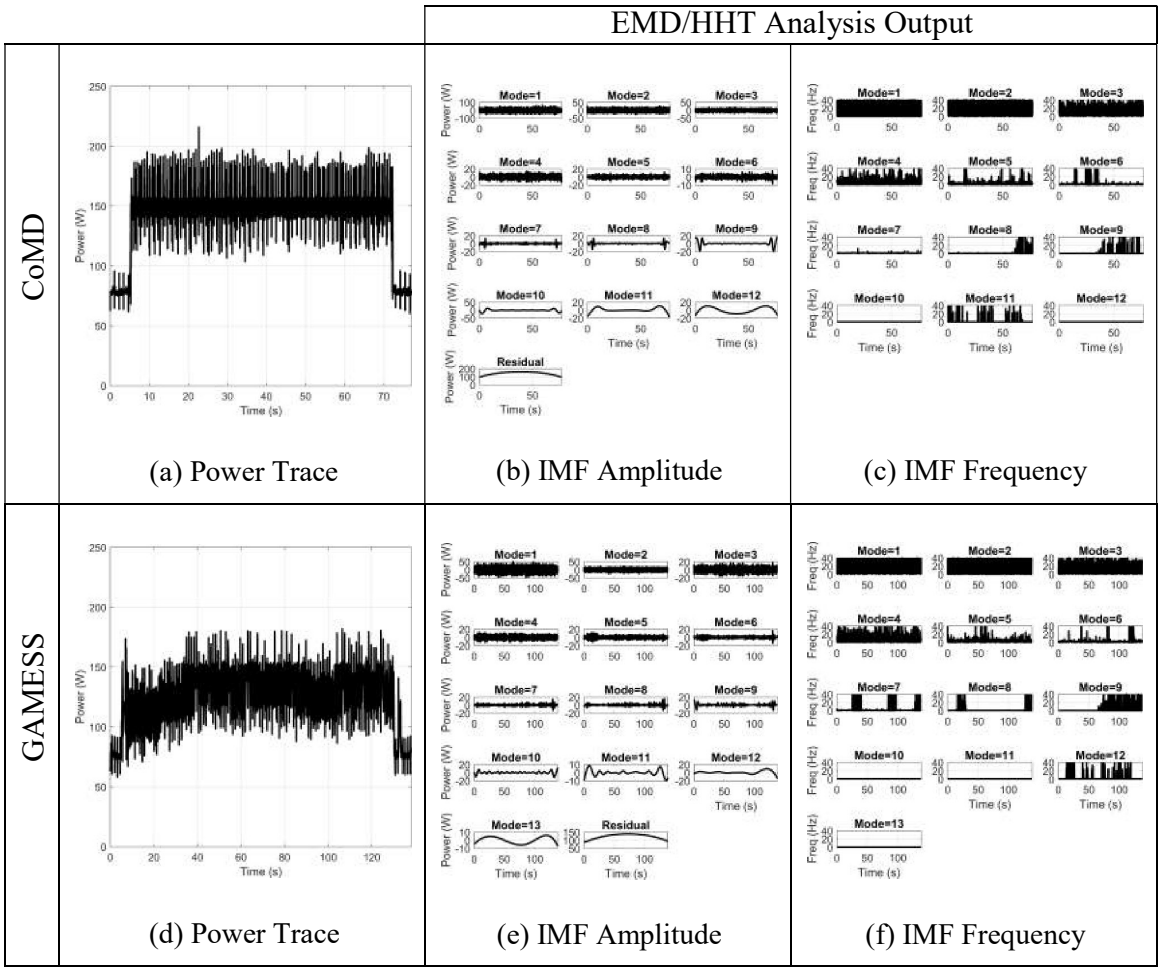


Fig. 6. Illustration of a residual with intermittent oscillations found using EMD and the same residual with EEMD (5 Watts, 100 Iterations).

It is important to note that the number of modes produced by EEMD and EMD are the same for the traces explored in this work and possibly in general. The difference between the EEMD and EMD is in the shape of the resulting IMFs and residual, where modes now include intermittent oscillations otherwise missed. The error introduced by EEMD can be calculated as:

$$\varepsilon = \frac{\sqrt{\sigma_w}}{N} \tag{6}$$

where ε is the standard deviation of error introduced to by the white noise, σ_w is the specified amplitude of white noise, and N is the specified number of iterations [109]. Based on this equation, applying EEMD with 5 Watts and 100 iterations yields a standard deviation of error of 0.5. Indeed, this error is high but has been shown to improve the result of EEMD as shown in Fig. 6 and in [60].



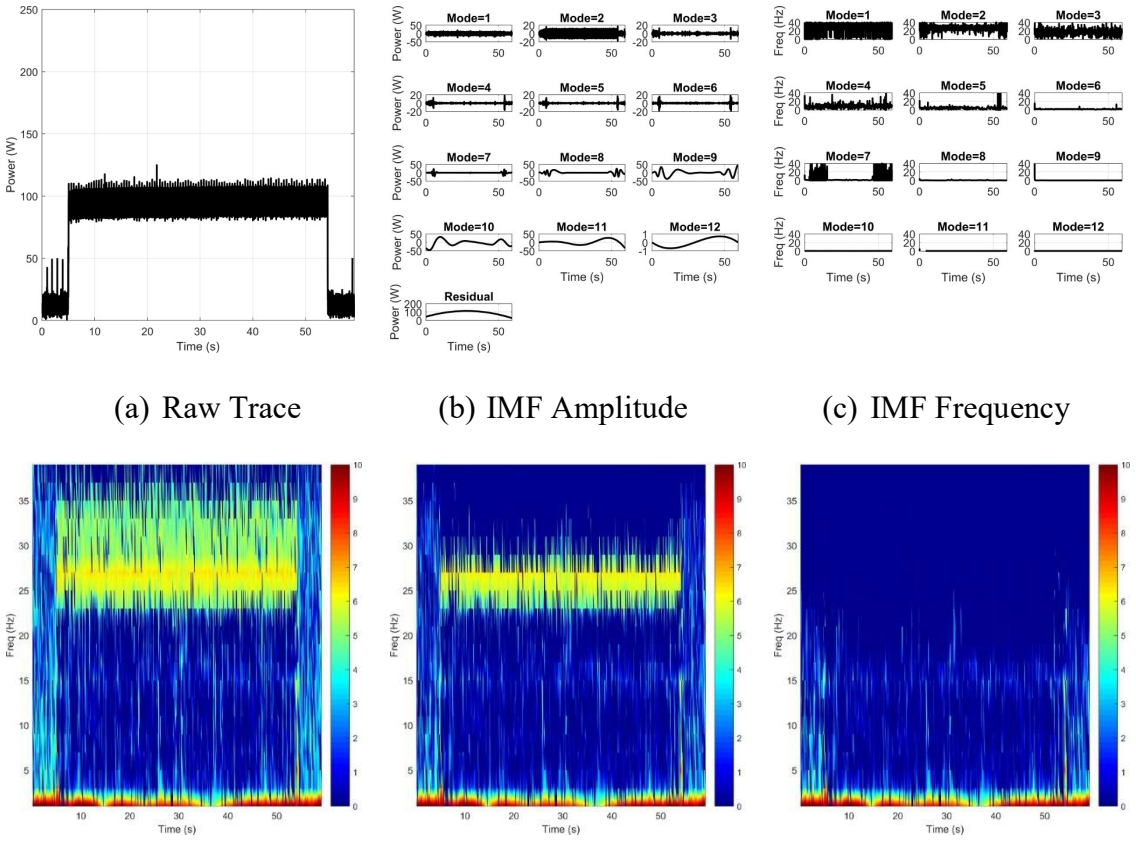
(a) Power Trace (b) IMF Amplitude (c) IMF Frequency

(d) Power Trace (e) IMF Amplitude (f) IMF Frequency

Fig. 7. Illustration of EMD/HHT procedure on CoMD (top row) and GAMESS (bottom row). The original power trace (a & d) is decomposed into intrinsic mode functions (IMFs) with respect to amplitude (b & e) and instantaneous frequency (c & f). Each trace is collected while executing CoMD or GAMESS with 59 cores at the maximum computer clock- rate.

3.4 Energy-Frequency-Time

Consider for this section the following power traces, IMF amplitudes and frequencies shown in Fig. 7. Once a power trace has been analyzed using EMD/HHT



(a) Raw Trace (b) IMF Amplitude (c) IMF Frequency

(d) All IMF Modes (e) Mode 1 Removed (f) Modes 1 & 2 Removed

Fig. 8. Illustration of EMD/HHT histograms generated using a power trace (a) collected on Bolt-CPU running CoMD-50 with maximum cores and clock-rate. The EMD/HHT analysis produced IMFs shown as amplitudes (b) and frequencies (c), which were then used to generate histograms (c–e); Histogram (c) was created with all available IMF modes, (d) all modes minus mode 1, and (e) all modes minus modes 1 and 2.

(see Fig. 7), the amplitude and instantaneous frequency may be combined into a 2-dimensional histogram. Time and frequency make up the x- and y-axes, respectively, and amplitude is collected in bins and represented as intensity using color from blue to red for low to high, respectively. Hence, intensity is the sum of all amplitudes for a given

time/frequency bin. Intensity is used to show the concentration of power draw with respect to time and frequency. The histogram uses bin sizes of 100ms (time) and 2Hz (frequency). A feature of these histograms is a band, which is a range of frequencies having a consistent intensity throughout execution.

Figure 8 presents a power trace collected on the Bolt system while running CoMD on the CPU with maximum cores and clock-rate for a problem size of 50 (500,000 atoms). The power trace (a) has been analyzed using EMD/HHT to produce IMFs (b and c), which were then combined to form the 2D histograms (d–f), of time and frequency, where intensity is the sum of all amplitudes for a given time/frequency bin. To better understand the histogram, consider Fig. 8d to Fig. 8f. In Fig. 8d, where all the IMF modes are included, notice the moderate-to-high intensity (in yellow) from 24 to 36Hz. In Fig. 8e, which is the same as Fig. 8d but without mode #1, the yellow band of moderate intensity has shrunk and only encompasses 24–30Hz. Therefore, one may conclude that the first mode contains high frequency oscillations from the original trace (in Fig. 8a).

One step further, in Fig. 8f, the band of moderate intensity has vanished. Comparing with the IMF data shown in Fig. 8b and Fig. 8c, it is now more apparent that the “high-frequency” modes (modes 1 and 2) contain a large portion of the total power draw for CoMD. Similarly, for GAMESS, modes 1, 2, and 3 contribute the most to total power draw (see Fig. 7e). Hence, in this way, it is possible to quantify a significant amount of power is used by high-frequency interactions. It is also of importance to note that the highest intensity is shown at frequency close to zero (see Fig. 8d), which can be explained by static power draw or low-frequency operations, such as data I/O.

Figure 9 presents the EMD/HHT histograms generated for power traces collected

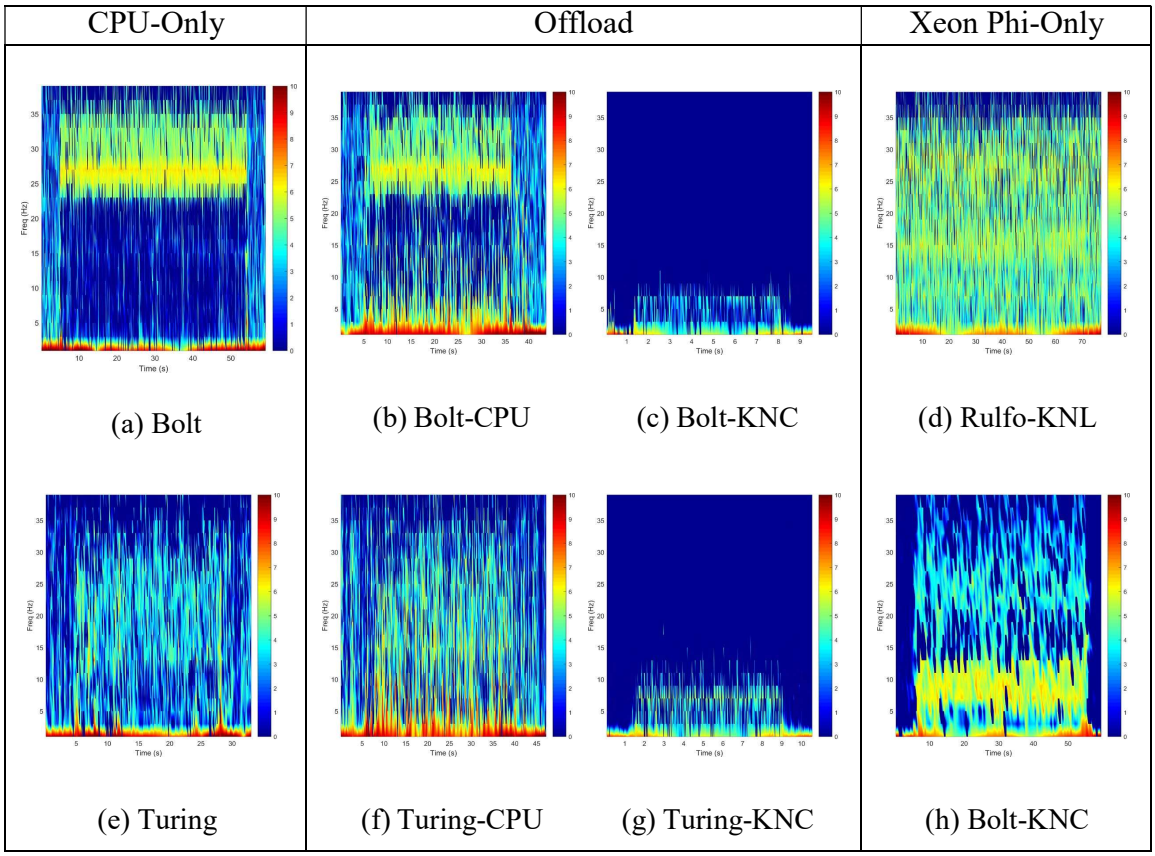


Fig. 9. Comparison of EMD/HHT histograms generated for power traces collected by running CoMD on different systems and for different usage modes. From left to right, the first column presents the histogram on the Bolt (a) and Turing (e) systems. The following two columns present the offload histograms, with the CPU output on the left and KNC output on the right for Bolt (b & f) and Turing (c & g). The final column presents the histograms for the two Xeon Phi systems, KNL on Rulfo (d) and KNC on Bolt (h).

by running CoMD on different systems and for different usage modes. From left to right, the first column presents the histogram on the Bolt (Fig. 9a) and Turing (Fig. 9e) systems. The following two columns present the offload histograms, with the CPU output on the left and KNC output on the right for Bolt (Fig. 9b & Fig. 9c) and Turing (Fig. 9f & Fig. 9g).

The final column presents the histograms for the two Xeon Phi systems, KNL on Rulfo (Fig. 9d) and KNC on Bolt (Fig. 9h). Comparisons of the histograms in Fig. 9a and Fig. 9e provide insights on how the different hardware platforms respond to a similar workload—CoMD on the CPU with maximum cores and clock-rate for a problem size of 50. The histogram for Bolt (Fig. 9a) shows a concentrated band of moderate-to-high intensity (yellow) above 24Hz, suggesting that the hardware is approaching performance bottlenecks. Specifically, an operation that occurs at 28Hz causes high intensity throughout the Bolt execution and may be indicative of a performance bottleneck. Turing (Fig. 9e), on the other hand, shows a moderate-to-low intensity (cyan) throughout execution, and this intensity band spans the entire spectrum from 0 to 40Hz. From such comparisons, it may be deduced that a more consistent intensity over frequency and time suggests the application performs more optimally. Indeed, Turing is able to solve the problem almost twice as fast as Bolt thanks to having increased parallelism of 20 cores versus 6 cores on Bolt.

By comparing Bolt and Turing, and the CPU and Offload usage modes, the following findings may be observed. Comparing CPU executions (Fig. 9a vs Fig. 9b) and (Fig. 9e vs Fig. 9f), data transfer over the PCI bus can be observed. This is the critical difference between CPU-only and Offload usage modes, since data must be shared between the host CPU and KNC devices. In particular, an increase in intensity is found for frequencies below 10Hz throughout execution. Data transfer over the PCI bus is a form of I/O, which is considered low-frequency because data is often transferred in large chunks that experience varying degrees of performance. High-frequency data transfers include RAM and cache memory because these subsystems operate more frequently than PCI bus

transfers. The KNC histograms (Fig. 9c and Fig. 9g) also provide insights with the frequency limit of 10Hz. The low intensity on Bolt suggests the KNC device was prone to latency due to load balance problems between the CPU and KNC. Bolt suffers from a lack of parallelism, whereas Turing can achieve better load balancing due to the increased parallelism. Note that obtaining frequencies above 10Hz, as in Fig. 9h, for a sampling rate of 20ms suggests that the sampling resolution is not sufficient for EMD/HHT analysis. Figure 9d presents another example of an optimal execution performance, as is explained further using Fig. 10.

Figure 10 presents the EMD/HHT histograms generated for power traces collected by running CoMD and GAMESS on different systems while varying the number of cores or clock-rate. From left to right, the first two columns present the histograms on the Rulfo for CoMD and GAMESS with 63 cores (Fig. 10a & Fig. 10b) and with 32 cores (Fig. 10d & Fig. 10e). The final column presents the histograms for the Turing system with maximum clock-rate (Fig. 10c) and minimum clock-rate (Fig. 10f). Consider two numbers of cores, 63 and 32, as shown for CoMD in Fig. 10a and Fig. 10d, and for GAMESS, in Fig. 10b and Fig. 10e, respectively. For the smaller number of cores, the intensity of the trace decreased over the entire time-frequency domain. Although this an expected behavior, the histograms are telling because they show that the processor power draw impacts at all frequencies.

In particular, CoMD is a compute-intensive application that achieves optimal performance with the maximum number of cores. The intensity for the maximum number of cores is moderate, and for the minimum number of cores the intensity is moderate-to-low; factoring time-to-solution with this difference, it is apparent that a moderate intensity

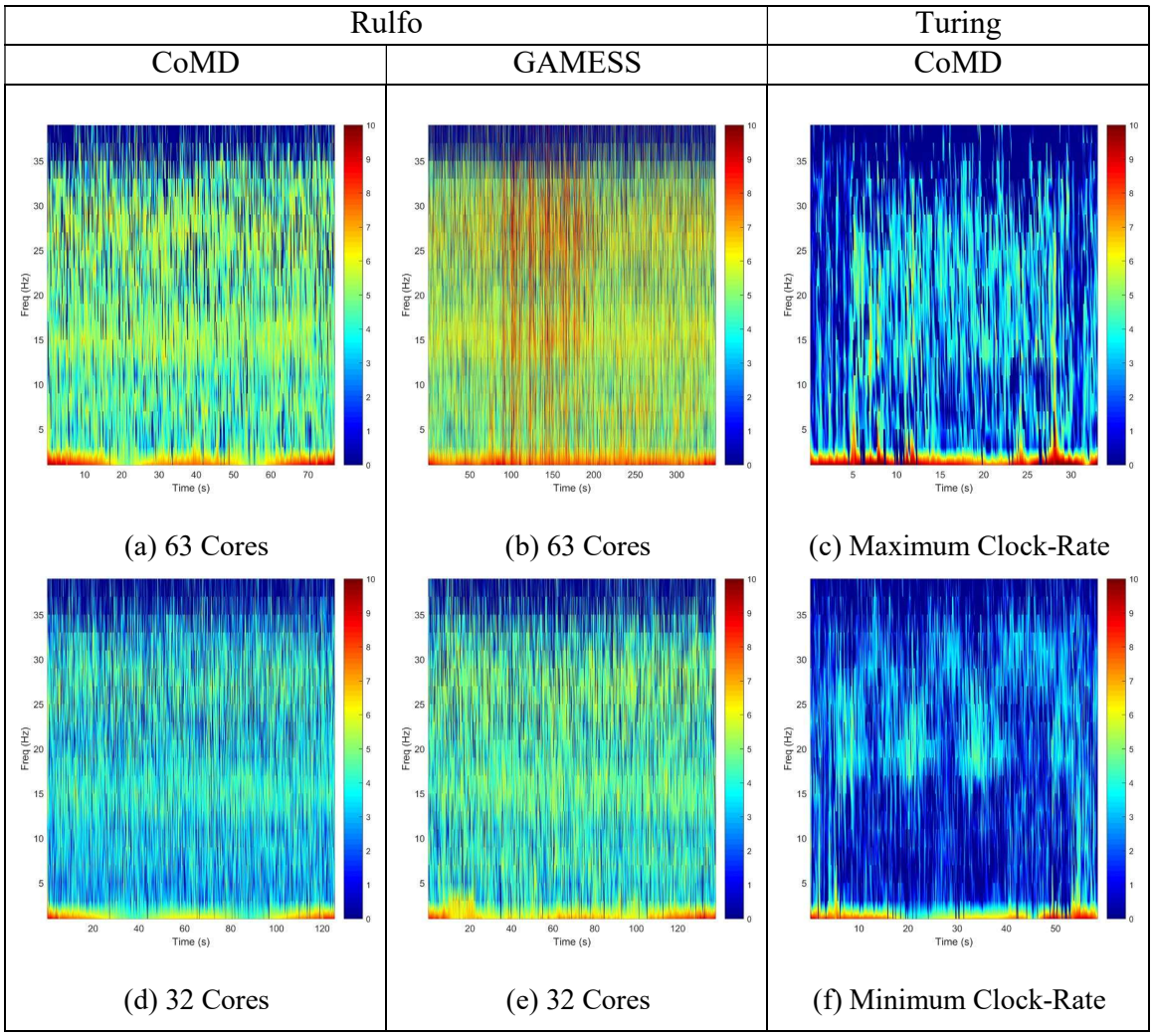


Fig. 10. Comparison of EMD/HHT histograms generated for power traces collected by running CoMD and GAMESS on different systems while varying the number of cores or clock- rate. From left to right, the first two columns present the histograms on the Rulfo for CoMD and GAMESS with 63 cores (a & b) and with 32 cores (d & e). The final column presents the histograms for the Turing system with maximum clock-rate (c) and minimumclock-rate (f).

coincides with the more optimal execution. It has been observed earlier [60] that GAMESS is a memory-intensive application that achieves optimal performance with half of the

maximum number of cores on Rulfo because of the limited L2 cache size (32 MB for 64 cores). Indeed, for GAMESS, a moderate intensity is seen in the 32-core trace (Fig. 10e) while the plot for 63-core trace exhibit high intensity, notably between 50 and 200 seconds (Fig. 10b). Such a high intensity for larger frequencies suggests that performance bottlenecks have been encountered by the execution.

Figures 10c and 10f present comparisons of the maximum and minimum clock-rate (P-state) for Turing. Similarly, to decreasing the number of cores, smaller clock-rate reduces the intensity across the entire time-frequency domain. A smaller clock-rate, however, did not impact the bands found in the Turing trace for frequencies from 18 to 34Hz.

3.5 Modeling the EMD Residual Trend

The residual produced by EMD may be used to describe power as a function of time [60]. The model is obtained by first applying EMD to a complete power trace and then fitting a quadratic equation (see Eq. (7)) to the residual.

$$P(t) = at^2 + bt + c\ . \tag{7}$$

Denote this model as the quadratic-fit residual (QFR) model.

3.5.1 Constructing the QFR

The first step is to obtain a power trace. In this work, power measurements are sampled at a rate of 5ms, which is close to the maximum available sampling rate of 1ms. A sampling rate of 5ms ensures all samples return a reliable measurement as well as allows for a significant number of IMF modes to be extracted from the trace to get an accurate and

reliable residual. Lower sampling rates, even on the order of hundreds of milliseconds, would suffice for producing a residual trend. However, the higher the sampling rate, the more IMF components that may be extracted since each IMF component resembles a particular time-scale (defined as the time between successive extrema). The residual is on the largest time-scale obtained by EMD.

The next step is to apply EEMD to the power trace. For a more reliable fit, EEMD may be applied to power traces collected for several duplicate runs and then fit the quadratic model to the collection of residuals. In [60], a total of 5 duplicate runs was considered, and a white noise of 5 W was applied to the time-series and averaged over 50 EEMD passes. These are the smallest values, which were found to be sufficient to remove the final residual contamination, and fit the model to the residuals with $R2 > 0.95$. Note that several traces are used due to the variability in execution characteristics (e.g., memory stalls and conflicts).

The shape of the residual is consistent between power traces because idle power is measured for several seconds before and after the execution. These "cool" periods influence the EMD residual to start at and return to the idle power draw, forming a concave-down quadratic. Therefore, the maximum power draw appears towards the center of the entire trace with minima at the ends (see Fig. 1 for examples).

Once the quadratic model is obtained, the execution parameters, such as the total time, average power, and total energy may be quite easily defined as follows:

- Total time is difference in the start and end times, the start time is always zero for the model, and the end time is taken as the time when the power draw equals or is less than power at the start (zero) time.

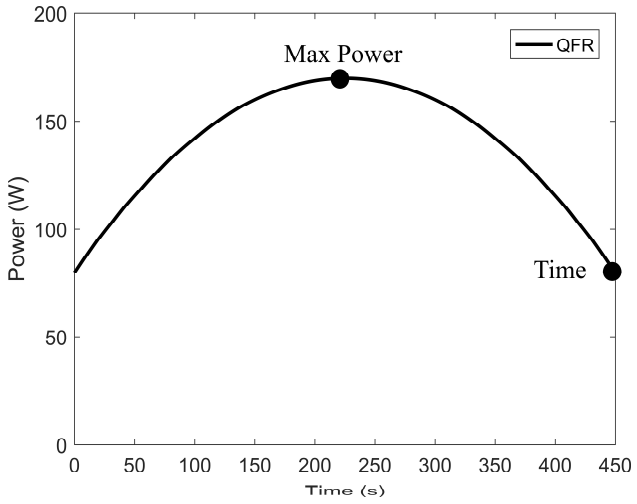


Fig. 11. QFR model of power over time.

$$T = \frac{-b}{a} \tag{8}$$

$$P_{max} = \frac{-b^2 + 4ac}{4a} \tag{9}$$

$$a = \frac{-b}{T} \tag{10}$$

$$b = \frac{4P_d}{T} \tag{11}$$

- Average power is an average of the power draw as defined by the model.
- Energy is found by integrating the model between start and end times.

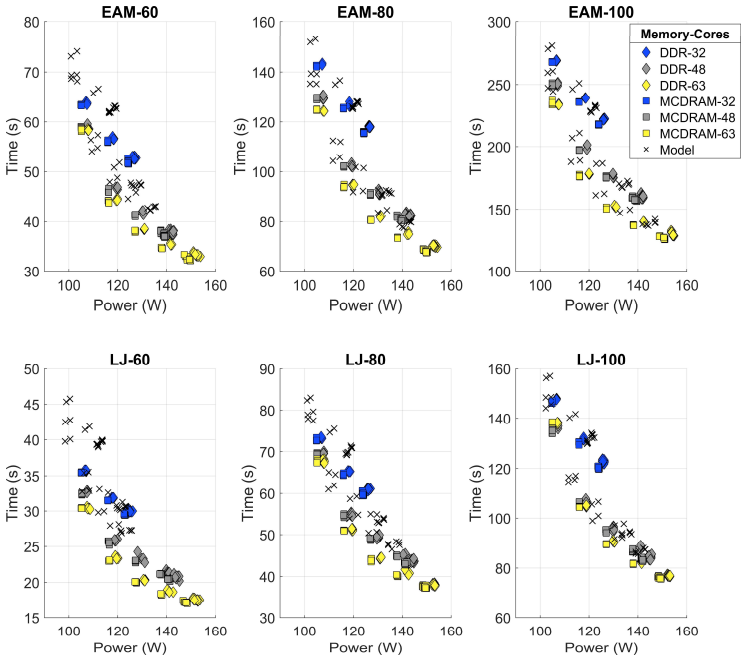
An example QFR model is shown in Fig. 11. Power is shown on the y-axis, and time is the x-axis in Fig. 11. The power model describes the trend in power draw over time where power draw always returns to idle. The coefficients relate to time in maximum power draw as shown in Eq. (8) and Eq. (9). To obtain these definitions, assume that c=0 and static power draw is removed from the trend; then time may be described as the x-intercept greater than zero, see Eq. (8). Power is defined at the apex, or axis of symmetry [104], as

shown in Eq. (9). Notice that power, even defined as a quadratic, has a static and dynamic component, where static power is coefficient c. Conversely, the coefficients may then be defined using these definitions; a is shown in Eq. (10) and b is shown in Eq. (11). The QFR shown in Fig. 11 was created for a time of 450s, static power of 80W, and dynamic power of 90W - the coefficients are then a = −0.0018, b = 0.80, and c = 80. Using the QFR to model energy, it has been shown that measured energy for traces longer than 100 seconds has an error of 10% or less [60].

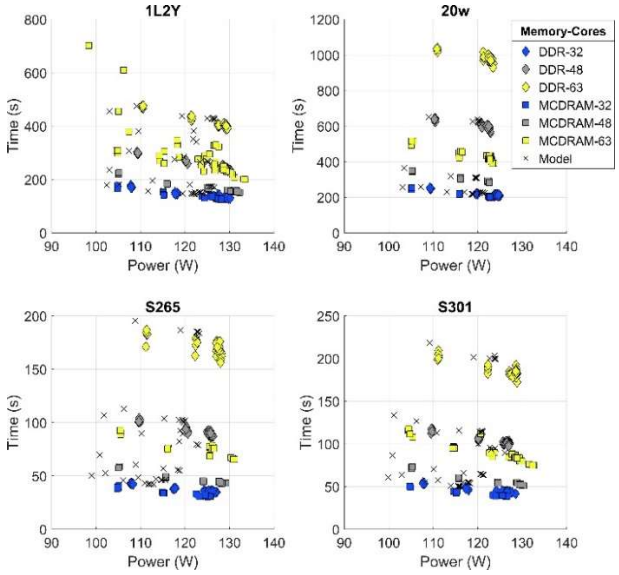
3.5.2 Modeling Energy using the QFR

Table III and Fig. 12 present the measured and modeled time-to-solution and average power draw for each workload, memory type, and number of cores. First, notice that the power limiting has a larger impact for CoMD than GAMESS. In particular, CoMD shows a linearly decreasing pattern as power draw increases while GAMESS shows a marginal decrease in the execution time as power draw increases and a large range of time-to-solution values, depending on the number of cores.

These results may be explained by a general observation that, for CoMD, the maximum number of cores is always preferred while, for a GAMESS workload, smaller numbers of cores may lead to the best execution time, which is less affected by the power limiting and L2 cache saturation. Also, recall that the minimum power limit tested is 90 W, yet the resulting minimum measured power for CoMD or GAMESS is 105W as indicated in Fig. 12, which is in line with authors' previous findings in [67]. When comparing modeled and measured values in Fig. 12 observe that the model calculates reasonably well both the time and average power usage for each configuration, even though power trends



(a) CoMD



(b) GAMESS

Fig. 12. Measured and modeled time vs power for (a) CoMD and (b) GAMESS with two memory types (DDR and MCDRAM) and three core counts (32, 48, and 63); and one subplot per workload.

TABLE III

BEST EXECUTION TIME, POWER, AND ENERGY ACROSS ALL THE WORKLOADS IN COMD AND GAMESS AND DRAM MEMORY TYPES.

			DRAM Memory					
		Best Config	DDR			MCDRAM		
Application	Workload	# Cores, PLimit(W)	Time(s)	Power(W)	Energy(J)	Time(s)	Power(W)	Energy(J)
CoMD	LJ (60)	63, 120	19	141	2654	18	138	2528
	LJ (80)	63, 120	41	142	5830	40	138	5549
	LJ (100)	63, 120	82	142	11664	82	138	11279
	EAM (60)	63, 120	35	142	5007	35	138	4776
	EAM (80)	63, 120	75	143	10681	73	138	10153
	EAM (100)	63, 120	140	142	19906	137	138	18921
GAMESS	1L2Y	32, 215	130	130	16862	127	128	16190
	20w	32, 215	212	125	26332	201	123	24708
	S265	32, 215	35	126	4444	31	125	3865
	S301	32, 215	44	127	5534	39	126	4905

TABLE IV

MODEL COEFFICIENTS AND CALCULATED ENERGY FOR ALL WORKLOADS OF COMD AND GAMESS AND DRAM MEMORY TYPES.

		DRAM Memory							
		DDR				MCDRAM			
Application	Workload	*a*	*b*	*c*	Energy (J)	*a*	*b*	*c*	Energy (J)
CoMD	LJ (60)	-0.432	12.161	63.461	3388	-0.401	11.202	64.754	3267
	LJ (80)	-0.105	5.340	84.493	6561	-0.102	5.145	83.051	6362
	LJ (100)	-0.026	2.414	96.495	12297	-0.025	2.298	95.328	11933
	EAM (60)	-0.134	6.138	80.571	5844	-0.133	5.940	80.123	5537
	EAM (80)	-0.030	2.548	97.848	11321	-0.029	2.481	95.512	10866
	EAM (100)	-0.008	1.134	108.488	20461	-0.008	1.130	104.974	19512
GAMESS	1L2Y	-0.009	1.381	90.230	18109	-0.010	1.406	88.682	17565
	20w	-0.003	0.599	99.350	27176	-0.003	0.640	95.837	25944
	S265	-0.107	4.912	76.968	5286	-0.128	5.428	73.563	4746
	S301	-0.075	4.128	80.108	6454	-0.088	4.400	79.125	5768

are underestimated and further model tuning may be warranted. In particular, for CoMD (Fig. 12a), the model closely matches the measured values with power underestimated by approximately 5–10 W. For GAMESS (Fig. 12b), on the other hand, the power is underestimated by almost 15 W in some cases.

Using the constructed model, the same best configurations, specified by the (# Cores, Power Limit(W)) pair, were found as those observed with measurements (see column Best Config in Table III). For these best configurations, Table IV provides the modeled energy values and quadratic model coefficients a, b, and c. Observe that the model acceptably calculates the total energy consumption (cf. columns Energy(J) in Tables III and IV). Specifically, the modeling error is within 10% for longer executions, i.e., for those taking greater than 100s, while the error increases up to 30% for shorter ones. For CoMD, which contains shorter traces, the overall average error has been found to be 15%. CoMD problem size of 100 shows the least error of 5–10%, whereas the problem size of 60 shows the largest error of 15–30%. For GAMESS, workloads 1L2Y and 20w result in the smallest error (less than 10%), but the errors in S265 and S301 are in the 15–25% range. Large model errors may be attributed, in part, to fitting the quadratic model into the outcome (residuals) of the EEMD procedure, which itself may incur errors of up to 10% [26], as verified empirically in the course of this work. Note that fitting into the raw traces is practically an impossible task, circumventing which is a principal objective of the current work. It may be possible to decrease the errors by increasing the power sampling rate, which is set to 5ms in this work.

It may be also observed in Table IV that, for the test cases with MCDRAM, the model predicts always less energy consumption than that predicted for the cases with the

DDR memory, which is in line with the measured results. From Table IV, some tendencies of the model coefficients may be noticed. In particular, as problem size increases, the coefficients *a* and *b* decrease for both CoMD and GAMESS. Also, *a* is always negative, which is a trait of a concave-down shape of the quadratic residual. Finally, the coefficient *c* always increases with problem size. Further testing is necessary to observe how these tendencies hold across other platforms.

3.6 Trace Segmentation

This method, denoted here as *segmented trace modeling* (STM), uses segments of the power trace to approximate the QFR on the entire trace, thereby reducing the amount of measured data required for the model construction.

STM approximates the QFR using a power trace of only a fraction of total execution time, which speeds up the entire EMD modeling process, and makes the power trace handling manageable. In particular, the modeling with EMD is dominated by the number of times the EMD is applied to the trace, which increases non-linearly with the number of samples. For example, a 30-second segment at 5ms (6,000 samples) the processing time is about 24s, and after 60 seconds of the trace, the time for processing is already about two times greater than the segment length. Thus, only small-size segments may be processed in at the runtime. Segment trace modeling becomes possible because EMD can be applied to a time-series of any length as long as there are enough measurement samples (as few as five).

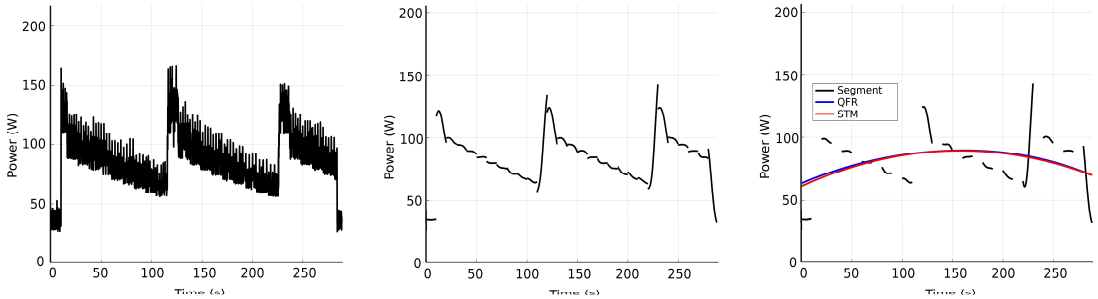


Fig. 13. Segmenting a power trace: the original power trace (left), set of residuals when EMD was applied to each 10-second segment (center), and a comparison of residuals for the trace with missing segments (right).

3.6.1 Segmenting Power Traces

In this work, 2,000 measurement samples are used per segment. EMD is sensitive to the sampling rate; the more fine-grained the samples, the more information EMD can yield. However, the more fine-grained, the more space required to store such a trace. Note that modern systems are also limited in the maximum sampling rate allowed, which is about 1ms. For a realistically stable sampling rate, a value of 5ms is used, which leads to 10-second trace segments. A 10-second segment may also fit the experimentally found durations of the pre- and post-execution measurements, which span five seconds each. See [60] for a discussion of the importance of these measurements.

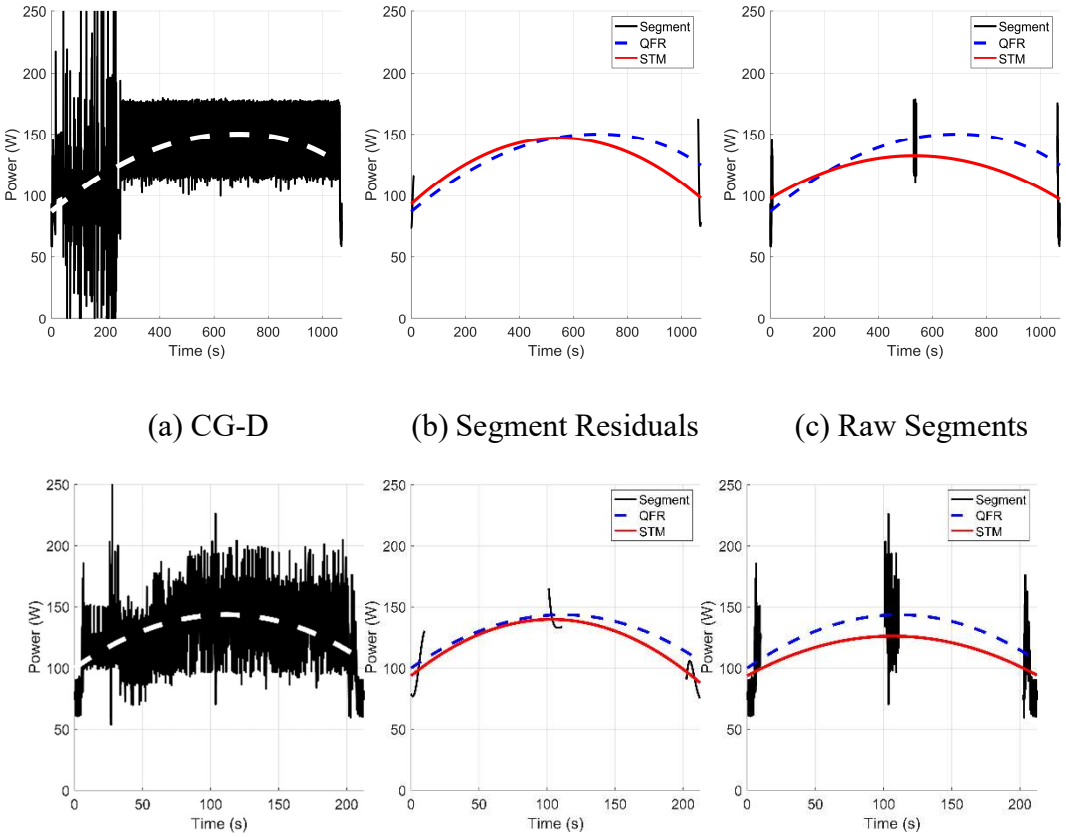
Figure 13 shows a power trace and the result of applying EMD to a series of segments (center). The trace was collected for the CoMD proxy application on an Intel Xeon E5-2650 v1 with 16 cores. Notice that EMD closely mimics the trend of each segment with respect to the original power trace. This shows that EMD can be applied to segments, and also that the resulting residual will accurately represent the trace segments. The next

step is to investigate the residual with missing segments, as is shown in Fig. 13 (right) where every other segment is considered for the STM. The difference between the STM with every other segment missing and the QRF, which is done on the entire (non-segmented) trace, is within 1% (cf. red and blue curves, respectively, in Fig. 13 (right), which essentially overlap). This shows that STM with missing segments is a good candidate for approximating QFR.

The STM method requires a minimum of three *key* segments, broadly denoted as *start*, *end*, and *workload*. The *start* and *end* segments are required to capture power draw at the start and end of the application with respect to idle power. Generally, an application begins by allocating memory and reading data from the hard-drive; this causes a large spike in power draw which is captured by the *start* segment. Likewise, when the application exits and memory is released, a large drop in power draw is observed which is captured by *end*. The *workload* segment depends on the application; at least one segment must be provided. Applications with large variations in power draw may require additional segments to more accurately estimate workload power draw. In this work, only one *workload* segment is used, for the sake of simplicity of exposition. The number of *workload* segments, however, may depend of the nature of the application power trace, and its determination is left as future work.

3.6.2 EMD on Partial Trace

Figure 14 presents two examples of the STM method applied to complex power traces. The two applications, CG [76] and GAMESS [30, 86], were chosen because their power traces exhibit much variability, which may hinder their modeling if relied on the



(a) CG-D (b) Segment Residuals (c) Raw Segments

(d) GAMESS-1L2Y (e) Segment Residuals (f) Raw Segments

Fig. 14. The STM method applied to complex power traces. The QFR is shown as a dashed line in each plot (white and blue), the STM is a solid red line, and segments in black.

periodicity in traces, e.g. Fig. 14a and Fig. 14d show the original power trace for CG and GAMESS, respectively, as well as their QFRs (white dashed lines). Fig. 14b, Fig. 14c, Fig. 14e, and Fig. 14f compare the corresponding QFRs and STMs with only three segments, chosen in a certain way. Specifically, the *workload* segment is composed of one 10-second interval taken from the absolute center of the trace. This segment was chosen to keep the end of execution which impacts the resulting STM. This may be desired to more accurately model the power draw of an application that ends with a higher power draw than that when

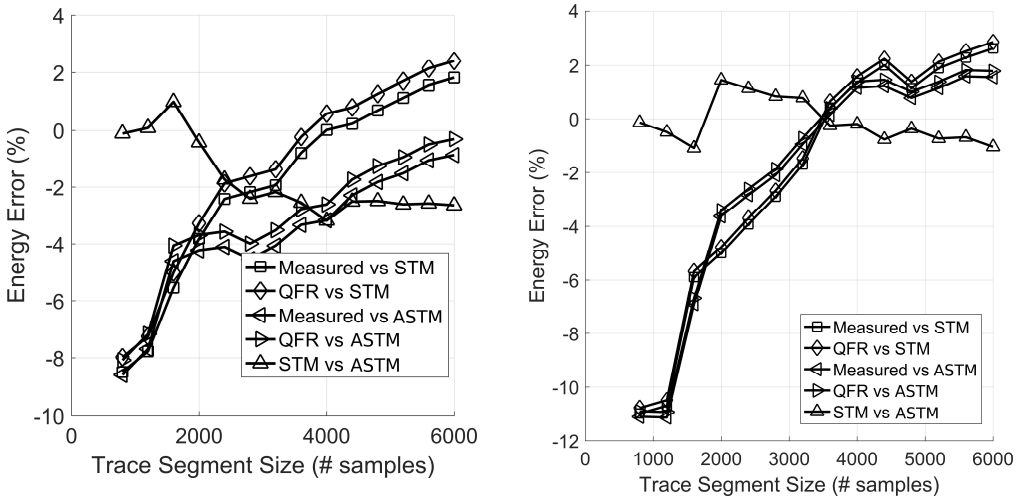
starting the application (cf. CG in Fig. 14Fig. 14a).

Observe the differences between the QFR and STM model curves in Fig. 14b and Fig. 14c and Fig. 14e and Fig. 14f, respectively. When EMD is applied to each of the three chosen segments, the error between QFR and STM is within 5% of the measured energy as shown in Fig. 14b and Fig. 14e, while the error is greater than 10% when using the raw traces of the three key segments for the quadratic fit as in Fig. 14c and Fig. 14f. Hence, the STM method, which employs EMD on the key segments followed by the quadratic fit, is beneficial.

Next observe that the QFR model more closely mimics the power trend since it follows closely the entire trend. On the other hand, the STM method accuracy may be improved by adding more *workload* segments, and thus, capturing various trace spikes. Hence, a trade-off between the STM method accuracy and speed of processing with EMD may be sought and tailored to the particular needs and resource availability.

3.6.3 STM with Segment Approximations

Recall that the STM requires three key segments: *start*, *workload*, and *end*. The *start* segment can be measured easily by the user, since only one segment is needed, and the time for each segment is relatively short compared to the total execution time. Assuming that average power is known, an *artificial* segment, where every sample is equal to the average power, may then be created as substitute for the *workload* segment. The *end* segment may be approximated also, if assumed that the "cool-down" period mirrors the start-up one—corresponding to the *start* segment—with a negative slope. Hence, the *start* segment characteristics may be used in place of those for the *end* with the samples in



(a) CG-D (b) GAMESS-1L2Y

Fig. 15. Energy consumption error for the STM and ASTM methods applied to complex power traces.

reverse order (with respect to time). The STM with the *start* segment mirroring and the *artificial* segment creation is denoted henceforth as *approximating* STM (ASTM).

3.6.4 Relative Modeling Error

Figure 15 shows the relative error in energy consumption for three pairs of models—(QFR, STM), (QFR, ASTM), and (STM, ASTM)—and for measured energy vs STM and ASTM with the increase in the number of samples used. CG class D and GAMESS IL2Y were run on the Rulfo system (KNL) with 48 and 63 cores respectively.

All but one error curves approach zero as the segment size increases, although errors between STM and measured or QFR-modeled errors continue to grow beyond 30-second segments. The error in the energy consumption between the STM and ASTM exhibits a

horizontal trend, stating from zero and leveling at about 3% of difference. This indicates that the ASTM is a good approximation of the STM. In general, the overall small magnitude of errors demonstrates that the segmented trace modeling STM as well as its variant ASTM approximate QFR of the entire trace with an acceptable accuracy. Note that, although Fig. 15 only depicts CG and GAMESS modeling errors, these errors were computed for all the applications tested and were found to be of magnitudes and trends comparable to the ones in Fig. 15.

CHAPTER 4

INVESTIGATING ENERGY ON PLATFORMS FEATURING INTEL XEON PHI

In this chapter, several methods for investigating energy are presented. First, an evaluation of thread affinity and the impact of thread mapping is conducted for the Xeon Phi. Then, DVFS has been applied to heterogeneous executions featuring offload execution. Finally, a real-time strategy has been applied to the Xeon Phi with power limiting to attempt DVFS on the device, although not in the same manner as traditional CPU's. Power limiting is closely related to power capping strategies. The investigations in this chapter motivate further investigation and analysis of power and performance modeling for applications and hardware.

4.1 Thread Affinity on Intel Xeon Phi

The work in this section investigates both the performance and energy effects of thread affinity on the NASA Advanced Supercomputing (NAS) Parallel Benchmarks, which are compiled to run natively on the Intel Xeon Phi (KNC). Specifically, the execution time and energy consumed by the Intel Xeon Phi are evaluated under different thread affinity modes. Going beyond measuring the execution time, other performance metrics relevant for Xeon Phi are explored, such as average cycles per instruction (CPI) per thread, memory bandwidth, and vectorization intensity. The energy and approximate execution time are computed based on measurements captured using the MIC System Management and Configuration micsmc utility tool [8].

The Xeon Phi coprocessor (KNC) is composed of 50+ cores at approximately 1GHz. Each core is capable of concurrently processing four hardware threads [45]. It is not

possible to execute instructions from a single thread in back-to-back cycles; therefore, a minimum of two hardware threads per core is suggested. Threads may be mapped to cores through the affinity environment variable that governs six possible modes: balanced, compact, scatter, none, disabled, and explicit [44, 32]. The “disabled-affinity” setting provides no thread affinity interface, while the “explicit-affinity” setting allows the user to manually assign each thread to any core. These two affinity settings are not considered here however.

The balanced affinity mode evenly distributes threads among the cores. This mode attempts to use all the available cores while keeping the thread neighboring logical IDs physically close to one another. The scatter affinity also evenly distributes threads among the cores but it does so in a round-robin fashion. Hence, threads with the adjacent IDs are not guaranteed to be physically adjacent. The compact affinity distributes threads by assigning the maximum number of threads (which is 4) to a core before assigning threads to another core. This mode keeps the threads grouped tightly together and uses fewer cores than other affinity modes unless the maximum thread count is set, at which point the balanced and compact thread mappings are identical. When the thread affinity is not specified explicitly, the system-default setting is none.

Thread granularity specifies the way various affinity modes are applied. On the KNC, there are three levels of granularity: fine, thread, and core. The fine and thread levels are similar in that they bind threads to a single context when the thread is assigned to a core. The core granularity binds threads to a core, such that the threads may float within the context of the physical core [44, 5]. Using the former may improve reproducibility of the results and avoid the overhead from thread context switch. The system-default

TABLE V

EXECUTION TIME (SECONDS) OF 59 AND 236 THREADS FOR VARIOUS BENCHMARK PROBLEM-CLASS SIZES.

	59 Threads					236 Threads				
NPB	A	B	C	D	E	A	B	C	D	E
EP	1.97	7.83	28.94	455.84	—	1.05	4.18	16.44	268.69	4229.20
CG	0.49	49.61	206.82	†	†	0.21	12.98	53.31	†	†
FT	0.93	12.54	52.50	†	†	0.71	10.57	45.15	†	†
IS	0.17	0.52	3.26	†	NA	0.39	0.59	1.99	†	NA
MG	0.26	1.41	8.28	†	†	0.24	1.19	8.14	†	†

— - Did not test NA - Class size not available † - Out of memory

granularity setting is core.

4.1.1 Measuring Time and Energy

Table V presents execution time required to complete each benchmark for the various classes at 59 and 236 threads. Only the EP benchmark was able to execute class sizes larger than C on the Xeon Phi. This is due to the small memory footprint required for EP. Therefore, each benchmark is executed with class C problem sizes.

Energy consumption is calculated by the approximation of power timeslices provided by the Intel utility tool, micsmc. From the command line, micsmc can output a wealth of data including clock-rate, power, temperature, memory usage, and CPU utilization per core [50]. However, as additional types of data are requested, the delay between calls increase. Hence, to capture the energy readings in the smallest timeslice available, only clock-rate and power data are recorded (both are printed when the same input parameter is given). The micsmc tool measured and reported every 22–28ms, which is sufficient for a detailed evaluation. Data are collected from each unit on the device via

the performance monitoring unit (PMU) [45, 18]. Each core contains an independently programmable PMU, which supports four hardware threads, two hardware counters per thread, and 40-bit precision per hardware counter.

Dynamic Voltage and Frequency (Clock-Rate) Scaling (DVFS) involves changing the voltage and clock-rate levels of the processor to reduce or increase power, which may be performed in application software. This technique generally requires a careful implementation to reduce potentially severe performance penalties. When applied judiciously, however, it may yield as much energy savings as 14% with a modest performance loss of 2% on certain NAS parallel benchmarks as was shown by the authors in [94].

Unfortunately, Xeon Phi does not allow user-controlled DVFS. The hardware performance levels (P-states) are selected and set through the coprocessor OS kernel. P-states may change depending on the thermal or power readings. A new P-state is selected by the OS upon crossing a high thermal threshold or approaching one of the upper power limits [45]. Additionally, the Intel Xeon Phi may perform DVFS selectively for inactive cores. Hence, varying power consumption is explored here by varying the active core count under the compact affinity mode, in which it is possible to leave some cores idle when mapping threads numbered greater than the total core count. These idle cores may allow the device to save power with a certain performance loss.

To demonstrate the accuracy of the data obtained by micsmc, its output is compared to that collected by the Wattsup power meters, having a sampling rate of 1 Hz, which does not affect the measurements considerably since the benchmark used here has a sufficient

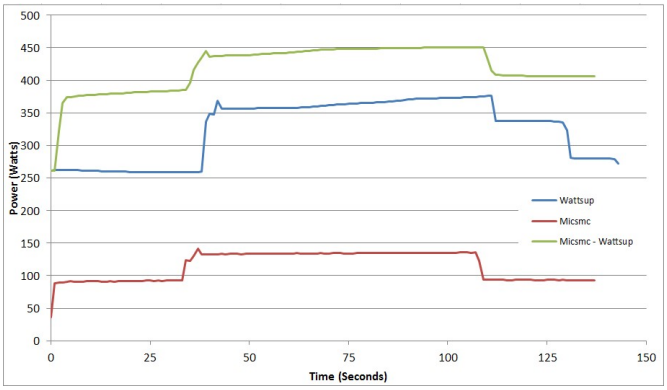


Fig. 16. The power profiles as obtained by Wattsup and *micsmc* for the CG benchmark with the *compact* affinity mode at 180 threads.

execution time. Wattsup meter records the total power for the computing system to which it is connected; two meters are used here because the system has two power outlets. Fig. 16 presents the power profiles, averaged over three runs, for micsmc alone (curve Micsmc), for the sum of the two Wattsup power (curve Wattsup) meters, and for the Wattsup power meters during Micsmc (curve Micsmc-Wattsup) when the CG benchmark is executed at 180 threads in the compact affinity mode. From Fig. 16, it is clear that micsmc and Wattsup are reporting power traces of similar patterns, with fluctuations appearing at the same time points. However, from the difference between the "pure" Wattsup and Micsmc-Wattsup power profiles, it may be observed that micsmc incurs a substantial overhead, which is caused by the fact that micsmc always puts the device cores into a rather high energy state (around 100 watts, as seen near time 0 in Fig. 16), even if they are idle, when polling each core for the power data.

4.1.2 Results

The experiments were conducted on Borges computing platform. The execution time is obtained from the NPB output, and Intel VTune Amplifier XE software samples the hardware event counters. The Intel Xeon Phi power was measured with *micsmc* to compare the energy of different affinity modes used on the device only.

Figure 17 presents the average normalized execution time observed for the EP, FT, IS, and MG benchmarks with different affinity modes and *thread* granular ity. The execution time shown is an average over four runs, which is normalized against the default test. (Thus, values less than one are an improvement over the default case.) Recall that the default case uses the *core* granularity and *none* affinity. Figure 17 provides several examples of applications that perform better with a specified affinity and granularity. IS and MG under the *compact* affinity perform exceptionally well after 180 threads when compared to the default test.

However, it also provides several examples of applications which under-perform when fewer cores are utilized: All four benchmarks perform worse (value less than 1) while using the *compact* affinity for thread counts fewer than 180. In EP and FT, the *balanced*, *scatter*, and *none* affinities chaotically shift between being more or less efficient than the default. However, after 180 threads, each affinity observes a boost in performance, except for the *scatter* one. Thus, NPB perform best when each Intel Xeon Phi core is running at least three threads per otherwise empty core, which is in agreement with the general optimization guideline of running four threads per core. After 236 threads, the performance of each affinity dramatically increases or decreases, depending on how well the threads are able to compete with the OS for resources, which occupy one core. Generally, the *scatter*

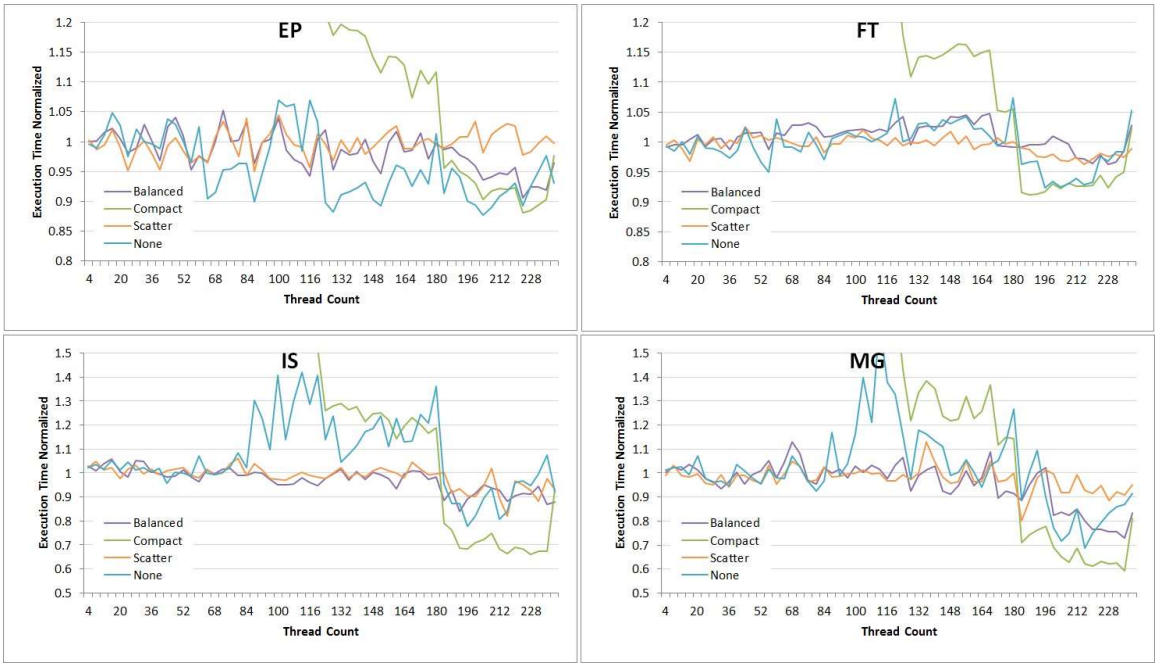


Fig. 17. Normalized execution time for the EP, FT, IS, and MG benchmarks with different affinity modes and the *thread* granularity. Each value is the average of four runs for each benchmark, and is normalized for each affinity against the default test, which has the affinity *none* and granularity *core*. The data for *compact* affinity are shown starting at the number of threads greater than for the other affinities to permit good scaling of the plots.

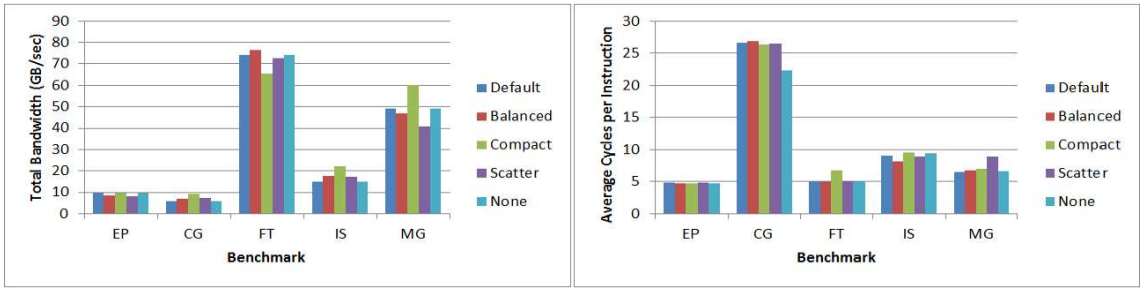


Fig. 18. Benchmark total bandwidth (left) and average CPI (right) for each affinity mode with the granularity *thread* at thread counts from Table VI.

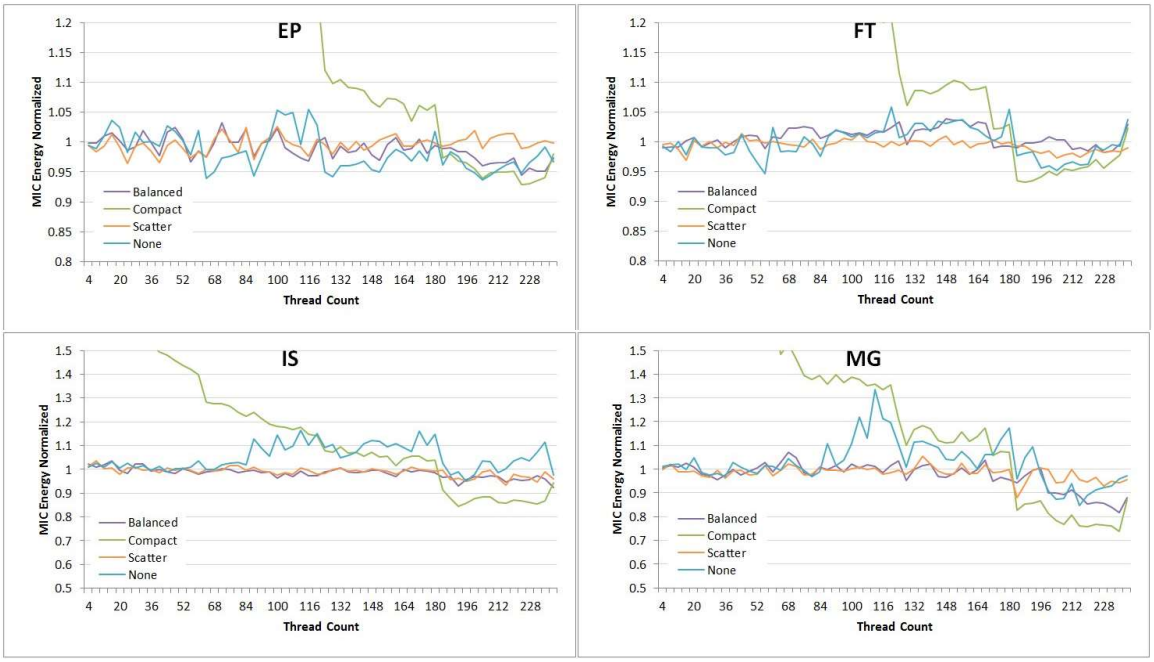


Fig. 19. Normalized energy for the EP, FT, IS, and MG benchmarks with different affinity modes and the *thread* granularity. Each value is the average of four runs for each benchmark, and is normalized for each affinity against the default test, which has the affinity *none* and granularity *core*.

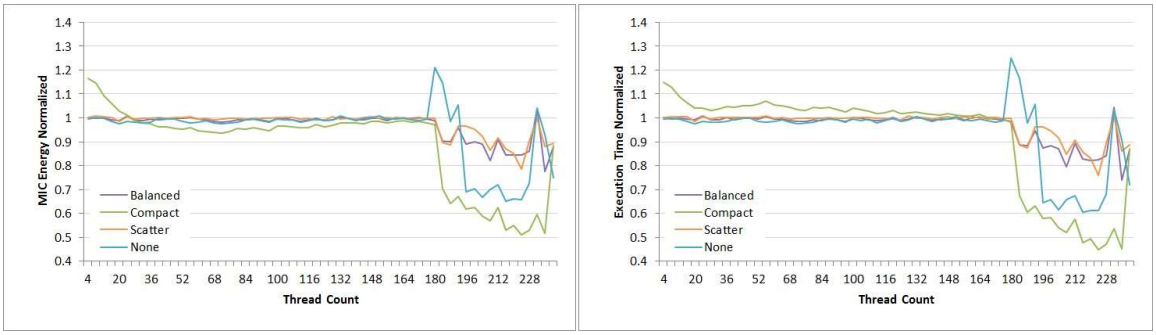


Fig. 20. Normalized energy (left) and execution time (right) for the CG benchmark with different affinity modes and the *thread* granularity. Each value is the average of four runs & is normalized against the default test, which has the affinity *none* and granularity *core*. and *none* affinities perform better at 240 threads.

4.1.3 Performance Metrics

To gain further insights into the performance of each benchmark, this work considers certain performance metrics as obtained from the hardware event counters provided by the Intel VTune Amplifier XE 2013 [101]. In particular, these metrics are average CPI per thread, vectorization intensity (VI) and memory bandwidth. The average CPI per thread (CPI) is the average number of CPU cycles required to retire an instruction, averaged per thread. It can be used to detect latency in the system which affects the applications execution. VI provides an indicator for how well the entire code maps to the VPU. For double-precision and single-precision operations, the VI ideally should be close to 8 and 16, respectively. The memory bandwidth metric describes the average streaming memory bandwidth achieved during execution.

After running the Intel VTune Amplifier, it was observed that VI did not vary when varying affinity for any of the benchmarks. The EP, CG, FT, IS, and MG benchmarks exhibited a VI of 3.2, 3.0, 8.0, 1.33, and 6.4, respectively. This is attributed to the NPB benchmarks applications using 64-bit precision and the VPU length is 512-bit. FT and MG are observed with high vectorization utilization, suggesting few or no data dependencies within the vectorized loops. The total bandwidth (Fig. 18 left) is much lower than the peak value of 320 GB/s or even than the observable 140 GB/s, as presented in [12]. The measured CPI (Fig. 18 right) is much closer to the expected value of four, except for CG which incurs around 26 cycles per instruction on average. In general, FT, with over 70 GB/s and a low CPI, utilizes the resources of the Xeon Phi well compared to the other benchmarks. MG features moderate bandwidth, acceptable CPI, and high VI.

4.1.4 Energy

For the executions shown in Fig. 17, Fig. 19 presents the average normalized energy observed for the EP, FT, IS, and MG benchmarks, which is averaged and normalized as in Fig. 17. From Fig. 19, it is seen that, for the thread counts greater than 180, specifying affinity and granularity can decrease energy consumption. It is not until after 180 threads that the energy savings of using fewer cores outweigh the loss of the performance. Consecutively, EP results in almost 8% of energy savings with the *compact* affinity, FT results in 6%, IS results in 13%, and MG results in almost 23% energy savings.

For the CG benchmark (Fig. 20 left), energy savings may be observed already for fewer cores under the *compact* affinity mode, starting at 28 threads even though execution time is larger than the default test (normalized value greater than one in Fig. 20, right) and larger than other affinity modes until 180 threads. Since CG is the most memory-accessing application among the ones considered here as was shown in [94], this situation may be explained by the findings in [14] that Intel Xeon Phi uses less energy for memory accesses. Under the *compact* affinity mode, the neighboring threads, which are more likely to access memory simultaneously, are located in the same core. Hence, an entire core may be considered as memory-accessing to enjoy the lower energy usage, while the execution time is high because of thread contention.

After 180 threads, which was the same threshold observed in every other benchmark, the CG performance increases drastically leading to substantial energy savings. At the 180-thread count, better thread load balancing takes place as threads are evenly distributed to cores, at three per core. Good load balancing occurs at other thread counts as well, such as 60, 120, and 240. In particular, after 120 threads (two per core), one of the

TABLE VI

LOWEST ENERGY CONSUMED WHEN SPECIFYING THE AFFINITY MODES AS COMPARED TO THE SYSTEM-DEFAULT SETTING.

	EP	CG	FT	IS	MG
E Default (J)	3,173.16	17,005.39	8,274.34	1,042.97	1,813.89
E Test (J)	2,986.20	8,787.69	8,386.27	892.59	1,731.50
Aff Mode	compact	compact	scatter	compact	balanced
Thread Count	236	236	104	232	180
E Saved	+5.89%	+48.32%	-1.35%	+14.42%	+4.54%

two compute cycles will gain the option to switch between threads during execution and an increase in the performance is observed. An even greater increase is observed when the fourth thread is added and the second cycle has an additional thread (after 180 threads). Such performance increases, which lead to energy savings, are noticeable for all the user-specified affinity modes with the *thread* granularity considered here.

Fig. 17, Fig. 19, and Fig. 20 indicate that the DVFS was not applied during the benchmark executions since the performance and energy correlate so closely in all the cores. Further, this is supported by the output from the *micsmc* that shows that clock-rate did not change during any of the benchmark executions.

For each benchmark, Table VI provides the lowest energy (row `E Test`) along with the affinity mode (row `Aff Mode`) and thread count (row `Thread Count`) at which it was observed. The corresponding energy value of the default case is in row `E Default` and the difference in the energy consumption is in row `E Saved`. CG features the highest energy saving, a staggering 48% with respect to the default test. The FT benchmark was the only benchmark to consume more energy as a result of specifying affinity. It may be

due to the granularity difference with the default case because the energy consumed by the *none*,*scatter*, and *balanced* affinities are almost the same (within 0.1% of one another).

4.2 DVFS with Heterogeneous Executions

This section investigates the impact on performance and energy when CoMD is adapted to use the Xeon Phi co-processor. The impacts of varying host-side clock-rate during offload executions and the maximum atom count of link cells are explored with focus on minimizing the energy-to-solution. Impacts to time-to-solution, performance, and offload timings are also discussed. The tests are performed on a single-node computing platform that offers two Xeon Phi's; hence one- and two- Xeon Phi configurations are explored. In this section, Xeon Phi and MIC may be used interchangeably to reference the Intel Xeon Phi co-processor.

To reduce energy consumption by the CPU, a dynamic voltage and clock-rate scaling (DVFS) technique is commonly used at the application runtime (see, e.g., [94]). The current generation of Intel processors provides various P-states for clock-rate scaling. clock-rate may be specified by manipulating the model-specific registers (MSR). P-states are defined as (f_i, . . . f_n, where $f_i > f_j$ for $i < j$). For the Intel Xeon Phi, DVFS is not software-accessible. DVFS may be used on the host CPU only to reduce host-side energy consumption while computation is being performed on Xeon Phi.

4.2.1 Execution Time Model

The time on- and off-chip model [16] may be used to determine how compute- or memory-intensive an application is. This information may then be used to determine the

effect MIC acceleration had with respect to the host and the effect of DVFS during MIC offloads with respect to MIC executions without DVFS. Execution time T for an application that offloads computation to the Xeon Phi may be defined as $T = t_{host} + t_{mic}$, where the execution time of the host and offloaded code sections are non-overlapping. t_{hos} represents the execution time spent on the host multi-CPU and t_{mic} represents the execution time spent on the Xeon Phi. DVFS may save power during those CPU cycles that are not computationally intensive, e.g., when the CPU is stalled waiting for memory, I/O, branch misprediction, or reservation station stalls [94]. The host execution time t_{host} consists of the time on-chip t_{on}, when the CPU is engaged in computations, and time off-chip t_{off} for the remainder of CPU cycles. DVFS affects only the time on-chip, which scales linearly with the change in clock-rate [94]. This relationship may be described as:

$$t_{host} = t_{on} \frac{f_{max}}{f_i} + t_{off} , \tag{12}$$

where f_{max} is the maximum allowable clock-rate and $f_i, i > 0$ is any lower available clock-rate level. Substituting t_{host} to solve for T yields:

$$T = t_{on} \frac{f_{max}}{f_i} + t_{off} + t_{mic} . \tag{13}$$

Equation (13) provides the most basic relationship between total execution time, CPU (host) clock-rate, and times spent on the host and MIC. It may be used to determine the energy-saving potential for a hybrid CPU-MIC application, indicated by the ratio of t_{off} to t_{on}.

4.2.2 Experiments

The experiments conducted here aim to compare the multicore execution of CoMD, referred to as *Host*, and the CPU-MIC execution of CoMD, referred to as *MIC*. Each

execution is run using one or two MPI tasks, hence: *Host 1*, *Host 2*, *MIC 1*, *MIC 2*. When DVFS was applied to the host during MIC offloads, the corresponding tests are referred to as *MIC 1 DVFS* and *MIC 2 DVFS*. For the MIC executions, the number of MPI tasks is equivalent to the number of MIC devices employed, such that each MPI task is assigned a Xeon Phi for its portion of the computations. In each iteration of the EAM force kernel, data must be transmitted between the Xeon Phi and host four times for three offload events. In the initialization phase of CoMD, one offload to Xeon Phi is required to instantiate and allocate its memory. All static data, such as interpolation tables, are transmitted to the device at this time. The memory transfer times have been measured by setting the environment variable `OFFLOAD REPORT` to a value greater than zero.

Experiments were performed on the "Borges" computing platform at Old Dominion University. Power measurements are collected externally via two Wattsup meters which power and monitor the Borges computing platform. Data is sampled at a rate of 1Hz, which does not affect the measurements considerably since the problem size used provides sufficiently longer execution times. Although several tools exist for measuring power on the Xeon Phi itself [50, 46], they often incur a substantial overhead from measuring the device power. Wattsup offers a coarse-grained sampling solution, which, however, does not impact power measurements. The power is measured for 15 seconds before and after execution of CoMD in all the experiments. Additionally, 45 seconds of idle time is allocated in-between executions to allow the idle power draw to reach steady state. The results have been averaged over five executions of each test.

In this work, two parameters are varied: link-cell count and host CPU clock-rate.

TABLE VII

EXECUTION TIMES ON HOST IN SECONDS AND THE ASSOCIATED GOODNESS-OF-FIT METRIC R^2 FOR ALL EXPLORED CONFIGURATIONS.

Configuration	t_{on}	t_{off}	t_{off}/t_{on}	R^2
Host 1	97.139	3.769	0.039	0.992
Host 2	94.965	1.019	0.011	0.973
MIC 1	6.990	2.630	0.376	0.995
MIC 2	4.820	1.293	0.268	0.965
MIC 1 DVFS	7.051	2.635	0.374	0.996
MIC 2 DVFS	4.460	1.680	0.377	0.985

Conversely, the parameters that are not varied during the experiment are problem size, thread affinity, and thread count. Problem size has been set to 70 (1,372,000 atoms) because this workload provides a sufficiently long execution time (more than 30 seconds) to collect power samples. Thread affinity has been set to *compact* and granularity is set to *thread* because this affinity combination has been shown to provide the most efficient execution [61, 62]. Thread count has been set to 236 threads (out of possible 240), which are mapped to 59 cores. One core is left free from CoMD because it is occupied by the MIC operating system.

4.2.3 Execution-Time Model Validation

Compute- or memory-intensity of an application may be measured by the ratio t_{off}-to-t_{on} from Eq. (12). A ratio which is close to 0 represents a computationally intensive application; similar, a value greater than 1 represents a memory intensive application. Here CoMD is executed on "Borges" for each configuration at different operating clock-rates such that linear regression may be used to solve Eq. (12) for t_{on} and t_{off}. In all of the "one-

MPI" configurations (Table VII), t_{off} is about the same while it is much smaller for those with two MPI tasks because, in the latter, certain data sets are treated in a two-way parallel fashion. For the *MIC* executions, t_{on} is much smaller than *Host* because the CPU (host) counts the Xeon Phi computations as its own time off-chip. It is for this reason, that the DVFS applied during Xeon Phi offloads may benefit energy without affecting the performance.

Notice that CoMD remains rather compute-intensive (t_{off}-to-t_{on} ratio is less than one) even when the force kernel is computed on the accelerator (rows *MIC* in Table VII). In particular, for all the *MIC* configurations, this ratio is approximately 0.3, which indicates that the computations remaining on the host, such as position and velocity updates, are also compute-bound. Offloading the update sections does not benefit the MIC implementation because additional atom data would need to be transmitted to the Xeon Phi each iteration. The extra memory transfer per iteration is comparable to the computation time to perform the update. Therefore, the velocity and position updates are computed by the host.

The *MIC 2* tests with and without DVFS have rather high variation in the t_{off}-to-t_{on} ratios, which is 0.1 higher for *MIC 2 DVFS* whereas these ratios are almost the same in the *MIC 1* tests. This observation may be explained by the fact that, in the *MIC 2 DVFS* case, DVFS is applied to the entire node by only one (first) MPI task. However, no synchronization is algorithmically necessary between the MPI tasks. Hence, it is possible for the DVFS to be applied while one of the MPI tasks is still executing a host CPU portion, outside of the offloading segment. It has been verified experimentally that the host execution time t_{host} in the *MIC 2 DVFS* case was greater than that in the *MIC 2* case for several clock-rate levels, while t_{mic} remained constant independent of host DVFS.

4.2.4 Experimental Results

The impact on performance from varying link-cell count is investigated. Link-cell size has been set to either the default value of 64 atoms or to 16 atoms, which is the smallest size available for the problem size selected and, thus, may lead to more prominent differences in the performance. The highest CPU clock-rate f_{max} has been applied and remains constant throughout each execution, except in the *MIC DVFS* configurations. For *MIC DVFS*, CPU clock-rate is set to its lowest value of 1.2 GHz during offloads to the MIC(s), and restored to its highest value at the end of the offload section.

4.2.4.1 Link-Cell

The results presented in Fig. 21 compare the performance of the two link-cell counts (LC of 16 and 64) in each execution configuration. In particular, Fig. 21 (left) presents the total execution time while Fig. 21 (right) shows the offload memory transfer time for the hybrid configurations with four main offload events: Initialization and Loop 1, 2, 3. The smaller LC of 16 reduces execution time for each configuration except *Host 2*, as seen in Fig. 21 (left) *Host 2* does not benefit from the reduced link cell count because the link cell was able to fit into cache in the two MPI task version. *MIC 2* benefits from the reduced link cell count because its cache is much smaller than the hosts. In general, the cache utilization increases as a result of the reduced memory footprint of the link-cell container for LC of 16. For the *MIC* configurations, this observation may be particularly noticeable: Fig. 21 (right) shows that the memory transfer time decreases significantly with the reduced maximum atom count per link cell. Specifically, the transfer time decreases 2.75x (from eleven to four seconds) for *MIC 1* and 2.0x (from six to three seconds) for *MIC2.*

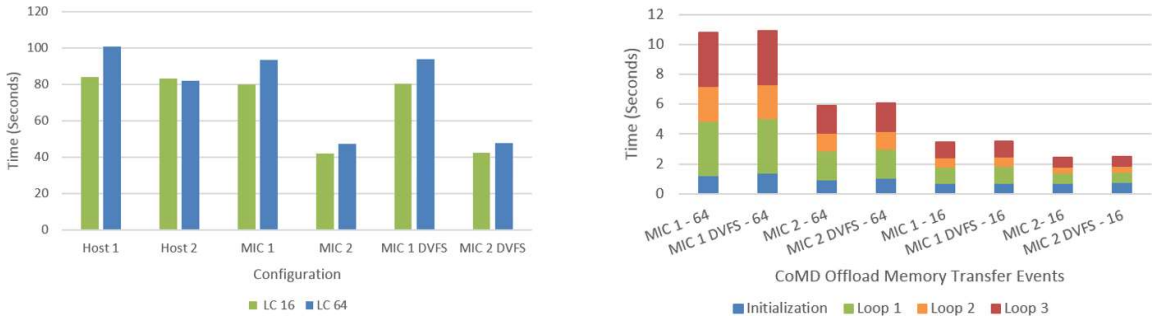


Fig. 21. Total execution time (left) and total memory transfer time per offload event (right) for two link-cell counts, 16 and 64.

4.2.4.2 Clock-Rate Scaling

Fig. 22 and Fig. 23 provide the average power, atom rate, execution time, and energy plots for the experiments when clock-rate on the host is scaled from the lowest clock-rate to the highest clock-rate. The link-cell count is fixed as 16 in the plots.

Fig. 22 (left) presents the average power during the execution of CoMD derived from Wattsup measurements for each test configuration. For the *MIC 2* configurations, the curves show the actual power measurements. For the *MIC 1* and *Host* configurations, the curves represent the measured power values adjusted to exclude the power draw of unused Xeon Phi devices, which could not be removed from the compute node in the experiments, as follows. The execution of CoMD on a single MIC causes both devices, if present, to enter an active power state. From the Xeon Phi datasheet, the active power state draws about 115W [17], therefore this number was subtracted from the actual power measurements and the resulting value shown on the *MIC 1* curves in Fig. 22 (left). For the *Host* executions, both Xeon Phi's are in an idle state, which draws about 45W [17], therefore this number was subtracted from the actual power measurements and the resulting value shown in the

Host curves in Fig. 22 (left). Note that, power draw has not been measured on the Xeon Phi devices directly because existing software tools, such as *micsmc*, incur excessive overheads as was shown in authors' earlier work [61].

Fig. 22 (right) presents the *atom rate* metric of the performance, defined as number of atoms processed per microsecond. For lower clock-rates, the atom rate is smaller because execution on the host longer. The atom rate is also indicative of the MIC performance. Hence, the rate is the largest for *MIC 2* cases, which also exhibit the best execution time as shown in Fig. 23 (left). For the MIC 2 tests, the atom rate decreases faster than that for the MIC 1 ones because the executions of the MIC 2 configuration are much shorter (cf. Fig. 22 (right) and Fig. 23 (left)). Hence, host operates at a lower clock-rate longer, which negatively affects the performance.

In Fig. 23 (left), the total execution time is plotted against the available clock-rate levels. The Host executions are significantly impacted by clock-rate, as expected, due to the computationally intensive requirements of the CoMD application. The MIC executions are not impacted by DVFS as significantly as the Host because the host clock-rate affects only the host-side computations and not the ones offloaded to MIC.

Fig. 23 (right) presents the total energy consumed for the various configurations under different clock-rates. The MIC 1 and Host values have been adjusted to remove the unused Xeon Phi power draw from the measurements as explained for Fig. 22 (left).
The MIC 2 tests consumed the least energy (with and without DVFS) for almost all the clock-rates, except for f_{max}, when Host 2, adjusted for unused MICs, shows the best result. MIC 1 does not compare to MIC 2 or to the Host executions, especially at the higher clock-rates. Hence, further optimizations are required to make MIC 1 energy efficient.

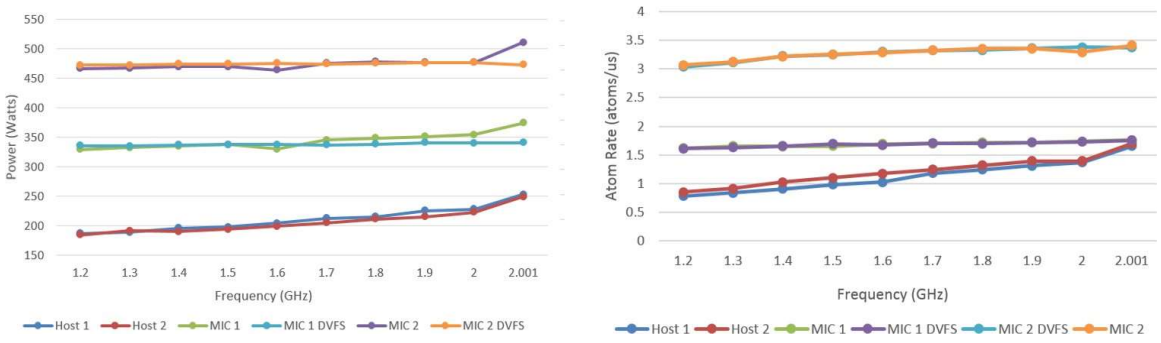


Fig. 22. Average power (left) and atom rate (right) for different clock-rate levels with and without DVFS for link-cell count of 16. (The curves *MIC DVFS* and *MIC* partially overlap for one and two MICs, respectively.)

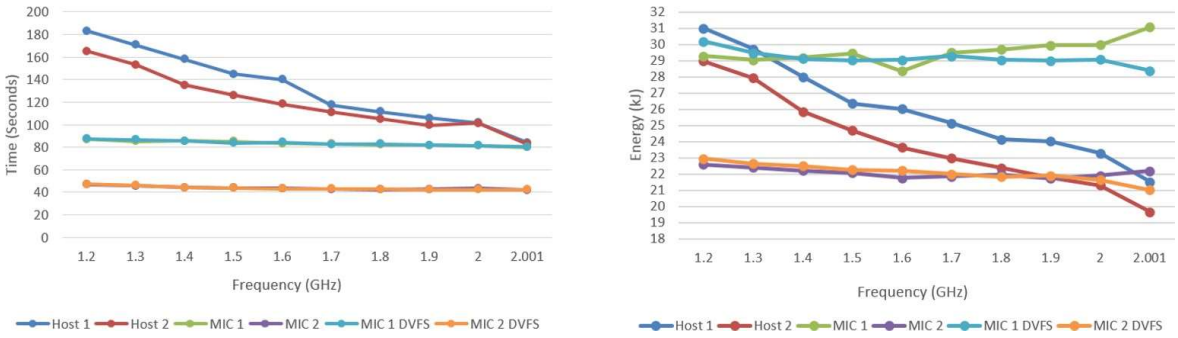


Fig. 23. Total execution time (left) and energy consumed (right) for different clock-rates with and without DVFS for link-cell count of 16. (The *MIC* curves partially overlap *MIC DVFS* for their respective tests.)

Energy savings for the MIC 2 executions were measured when varying link-cell count and when applying DVFS. When link-cell count was reduced from 64 to 16, MIC 2 saved 8.3% and MIC 2 DVFS saved 14.3% energy. The savings are the result of reduced

host– MIC memory transfer in the cased of 16 link-cell count and of reduced cache misses due to the decreased memory footprint. When DVFS is applied and link-cell count is 16, MIC 2 DVFS saved 9% in energy compared with the MIC 2 execution without DVFS while still consuming more energy than the Host 2 execution at the highest clock-rate, when the MIC 2 executions were not fast enough to compensate for the additional power consumption of the MICs (cf. Fig. 23 (left) and Fig. 22 (right)).

4.3 Real-Time Power Limiting of the Xeon Phi

In this section, an investigation of power limiting on the Xeon Phi is presented, where both an offline and real-time strategy are used to investigate the impact of power limiting on performance and energy consumption.

It is important to understand how the Xeon Phi SMC varies operating clock-rate as power surpasses the designated thresholds. Specifically, the Xeon Phi uses two power threshold values—*low* and *high*—each with a designated time window. By default, the low power threshold is set to the TDP with a time window of 100ms and the high threshold at 120% of the TDP and a time window of 10ms. When power exceeds the low threshold for the duration of the time window, clock-rate is decreased until power consumption is less than the threshold. When power exceeds the high threshold for the duration of the time window, the thermal throttling mechanism is engaged, which forces the device to the lowest operating clock-rate of around 500 MHz, as seen experimentally. More on Xeon Phi power limiting can be found in the datasheet [43]; however, it has not yet been updated for Knights Landing. For correlating the application performance with variation in clock-rate, a performance model is needed that is valid for modern out-of-order (OOO)

processors. The performance model presented here predicts the instructions retired at different processor clock-rates depending on a few system parameters. A power model is also outlined, which translates the clock-rate needs to power consumption limits. By using these power and performance models, the power thresholds are decided during application execution to obtain energy savings, based on a user-defined performance loss.

4.3.1 Performance Model

Processor clock-rate effects on the micro-operations retired may be accounted for by a cycle-accounting equation [96] as:

$$f_i = IR_i \times CPI_{exe} + N_{mem} \times \alpha \times \beta \times \frac{f_i}{f_{max}} \quad (14)$$

where

- IR_i is the number of instructions retired per second at processor clock-rate f_i.
- CPI_{exe} is the number of cycles per micro-operations retired barring the memory accesses per second.
- α ($0 \leq \alpha \leq 1$) is the OOO overlap factor, which determines the extent of memory stalls overlapped with execution cycles.
- N_{mem} is the number of memory accesses in a second.
- β is the number of cycles corresponding to the memory-access latency.
- f_{max} is the maximum processor clock-rate.

Since memory access stalls tend to have the maximum impact on the performance [96], the other types of stalls namely cache-access, branch effects etc. are not considered in Eq. (14). By rearranging Eq. (14), the instructions retired at processor clock-rate f_i

can be expressed as:

$$IR_i = \frac{f_i - N_{mem} \times \alpha \times \beta \times \frac{f_i}{f_{max}}}{CPI_{exe}} \ . \tag{15}$$

The instructions retired serve as a measure of application performance in the model, where the performance loss δ_i of an application when executed on a processor clock-rate f_i, may be expressed as:

$$\delta_i = \frac{IR_{max} - IR_i}{IR_{max}} \ . \tag{16}$$

The proposed performance model can be applied to hardware platforms other than KNL by considering the throughput as a measure of performance and relating it to the operating clock-rate.

4.3.2 Power Consumption Model

The power consumption P_i of Xeon Phi at a clock-ratef_i can be expressed [96, 13] as:

$$P_i = P_{static} + k \times c \times f_i^3 \ , \tag{17}$$

where P_{static} is the static power consumption of Phi, c is the number of physical cores and k is the factor which varies as per the workload.

4.3.3 Power-Threshold Selection

Figure 24 displays the steps of the algorithm for selecting lower and higher power thresholds at the runtime. At a given f_i, the contents of the CPR register, holding CPI_{exe} values for the past few time-slices, and the value of k are obtained by solving Eq. (15)

for CPI_{exe} and from Eq. (17), respectively. The contents of the MPR register, holding the memory-access values corresponding to the CPI_{exe} CPR are initialized through the performance counters. Then, the value of CPI_{exe} and memory accesses for the next time-slice is determined through a history-window prediction mechanism [96] (in Step 4) by using an averaging function to predict the future value as an average of the past values. If the registers are not completely filled, then their last-assigned values are used for CPI_{exe} and N_{mem}, respectively, otherwise an average is taken and the oldest value in each register is discarded. In Step 6, the smallest operating clock-rate is determined that satisfies the performance loss constraint such that the performance loss does not exceed the user-defined performance loss γ. The values of the power limits are chosen in Step 7 such that to allow the power consumption to remain close to P_{limit}. The lower and higher power limits are separated by a difference proportional to the user defined performance loss. This accounts for any inaccuracies in the performance model by giving some headroom to the performance governor to stay above the power limit calculated in Step 6.

4.3.4 Experiments

The Sandia National Labs PowerAPI [55] is used to measure energy via the Linux Power Capping Framework (LPCF) [2] plugin which reads energy from the Running Average Power Limit (RAPL) [103, 20] counters. The PowerAPI uses the hardware locality (*hwloc*) API [79, 9] to detect the underlying hardware and is very portable, hence no modification to the API was required to measure energy for the KNL processor. The Linux Power Capping Framework is also used to update power thresholds, and "perf" [1] is used to measure hardware performance counters since PAPI does not yet support KNL

[41, 10]. Power and performance measurements are collected every 250ms to be used by the power threshold selection procedure. Measurements are collected for five seconds before the application is executed to establish the idle power draw, about 72 watts. Memory latency was measured as 390 cycles using LMBench. As per [110], the out-of-order execution overlap (a) can be calculated as calculated as 0.9 using the methodology discussed in [110].

$$\alpha = \frac{Reorder\ Buffer\ Length}{Issue\ Width \times Memory\ Latency}. \quad (18)$$

Plugging the relevant values of the parameters for the performance model obtained from [86] in Eq. (18), the value of a was calculated as 0.9.

4.3.5 Results

For all the executions, 252 threads (63 cores) are used, and one core is left for the measurement application so that the performance impact of measurement is minimized. (The use of "perf" to collect hardware performance counters significantly degrades performance if all 256 cores are allotted to the execution.) Presented results are based on the average execution time and average energy consumption based on five executions for each configuration: each application without measurement (denoted as *baseline*), with measurement and default power limits (denoted as *default*), with static power limits (denoted as *best power limit (BPL)*), and with the runtime power-threshold as per algorithm in Fig. 24 (denoted here as *model*).

For the offline analysis, power limits are set before the execution and do not vary throughout execution. For NPB and CoMD, power limits are varied between 90W–150W at 10W intervals and for GAMESS, power limits are varied between 80W–95W at 5W

intervals because only 64 threads may be used and max power draw is about 100W. The maximum power draw, BPL, and the average runtime power limits are shown in Fig. 25a. The average runtime power limits were measured during the 'model' executions. Maximum power was measured with the default power limits (215W), and the BPL is chosen based on energy saved with less than 10% performance loss and when the power limits are not equal to the default ones. A special case – BT – incurred more than 10% performance (see Fig. 25c, the darkest bars). As shown in Fig. 25a, none of the workloads were able to stress the KNL processor up to the TDP (215W). Notice also that the memory-intensive applications (FT, CG, and MG) benefit from power limiting more than the compute-intensive ones (EP, CoMD, and GAMESS).

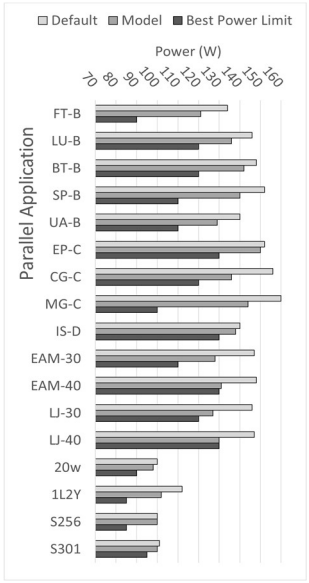
For the runtime method, power limits are dynamically set during execution, as in Fig. 24, where each timeslice duration τ is 250ms. Note that the time-to-solution for all applications is at least five seconds with CoMD and GAMESS executing longer than 20 seconds. Figure 25b presents the energy saved for each application relative to the *default* execution, while Fig. 25c shows the corresponding performance changes compared with the baseline execution. Note that the *default* execution of BT-B (i.e., just by measuring its power) 12% of the performance is lost, which is the most extreme negative effect seen in this work. All the other applications are within 10% of the performance loss for the *default* and *BPL* executions. For the runtime *model*, however, the performance losses, as recorded after the execution completion, often are higher. This is because they are only *approximated* at the runtime to be less than the specified value (10%), and calls for further fine-tuning of the *model*.

Input Parameters:
τ : Duration of the timeslice s.
γ : Performance-loss tolerance.
V : Number of timeslices.
$f_1, f_2, \ldots, f_N$: Available frequencies in Xeon Phi, where $f_i > f_j$ for $i < j$.
CPR and MPR : CPI_{exe} and N_{mem} registers of length $L = 3$.
c : index of CPR and MPR, initialized to 0.

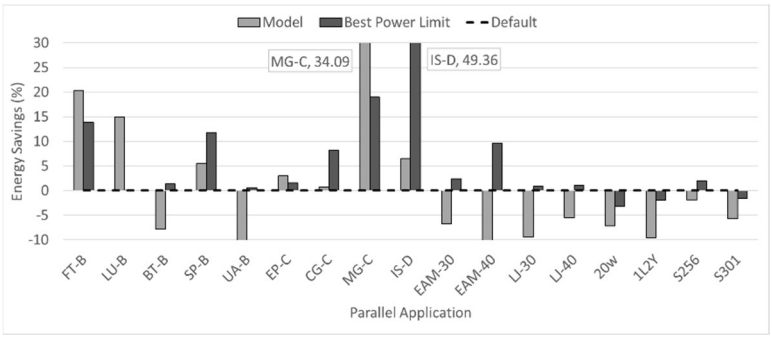
Algorithm:
Step 1. Execute application during the timeslice $s = 1$ at the highest clock-rate f_{max}.
Step 2. Determine IR_{max}, P_{max}, and MPR[c] at the end of first timeslice using hardware counters.
Step 3. Initialize CPR[c] and k using Eq. (15) and Eq. (17), respectively, for $f_i = f_{max}$.

For ($s = 2, s \leq V, s$++) **do:**
Step 4. Calculate CPI_{exe} and N_{mem}:
$c = c + 1$
If ($c \leq L - 1$) then
$CPI_{exe} =$ CPR[$c - 1$].
$N_{mem} =$ MPR[$c - 1$].
Else
$CPI_{exe} =$ avg(CPR);
$N_{mem} =$ avg(MPR);
Shift CPR and MPR to the left by one position
$c = c - 1$
Step 5. Calculate IR_i for all $i = 1, \ldots, N$ from Eq. (15).
Step 6. Determine the least operating clock-rate f_m from Eq. (16), such that $\delta_m \leq \gamma$
Step 7. Calculate $P_{limit} = P_{static} + k \times c \times f_m^3$
Step 8. Set the power limits as *lower*= P_{limit} $(1\text{-}\gamma)$ and *higher* = P_{limit} $(1 + \gamma)$
Step 9. Execute application for duration τ at the *lower* and *higher* power thresholds from Step 8.
Step 10. Determine IR_m, P_m, and MPR[c] at the end of timeslice s using hardware counters.
Step 11. For current f_m, calculate CPR[c] and k using Eq. (15) and Eq. (17), respectively.
EndFor

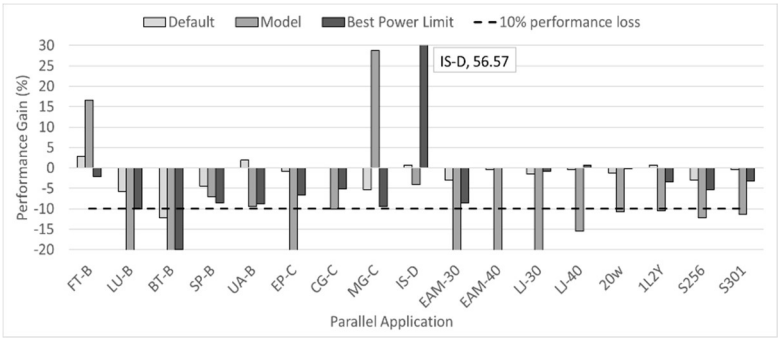
Fig. 24. Runtime power-threshold selection procedure for energy savings in Xeon Phi.



(a) Power Draw



(b) Energy Savings



(c) Performance Gain

Fig. 25. Baseline power draw (*Default*) compared to the online (*Model*) and offline (*Best Power Limit*) power limiting methods (a), energy savings (b), and performance gain (c) for each workload investigated. Energy savings and performance gain are compared to the *Default* measurements for each respective workload. In (c), the 10% performance-loss mark is indicated with the dashed line. The IS-D and MG-C bars are labeled with their actual values, which are beyond the scale of the plots.

For GAMESS, energy savings could not be obtained with the runtime *model* because varying voltage and clock-rate impacted both the compute and data threads on the Xeon Phi as opposed to CPU-only executions, in which compute and data processes may be

mapped to different cores with selective DVFS application in cores [90]. On the Xeon Phi, however, all the cores are subjected to the same power limits, thereby adversely affecting the compute processes.

For CoMD, the static analysis produced energy savings with a negligible performance impact. The most savings were found for EAM, 3% and 10% for the problem sizes 30 and 40, respectively. They may be explained by the energy-saving potential of the communication phase present in the middle of the EAM force computation (see authors' previous work [61]). The use of runtime procedure degraded the CoMD performance, however, and thus its energy consumption; hence, the need to refine the runtime analysis in the future work.

For NPB, energy savings were obtained mostly by memory-intensive applications: FT, LU, SP, MG, and IS. The most significant gains were obtained by FT, LU, and MG when the *model* was used, ranging from 15% to 30%. For FT and MG, such energy-savings are attributed to the performance gains found when decreasing clock-rate and voltage: The performance counters show that LLC misses were decreased for these executions, indicating that fewer memory misses increased execution performance. For LU, the energy savings were only obtained at the performance loss greater than 10%, which was not desirable.

Similarly, the compute-intensive EP obtained some energy savings with a large performance loss for the *model* execution CG, a memory-intensive application, obtained energy savings of 8% under BPL with a small performance loss, while under the runtime analysis larger performance loss yielded negligible energy savings. This is because the model does not include feedback to limit performance impact, which has been beneficial

on the CPU. The energy savings and performance gain obtained by IS may be attributed not only to the significant reduction in LLC misses with BPL, but also to the random memory accesses occurring in each execution: the standard deviation for its measurements was much higher (by about an order of magnitude) than that for the other applications. With the runtime method, however, only 7% of energy savings were obtained.

CHAPTER 5

HETEROGENEOUS EXECUTION-PHASE MODEL

One such method considered in this work is the *Execution Phase* model, presented in this chapter. The phase model leverages popular performance models, such as the Roofline and ping-pong communication models, with application specific measurements to estimate execution *phases*, i.e., computation and communication phases. In this chapter, a heterogeneous computing platform featuring a CPU and one or more accelerators is modeled. Computation on all devices is considered, as well as the communication between devices and among MPI domains.

In this chapter, the system characteristics are presented, followed by the derivation of the models with respect to the system. The experimental setup, results, and visualization of the method is then presented.

5.1 System Characteristics

A single-node heterogeneous architecture is composed of one multi-core host architecture and one or more multi-core accelerator architectures ($A_i \ i = 1, 2, \ldots, n_{\text{acc}}$), where n_{acc} is the number of accelerators. Note that such an architecture may contain accelerators of different types (e.g., Xeon Phi and GPU). Each accelerator is connected to the host and one another by the PCI bus, contains a two-level memory hierarchy (with slow and fast memories), and is a many-core processing unit. It is also assumed that the slow and fast memories are infinite and finite capacities, respectively, and that data must be moved between memories and processor (called resources) during application execution. The parallel application is assumed to employ a domain-decomposition scheme [36], which

is defined here as the division of a problem into sub-problems (called sub-domains) that are distributed among devices. Sub-domains may be computed in parallel, and may also require sharing data with neighboring sub-domains to solve the problem globally. It is assumed that data communication phase may not overlap computation phase. When executing an application, the total number of sub-domains is dependent both on the application and system configuration.

The distribution of sub-domains among resources is dependent on the execution mode: *device*, *offload*, or *symmetric*. For execution exclusively on the device, all work and data movement use only the resources of that particular device. For symmetric execution, sub-domains are distributed among the hosts and accelerators, serving as peers. For *offload*, on the other hand, the computations are performed either on the host or accelerators, such that one host sub-domain is shared with one accelerator only. In other words, each sub-domain resides on the host while portions of its computational phase and data are copied to the accelerator for processing and the result returned to the host. It is assumed that host and accelerator computations do not overlap, i.e., one is idle while the other computes. The communications among sub-domain performed only by the host(s) while leaving the corresponding accelerators idle.

5.2 Derivation of the Execution Phases

This section presents the time, power, and energy models used to describe each of the phases considered, with distinctions made for the hardware influencing the model (e.g. throughput).

5.2.1 Execution Time

The execution of an application that employs domain-decomposition may be described as having the following four phases: *initialize*, *compute*, *communicate*, and *output*. The initialization phase sets up a problem to be solved, and the output phase relays important statistics and output upon completion. Solving each sub-problem requires an iterative pattern of computation and communication phases until a global solution is achieved. Note that the initialization and output phases are not modeled because they are expected to have little impact on the overall performance for large-scale problems with multiple nodes.

The total execution time in the offload mode may modeled as:

$$T = T_{comp} + T_{comm}\,, \tag{19}$$

Where T is the sum of the times required to perform all the computations and communications, respectively.

5.2.1.1 Computation Phase

The total computation time is limited by the slowest time required to solve a sub-domain for a given execution mode. It is equivalent to being limited by the total time of a particular execution mode. Computation may be simply defined by the slowest execution mode because sub-domains of similar execution modes will require relatively the same time to solve; however sub-domains of differing execution modes may be vastly different, depending on load balance and the implementation. For the model, all sub-domains of similar execution modes are equivalent considering each would be modeled using the same parameters.

The total time to compute in offload mode T_{comp} is:

$$T_{comp} = T_{host} + T_{acc} + T_{pci} . \quad (20)$$

It is defined by the execution time for the host T_{host}, accelerator T_{acc}, and communication time across the PCI bus T_{pci}. The time T_{hos} to compute on the host is defined using the time-frequency model [16, 64, 95] as:

$$T_{host} = t_{on} \times \frac{f_{max}}{f} + t_{off} , \quad (21)$$

where t_{on} is the time on-chip, t_{off} is the time off-chip, and f is the operational frequency during execution for the device such that $f \neq f_{max}$. This general equation is used simply to estimate host execution time and deduce the applications computational intensity on the host. The time T_{acc} to compute on the accelerator is defined using the roofline model [107, 15] :

$$T_{acc} = \max(W_{acc} \times \tau_w, M_{acc} \times \tau_m) \quad (22)$$

which is the maximum of the time to perform work W_{acc} and time to move data M_{acc} between memories with τ_w and τ_m being the times to perform a unit of work and to transfer a unit of data, respectively. The time T_{pci} to move data across the PCI bus is:

$$T_{pci} = M_{pci} \times \tau_{pci} , \quad (23)$$

which is the product of the amount of data M_{pci} to be moved and the time per data movement τ_{pci} across the PCI bus.

5.2.1.2 Communication Phase

Total communication time T_{comm} is limited by the slowest transfer between sub-domains, and it may be simply limited by the slowest communication type. For offload

execution, there are two communication types to consider: transfers between sub-domains on the same node (called intra-node), and transfers between sub-domains on differing nodes (called inter-node). These two communication types may overlap. For configurations executed on one node, the intra-node communication model is used; and for multiple nodes the inter-node communication model is used.

Intra-node communication times T_{comm} may be defined as:

$$T_{comm} = M_{comm} \times \tau_{comm}\,, \tag{24}$$

where M_{comm} is the amount of data to be moved and τ_{comm} the time required to move a unit of data. Inter-node communication time T_{comm} is:

$$T_{comm} = t_l + M_{comm} \times \tau_{comm}\,, \tag{25}$$

where t_l is the network latency. Note that, for a single-node configuration, network latency time is not present in Eq. (24).

5.2.1.3 Throughput

The time τ_w to perform unit of work is computed by taking the inverse of throughput. The definition of throughput is generally the total number of cores performing work times the frequency per core. However, for devices, such as the Xeon Phi, throughput also depends on characteristics such as vectorization intensity [11] and operations per cycle:

$$\tau_w = \left(c \times n_{ops} \times VI \times f\right)^{-1} \tag{26}$$

where number of cores c includes only those used in the computation, f is the device clock-rate, the number of operations per cycle n_{ops} is a value between one and two representing an average of single and fused multiply-add operations performed, and VI is the

vectorization intensity, which is a measure of the number of SIMD instructions issued. For the Intel Xeon Phi, VI may be a value between one and eight for double-precision floating-point operations. Note that Eq. (26) is applicable to all the Intel devices based on the Sandy-Bridge or newer microarchitectures.

5.2.2 Power and Energy

The total power draw P for the system is the sum of the power draw for each device; the total number of devices is n_{dev}, and power is defined as:

$$P = \sum^{n_{dev}} P_{dev} \,. \tag{27}$$

Device power is defined as static and dynamic power; however dynamic power may fluctuate during execution depending on whether the device is idle or active. A device is considered active if performing computation, and otherwise is idle (that is to include all communications). Device power may be defined using the weighted sum of the power draw for each execution state:

$$P_{dev} = P_{active} \times \frac{T_{active}}{T} + P_{idle} \times \frac{T_{idle}}{T}, \tag{28}$$

where the total execution time $T = T_{activ} + T_{idle}$ and

$$P_{state} = P_{static} + k \times c \times f^3 \tag{29}$$

is the power draw for a given state and depends on the static power draw P_{static}, a power constant k, the number of cores for the device c, and the state clock-rate f (see, e.g., [95] and the references therein).

From Eqs. (19), (20) and (27), energy may be defined as:

$$E = E_{host} + E_{acc} + E_{pci} + E_{comm}\,, \tag{30}$$

where the energies E_{hos} , E_{acc}, and E_{pci}correspond to the three terms of Eq. (20), respectively, and E_{comm} is obtained using either Eq. (24) or Eq. (25) for single- or multi-node executions, respectively.

5.3 Experiment

Measurements have been collected on the Borges and Bolt systems using CPU + Xeon Phi offload. The measured energy is averaged over five runs for each experiment. For the Borges system, only two configurations are investigated, termed *MIC 1* and *MIC 2*, corresponding to employing only one or both Xeon Phi devices, respectively. On Bolt, six configurations are investigated, termed *N1 MIC 1*, *N1 MIC 2*, *N2 MIC 1*, *N2 MIC 2*, *N3 MIC 1*, and *N3 MIC 2*, where *N1*, *N2*, and *N3* correspond to one, two, and three nodes used to run CoMD. For each configuration, the host frequency, number of Xeon Phi (MIC) threads, and model problem size were varied as follows:

- All ten power states were considered on Borges (from 1.2 to 2.001 GHz with the 100- MHz stepping). On Bolt, only were seven (3.201, 3.2, 2.8, 2.3, 1.9, 1.5, and 1.2 GHz) out of sixteen possible states (from 1.2 to 3.201 GHz with variable stepping) chosen to make the number of measured configuration instances manageable while still having enough data to fit the models.
- Seven MIC-thread values ranging from 120 to 236 (four threads per core) were taken to execute CoMD. Note, that, since one core is always occupied by four threads dedicated to operating system tasks, such as servicing the offload daemon and to avoid thread over-

subscription, the maximum of 236 application threads is reasonable to use on the 60-core Xeon Phi considered here.

- Although problem sizes of 50, 60, 70, and 80 are explored to observe the computational intensity of CoMD for given platforms in this work, all the results presented here are for the problem size of 50 only. Executions with the other problem sizes exhibited similar behavior but took significantly longer to complete.
- The *compact* thread affinity and thread-level granularity were used on accelerator devices since they were found to perform better in [61, 62].

The relative error between the measured energy $E_{measured}$ and $E_{modeled}$, modeled by Eq. (30), is calculated as:

$$100\times\frac{E_{modeled}-E_{measured}}{E_{measured}}. \tag{31}$$

Examples of error quantification are featured in Figs. 26 and 27 for Borges and Bolt systems, respectively, where each configuration is considered for all the chosen MIC-thread and host-frequency values. For Borges—as seen in Fig. 26–the majority of configuration instances are modeled with no more than 5% of error. Note that only does *MIC 1* at the frequency of 2.001 and lower MIC-thread values appear outside the 5% error range but is still confined within the 10% threshold. In Fig. 27, the relative error is also confined in the 10% interval with the *MIC 1* configurations showing a slightly better prediction accuracy, in general.

5.3.1 Execution Time

Table VIII presents the execution-time model parameters for each configuration (column `Config`) of the CoMD and the two systems (column `System`) as calculated

from Eqs. (21) to (24) for the host compute time T_{host}, host communication time T_{comm}, Xeon Phi compute time T_{acc}, and PCI transfer time T_{pci}. The model parameters have been estimated using linear regression over a sample set of configuration instances with varying host frequencies and MIC thread counts. For detailed procedure to estimate parameters in Table VIII see [65]. The column n_{sub} lists the number of sub-domains for each configuration.

The host computation time (column-set `Host Comp`) is modeled as a sum of on- and off-chip times (columns t_{on} and t_{off}) using linear regression while varying host frequencies in Eq. (21). The coefficient R^2 was 0.99 throughout this estimation for all the configurations; thereby, indicating a high accuracy of the model. Note that for both systems, the ratio of t_{off} to t_{on} (column $t_{of}\ /t_{on}$) is less than one for all cases, which shows that the host computation is compute, rather than memory, intensive. Recall that in CoMD, the host updates atom position, velocity, and (when needed) redistribution of atoms. The host communication time (column-set `Host Comm`) is measured and used in Eq. (24) to obtain the (inter-host) communication latency t_l by reading the hardware counters for the amount of data transferred M_{comm} and the transfer bandwidth $1/\tau_{comm}$.

The workload on the Xeon Phi (column W_{acc} in the column-set `MIC Comp`) is estimated by varying the MIC thread count and observing its effect on the execution time. The parameters M_{acc} and τ_M were estimated similarly to those for T_{comm}. Note that, although both systems use the same model (5110p) of the Intel Xeon Phi, the estimated workloads are somewhat different for the same configurations (e.g., cf. 4.12 and 4.46 TFLOPs on Borges and Bolt, respectively, for *MIC 1*), which is expected because the W_{acc} values were modeled rather than calculated by counting all the algorithmic operations.

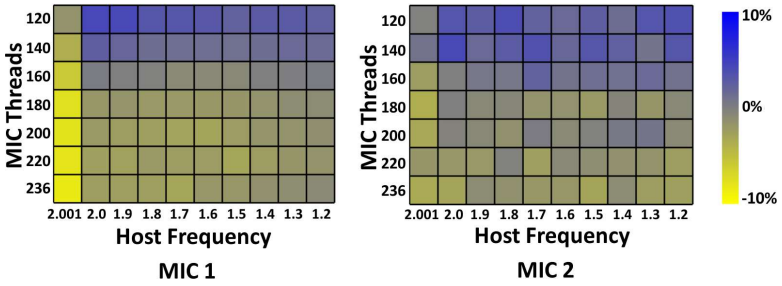


Fig. 26. Relative energy-model error per Eq. (30) on Borges.

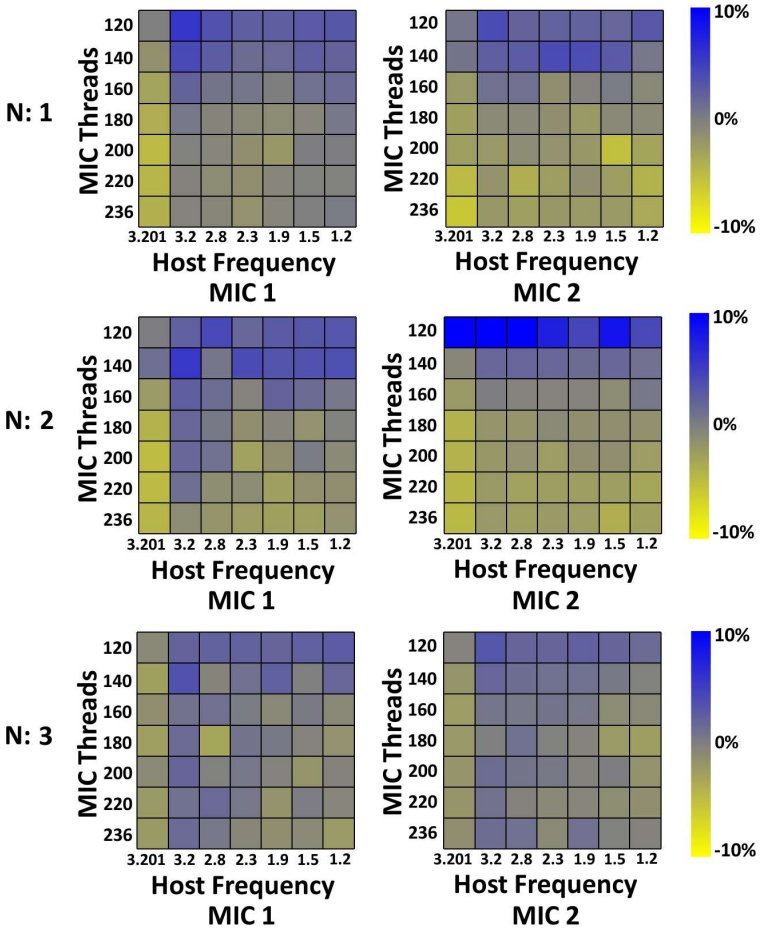


Fig. 27. Relative energy-model error per Eq. (30) on Bolt.

TABLE VIII

EXECUTION-TIME MODEL PARAMETERS FOR BORGES AND BOLT

			Host Comp T_{host}			Host Comm T_{comm}			MIC Comp T_{acc}			PCI T_{pci}	
System	Config	n_{sub}	t_{on} (s)	t_{off} (s)	$\frac{t_{\text{off}}}{t_{\text{on}}}$	t_l	M_{comm} (MB)	$1/\tau_{\text{comm}}$ (MB/s)	W_{acc} (TFLOPs)	M_{acc} (MB)	$1/\tau_M$ (MB/s)	M_{pci} (GB)	$1/\tau_{\text{pci}}$ (GB/s)
Borges	MIC 1	1	1.62	0.23	0.14	0	59.17	450.51	4.12	83.5	2.48	5.33	4.61
	MIC 2	2	0.99	0.06	0.06	0	67.1	514.97	2.16	36.1	2.10	2.81	3.78
Bolt	N1 MIC 1	1	1.21	0.30	0.25	0	63.26	564.18	4.46	84.18	2.44	5.64	2.83
	N1 MIC 2	2	0.93	0.19	0.20	0	56.23	549.98	2.32	36.13	2.04	2.97	2.51
	N2 MIC 1	2	0.63	0.12	0.19	0.31	52.39	569.27	2.33	36.06	2.03	2.97	2.23
	N2 MIC 2	4	0.46	0.13	0.28	0.63	51.66	553.03	1.14	10.33	1.14	1.56	0.62
	N3 MIC 1	3	0.43	0.06	0.14	0.23	49.14	580.15	1.60	19.52	1.61	2.08	1.93
	N3 MIC 2	6	0.32	0.05	0.16	0.80	51.02	572.12	0.79	5.80	0.94	1.09	0.44

TABLE IX

DEVICE POWER-MODEL PARAMETERS FOUND FOR THE BORGES AND BOLT SYSTEMS

Device	System	Config	Active Power P_{active}, W					Idle Power P_{idle}, W					Total Power P, W	
			P_{static}	ρ	P_{max}	T_{active}/T	R^2	P_{static}	ρ	P_{max}	T_{idle}/T	R^2	P_{static}	P_{max}
Xeon Phi	Borges	MIC 1	168.39	1.05	241.95	0.88	0.99	170.07	0.95	236.62	0.12	0.95	168.59	241.31
		MIC 2	169.91	0.93	235.06	0.84	0.97	171.24	0.79	226.58	0.16	0.80	170.12	233.70
	Bolt	MIC 1	161.69	1.26	249.96	0.87	0.99	160.51	1.28	250.18	0.13	0.99	161.54	249.99
		MIC 2	162.51	1.17	244.47	0.84	0.99	168.37	1.07	243.33	0.16	0.84	163.45	244.29
Host CPU	Borges	MIC 1	24.30	0.28	60.19	0.07	0.85	24.22	0.28	60.11	0.93	0.85	24.23	60.12
		MIC 2	13.11	0.11	27.21	0.08	0.99	13.11	0.11	27.20	0.92	0.99	13.11	27.21
	Bolt	MIC 1	30.60	0.22	73.89	0.04	0.99	29.03	0.18	64.45	0.96	0.91	29.09	64.83
		MIC 2	25.21	0.12	48.83	0.08	0.96	25.82	0.12	49.44	0.92	0.98	25.77	49.39

TABLE X

MINIMUM MEASURED AND MODELED ENERGY FOR BORGES AND BOLT

System	Type	N	n_{sub}	Config Instance			Min Energy (J)	MIC Comp (GFLOPs/J)	MIC Mem (MB/J)	Host Comm (MB/J)	PCI (MB/J)
				# MIC	Freq	Thread					
Borges	Measured	1	2	2	1.3	236	6436.6	0.81	0.013	0.69	16.77
	Modeled	1	2	2	1.9	236	6583.2	0.77	0.013	1.03	15.62
Bolt	Measured	1	2	2	2.3	236	*8112.1*	0.70	0.011	0.51	8.27
	Modeled	1	2	2	2.6	236	7986.8	0.70	0.011	1.04	8.04
	Measured	2	2	1	1.5	236	8786.1	0.34	0.005	0.22	3.59
	Modeled	2	2	1	1.8	236	8725.8	0.33	0.005	0.23	3.96
	Measured	3	3	1	1.5	236	8985.6	0.23	0.003	0.20	2.10
	Modeled	3	3	1	1.8	236	8980.4	0.22	0.003	0.19	2.32

The differences (amounting to about 8%) may be due, for example, to different numbers of co-processor stalls incurred during memory operations in each system, which are not accounted for explicitly in the proposed model. It is also interesting to observe that the speed-up with respect to the number of sub-domains n_{sub} is very good in all the configurations with its lowest gains of 2.78 on three sub-domains and highest gains of 1.92 on two sub-domains of Bolt. Hence, one may infer that, even this non-optimized version of CoMD with *VI* of only 2.6 and with a moderate problem size of 50 scales well to multiple accelerators, either attached to a single node or to multiple nodes (cf. W_{acc} for *N1 MIC 2* and *N2 MIC 1* configurations, which have the same n_{sub} on Bolt).

The amount of data transferred over PCI (column M_{pci} from the column-set `PCI`) was read directly from the offload output reports, which most likely estimate this value since it appears to differ somewhat across the systems. The peak PCI bandwidth is 8 GB/s (PCI Express 2.0 x16) for Xeon Phi. As seen in Table VIII, the configurations with fewer sub-domains (thus, more data to transfer per sub-domain), the bandwidth $1/\tau_{pci}$ reaches almost 3 and 5 GB/s on Bolt and Borges, respectively, which indicates that PCI transfers are well-optimized in offload executions.

5.3.2 Power

Table IX presents the obtained power-model parameters for a Xeon Phi accelerator and the host CPU (column `Device`) when a given configuration (column `Config`) isconsidered on a single node of Borges or Bolt (column `System`). The parameters were calculated using linear regression over the configurations instances, in which different execution states, active or idle, were distinguished (column-sets `Active`

`Power` and `Idle Power`, respectively). The samples are obtained from varying host frequency and MIC threads, for the host and Xeon-Phi device power models, respectively, obtained by Eq. (29). If the maximum frequency is used, then the maximum power draw P_{max} may be predicted for each state, as presented in Table IX . The total power P (column-set `Total Power`) is then calculated following Eq. (28).

The model accuracy is assessed by the R^2 term. All the models appear well-correlated because the obtained R^2 coefficients, also provided in Table IX, are all close to one. For each device and configuration, a ratio of its active (idle) time to the total time is provided in column T_{active}/T (T_{idle}/T). Note that the sum of these two ratios is one. For a give device, the times T_{active} and T_{idle} mutually exclusive with an active-state time being equal to the device computation time, T_{acc} for the Xeon Phi and T_{host} for the host CPU. During any type of communications (host-only or PCI), both host and Xeon Phi are assumed to be idle since communication/computation overlap is not considered in this work.

Different amounts of idle time may indicate varying benefits of CPU-based dynamic voltage and frequency scaling (DVFS) for each device. For example, as seen in Table IX for Borges and the *MIC 1* configuration, the Xeon Phi idles only for 12% of the execution time, so its DVFS potential may be rather small compared with that on the host, which is 93% for the same system and configuration. Even though the power draw is significantly lower on the host, the offloaded portions of the execution are ideal places to save energy by DVFS, as the authors have previously concluded [63], along with communication phases (see, e.g., [95]).

It may be observed from Table X, that the maximum host power draw for either

system is higher when only one accelerator is used (cf. lines 5 and 6; lines 7 and 8) because with two accelerators the host works on two smaller sub-domains in parallel. Using two sub-domains requires less computational intensity than one larger sub-domain does with only one accelerator in use. Observe also, that the *MIC 1* configurations required more total power than the *MIC 2* ones. Hence, using more than one accelerator may be beneficial not only to reduce execution time but also to expect less of a power surge in compute-intensive applications.

5.3.3 Energy

The proposed model aims to predict the *best* configuration instance defined as the one with the lowest energy consumption. For each system and the corresponding node count (column `N`), Table X presents configuration instances consuming (column-set `Config Instance`) the minimum amount of energy (column `Min Energy`), which is either measured or modeled (column `Type`). Since the problem size is fixed, each configuration instance shown in Table X is determined by the number of Xeon Phi's used (column `# MIC`), host frequency (column `Freq`) in GHz, and the number of threads per Xeon Phi (column `Thread`). The measured energy values are averaged over five runs for a given configuration instance.

As seen in Table X, the models are able to predict a configuration close to the one obtained experimentally. Specifically, in each case, all the configuration-instance parameters matched except that the host frequency was consistently over-predicted by 0.3 GHz throughout but on Borges, where the predicted frequency was 0.6 GHz higher. Overall, for either Borges or Bolt, the best configuration instances were those using a single

node with two Xeon Phi since the entire problem of size 50 fits on a single node. For larger problems, when only are multi-node systems employed efficiently, the best configuration instances will include multiple node.

To further verify the minimum energy estimations obtained for a single node `N` of Bolt, all the 16 available frequencies were examined for 220 and 236 thread counts and the corresponding energies measured. Their values, however, were always higher than that shown as italicized in column `Min Energy` of Table X. It may be observed also that, for multi-node executions, one-MIC configurations are better at large MIC-thread counts because the host frequency may be significantly reduced (cf. measured 2.3 and 1.5 GHz on Bolt for two sub-domains in column n_{sub}) to compute sub-domains distributed to multiple nodes rather than shared by a single node, considering that the total execution time is similar in these cases (cf. $T_{host} + T_{comm}$ equal to 1.12 and 1.06 seconds for *N1 MIC 2* and *N2 MIC 1*, respectively, in Table VIII), it may be inferred that an instance of *N2 MIC 1* is more energy efficient, as confirmed in column `Min Energy`.

The number of floating-point operations per joule as well as bytes per joule are typically used to assess architecture performance with respect to energy consumption (see [14, 15]). These metrics are modeled using the parameters from Tables VIII and IX using Eq. (30) and are provided in columns 9–12 of Table X, such that columns `MIC Comp`, `MIC Mem`, `Host Comm`, and `PCI` correspond to MIC-only computations and memory accesses, data movement in host communications and PCI transfers, respectively. The modeled values in each column were calculated from Eq. (26) as follows:

`MIC Comp`	$n_{\text{acc}} \cdot W_{\text{acc}} / E_{\text{acc}}$,
`MIC Mem`	$n_{\text{acc}} \cdot M_{\text{acc}} / E_{\text{acc}}$, N
`Host Comm`	$\cdot M_{\text{comm}} / E_{\text{comm}}$,
`PCI`	$n_{\text{acc}} \cdot M_{\text{pci}} / E_{\text{pci}}$.

5.4 Visualizing Phase for Heterogeneous Executions

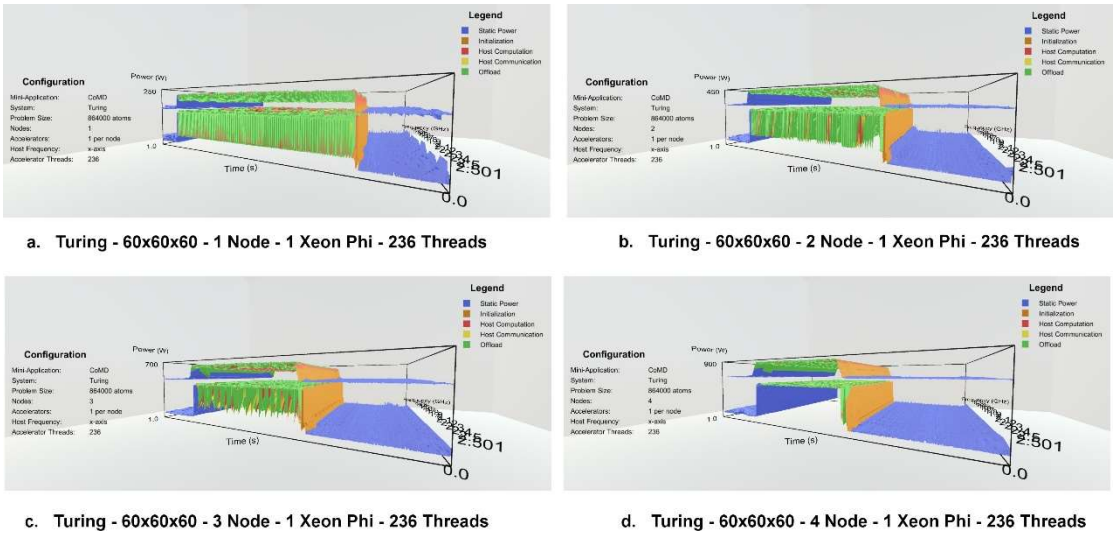
To better visualize power with respect to time for different configurations, it has been decided to employ the so-called *waterfall plots* [82]. Although software exists to render these plots, such as Mathematica or Matlab, rendering is slow and thus too cumbersome to use as a tool. Instead, Unity, a software used for game development on many platforms [100], has been adopted to render the plots. With Unity, a tool has been created by the authors to read simple input files (containing only a 2D array of power data and phase labels) and to automatically render a corresponding plot.

The three axes composing each waterfall plot are execution configuration (x-axis), power draw (y-axis), and execution time (z-axis); in Unity, the y-axis is the vertical axis by default. The input power and phase data are used to create a mesh: one mesh is created for the host and another for the accelerator, and, in cases of multiple host or accelerator devices, the power is summed. Recall that the application execution phases are static power draw, initialization, host computation, host communication, and offload phase. The offload phase contains both accelerator computation and data transfer phases. The created waterfall plots are meant to be viewed in color and from multiple viewpoints. Hence, the authors have published a webpage, available at [57], where any user may explore the plots firsthand.

Figure 28 shows the waterfall plots for 1–4 node configurations with 236 threads, one Xeon Phi, and frequency from minimum to maximum in each plot. The colors illustrate the phase transitions that occurred in each execution, and show the relative timing of phases across different executions. The data for the plots in Fig. 28 come from the authors' work [66]. One may observe across all four plots in Fig. 28 that the phase power draw differences diminish as the number of devices used increases; in (a), power draw is very jagged and fluctuates highly between large and small power draws, but in (d), power draw is relatively smooth and thus fewer fluctuations are perceived. This shows that there is significant overlap among phases executed on differing nodes which impacts the resulting power draw of the system. On a per node basis, power draw can be expected to fluctuate rapidly, but from the viewpoint of the system, power draw appears fairly consistent.

Figure 28 may also be used to show the power draw for each execution phase. In color, the plots are mostly green; green is used to show when power draw has been sampled during the offload execution phase. This is especially true for the host. Most of the high power readings on the host are colored green, although one would think that power would be low because the host is not actively computing. This shows that the host is moving data, but must utilize the core to handle the instructions. From the authors previous work [63], minimizing frequency during offload execution reduced power draw. However, from the waterfall plots in Fig. 28, it may be concluded that there is a potential for energy savings during offload through host DVFS. Alternatively, the host may be scheduled to perform computations during the offload phase.

Figure 29 shows a waterfall plot with data collected using the micmgmt API measurements. Note the power draw differences between Fig. 28 and Fig. 29: The power



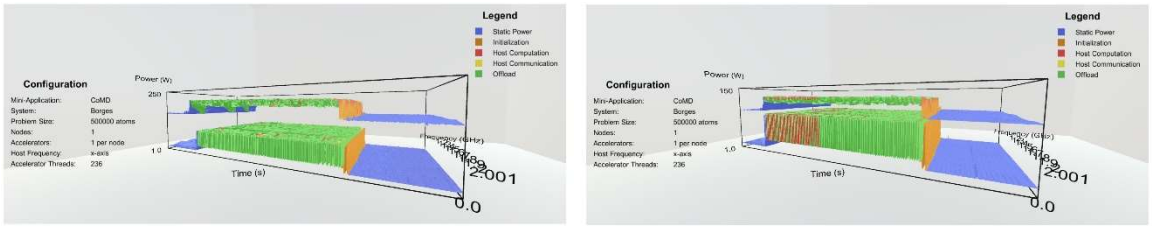
a. Turing - 60x60x60 - 1 Node - 1 Xeon Phi - 236 Threads

b. Turing - 60x60x60 - 2 Node - 1 Xeon Phi - 236 Threads

c. Turing - 60x60x60 - 3 Node - 1 Xeon Phi - 236 Threads

d. Turing - 60x60x60 - 4 Node - 1 Xeon Phi - 236 Threads

Fig. 28. Waterfall plots for Turing with 1–4 nodes (a–d), each with 1 Xeon Phi, 236 threads with a CoMD problem size of 60 (864,000 atoms).



a. Measured with "/sys/class/micras/power" file

b. Measured with micmgmt API

Fig. 29. Waterfall plot on Borges with 1 node, 1 Xeon Phi, 236 threads with a CoMD problem size of 50 (500,000 atoms). On the left (a), executions were measured using the old method, and on the right (b), measured using the new method with micmgmt API.

per Xeon Phi does not exceed 150 W with the micmgmt API measurements (Fig. 29). All the plots have been generated using power data collected by reading the connector sensors (PCI, 2x3, and 2x4) which should yield a maximum power draw of 300 W. The device TDP is 245 W, recall that this is the average power draw while the device is in the C0 (active) state; therefore, a power draw of over 200 W was the expected result, yet power draw nears only 130 W. This shows there is room for optimization for this application on this hardware.

5.5 Conclusions

In this chapter, the execution phase model was presented. The models used here were successfully fit to CoMD on two different heterogeneous platforms with Xeon Phi, and the results were visualized using the Unity Game engine. The model was able to capture the execution trends identified by the models, although the method for collecting this data is not extendable to all applications. The method relies on quantifying each execution phase using timers, such that the data can be correlated with power measurements and thus execution phases may be determined. Although this approached worked for CoMD, more complex applications (such as GAMESS) are not easily dissected, especially since GAMESS offers many more execution kernels than CoMD. Further, this measurement approach incurred additional performance penalties due to writing data to the output file (write operations to the hard disk), which made the method impractical for general use. Thus, a different approach is needed to model applications and hardware.

CHAPTER 6

PREDICTING ENERGY CONSUMPTION

Energy consumption is a major concern for HPC systems because energy is a measure of operating costs – power draw is expensive, and applications take time to execute. Two techniques have emerged to control power and energy consumption for HPC systems: power and energy capping [88]. Power capping is used to limit the amount of power that the system may consume while executing a workload. Energy capping is used to limit the amount of energy that may be consumed to execute a workload. The difference between the two techniques is subtle; power capping limits the peak power draw and may consequently cause execution performance to decrease, whereas energy capping limits the integral of power over time but does not limit instantaneous power draw and thus does not reduce execution performance.

Predicting energy consumption can be tricky. Application workloads are highly variable, especially for more complex algorithms, thus power draw varies over the execution. Further, not all compute resources respond the same to a given workload, i.e., certain nodes will consume more power than others given the same workload; this is also true of time / performance. Finally, power draw does not necessarily reflect the state of the execution, e.g., computing or moving data, for several reasons. (1) Power draw depends on the efficiency of an operation – the larger the delay between requesting data and operating on it can potentially reduce the power draw. (2) Computations and data movement overlap which makes identifying particular operations difficult. (3) Multiple cores may share a power plane, thus the actual power consumption may be higher than that required by the workload because of this limitation (power will be the max required of all cores on the

plane).

To demonstrate several methods for predicting energy consumption, consider the following analysis of several parallel applications (CoMD and NPB) of which problem size may be defined. The ECM model is considered here because performance is defined by problem size (in FLOP's), the number of cores, and clock-rate; these application and system parameters are most commonly adjusted to determine the optimal execution. Recall from Chapter 2 the definition for the ECM performance model (Eq. (3)); for reference, it is presented again below:

$$T = \min\left(cN_0\left(1 + \frac{f_0 - f}{f_0}\right), N_{max}\right)^{-1} \tag{32}$$

where N_0 is the performance in FLOPs for one core, N_{max} is the maximum achievable performance given all bottlenecks, c is the number of cores, f is the current clock-rate, and f_0 is the baseline clock-rate. The ECM power model is:

$$P = W_0 + c(W_1 f + W_2 f^2), \tag{33}$$

where W0 is static power draw, W1 is the coefficient for the linear term of power draw, and W2 is the quadratic term.

The ECM performance model has been fit to measurements collected while running each application with several small problem sizes. Power is not modeled here, since average power varies little with problem size, and predictions are verified against large problems executed with only the maximum number of cores and clock-rate. Instead, the maximum power measurement obtained from the small problem sizes will be used to estimate the power draw for a larger problem size.

The results contained in this chapter are obtained using *two* definitions of problem

size: (1) “problem size” refers to the input problem size used for each application, and (2) “problem size per core” refers to the ECM-defined problem size (see Eq. (32)) which scales the input problem size by the number of cores and clock-rate. Also note here that the performance bottleneck term defined in the ECM performance equation is ignored for the clarity of exposition.

6.1 Problem Size Definitions

The NAS Parallel Benchmark Suite contains many different benchmarks for which problem size relates to inputs, such as the number of iterations or grid size. Problem size for the NPB can be found by reviewing the source code for the computation of millions of flops, a metric output by each benchmark. The problem size definitions for CoMD and the NPB suite are presented below.

6.1.1 NPB Problem Size

Here the definitions for problem sizes are presented for the CG and LU benchmarks for the NPB benchmarks. Only these two are shown here for brevity, however, the methods contained in this chapter may be applied to all of the benchmarks as will be shown in this chapter. The NPB benchmarks all estimate problem size in unit FLOPs.

For CG, problem size is defined as:

$$NZZ = NZ \times (NZ + 1)\,, \tag{34}$$

$$N_{flop}^{CG} = 2 \times I \times R \times \Big(6 + NZZ + \big(25 \times (5 + NZZ)\big)\Big)\,, \tag{35}$$

where I represents iterations, R is number of rows, and NZ is the number of non-zeros

per row. This function can be found in the NPB source code for CG. Performance modeling of CG has been conducted in the literature [91, 21], although the version provided by the NPB is a modern implementation of the algorithm.

For LU, problem size is defined as:

$$N_{flop}^{LU} = I \times (1984.77 \times NX \times NY \times NZ - 10923.3 \times \left(\frac{NX + NY + NZ}{3}\right)^2 + 27770.9 \times \left(\frac{NX + NY + NZ}{3}\right) - 144010) \quad (36)$$

where I represents iterations, and NX, NY, and NZ represents the grid size. What is interesting here is that the model for N_{flop}^{LU} is based on a regression model and not on an approximate computation of the total FLOPs, thus Eq. (36) may be subject to error when tested on machines other than that used to obtain it.

6.1.2 CoMD Problem Size

For CoMD, the total number of atoms represents problem size, and is defined as:

$$N_{flop}^{CoMD} = 4 \times I \times Nx \times Ny \times Nz \quad (37)$$

where I again represents iterations, and Nx, Ny, Nz represents the number of atoms in the x, y, and z-directions respectively. It is more difficult to approximate the number of FLOPs associated with this algorithm since the number of atoms involved per iteration depends on the number of atoms within a cutoff distance. Further, the computation may involve an interpolation which the number of FLOPs is not clearly defined. However, supplementing the number of atoms for FLOPs provides the linear model with a

sufficient definition of problem size for this investigation.

6.2 Measurements

Figures 30 and 31 present the measured energy, time, and power for CoMD (EAM and LJ) and NPB (CG and LU), respectively, for two platforms: Borges (Sandy-Bridge) and Marquez (Haswell). Energy and time are plotted against the problem size per core according to the ECM performance model Eq. (32). Power is plotted against clock-rate in GHz as in the ECM power model Eq. (33). Notice that the linear ECM model has been fit to the time measurements, represented as a solid line.

For CoMD, three problem sizes are tested, 25.6, 50.0, & 86.4 million atoms, and for CG and LU, two problem sizes are tested, classes B and C, while varying clock-rate and number of cores. Because strong scaling has been used while varying the number of cores, the data has been divided into two general problem sizes / core, noted as `PS1` and `PS2`, where `PS2` > `PS1` by a factor of 2. Note that the problem size / core includes all problem sizes considered for each application (25.6-86.4 mil atoms for CoMD and classes B and C for NPB).

Clock-rate has been included in the measurements because it can be used to identify the computational- or memory-boundedness of an application [16], and so it is useful to define whether the application scales well with problem size (upward linear trend between all problem sizes) or if the application encounters bottlenecks (horizontal linear trend between local problem sizes). This may be observed in Fig. 31c where the linear trend of the ECM model does not accurately reflect the trend of the measurements (see `PS2`). In this situation, an underlying performance bottleneck has been encountered, although this is not

predicted by the model since data movement (between cache and DRAM) is not modeled here.

Power is not modeled here, although the quadratic trend may be observed in Figs. 30 and 31; the trend is less obvious for the NPB measurements since class B is much smaller than class C which distorts the trend (e.g. two parallel quadratic trends are shown in Fig. 31c for both `PS1`and `PS2`). Energy is to be predicted, and so the measured results are shown here for reference. Measured energy has been calculated as:

$$E(t) = \int_0^T P(t) \times \Delta t \, , \tag{38}$$

where $E(t)$ is energy, T is execution time, $P(t)$ is power at time t, and Δt is the sampling rate (5ms).

6.3 Predictions

Figures 32 and 33 presents the predicted and measured energy, time, and power for large problem sizes; for CoMD, problem sizes 400 million-3200 million atoms are considered, and for NPB, class D is considered. Predicted time and energy is shown as a solid line. The average power draw used to predict energy is shown in Figs. 32 and 33 using a black 'X'. The prediction results here only investigate `PS1` at maximum clock-rate because measurements were only collected for this configuration (16 cores @ max clock-rate). Power measurements for the larger problem sizes is shown as a blue circle, and the small power measurements from Figs. 30 and 31 are shown as orange squares for reference.

Tables XI and XII presents the time, power, and energy values obtained from measuring and predicting the larger problem size considered for each application (3200

million atoms and class D), shown in Figs. 32 and 33, respectively, for CoMD and NPB. Predicted energy has been calculated as:

$$E = T \times P \,. \tag{39}$$

Negative error shows an under-estimation with respect to the measured result, and a positive error shows an over-estimation.

First consider power in Figs. 32 and 33 and Tables XI and XII. The maximum average power draw (labeled 'Model' in the figures) is used to predict energy consumed; notice that the selected power draw is close to that measured for the larger problem sizes (max error 7%).

Time is less accurate, with error ranging from 6%–60%. Inaccuracy of the time model is the result of two factors: (1) the ECM performance model is based on small problem sizes, and (2) the performance bottleneck is not included in the model fit since data movement is not defined for the applications considered. Error in energy consumption ranges from 4%–60%. Thus, it is clear from these results that the error in the energy consumption is due to the error in time, for which only small problem sizes were used to fit the ECM performance model.

It must also be mentioned here that the LU predictions are less accurate than those for CG or CoMD. Again, this reflects on the model for problem size. LU may be used here as an example of the difficulty in defining problem size in the terms that are meaningful across platforms. CoMD also suffers from an incomplete definition of problem size, which is shown in Fig. 32; however because the code is compute-bounded, the definition for problem size still maintains a mostly linear trend. On Borges, it is clear that the code is less compute- bounded than on Marquez; the linear fit is better on Marquez than Borges.

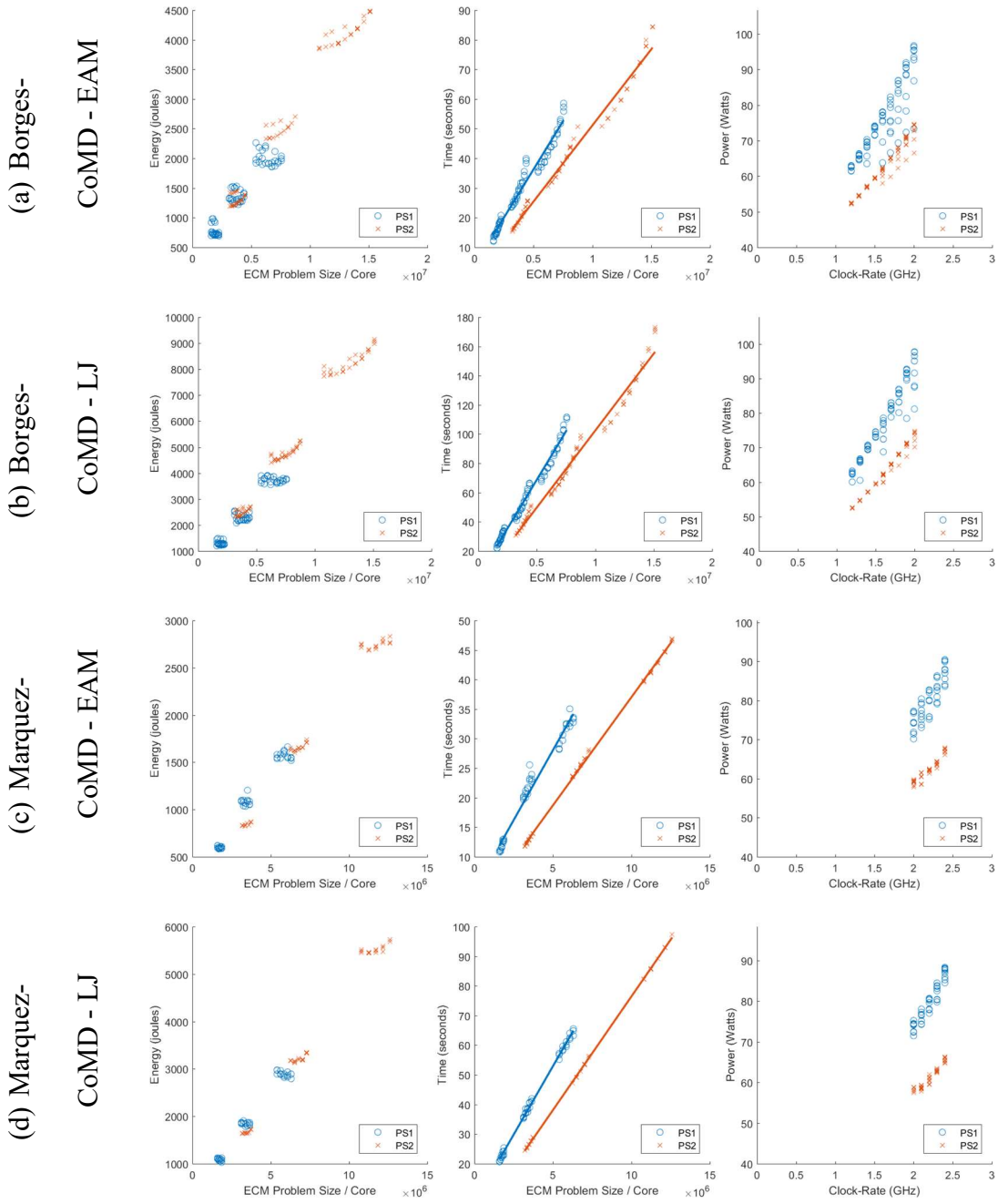


Fig. 30. Predicted vs. measured energy, time, and average power of CoMD on Borges and Marquez.

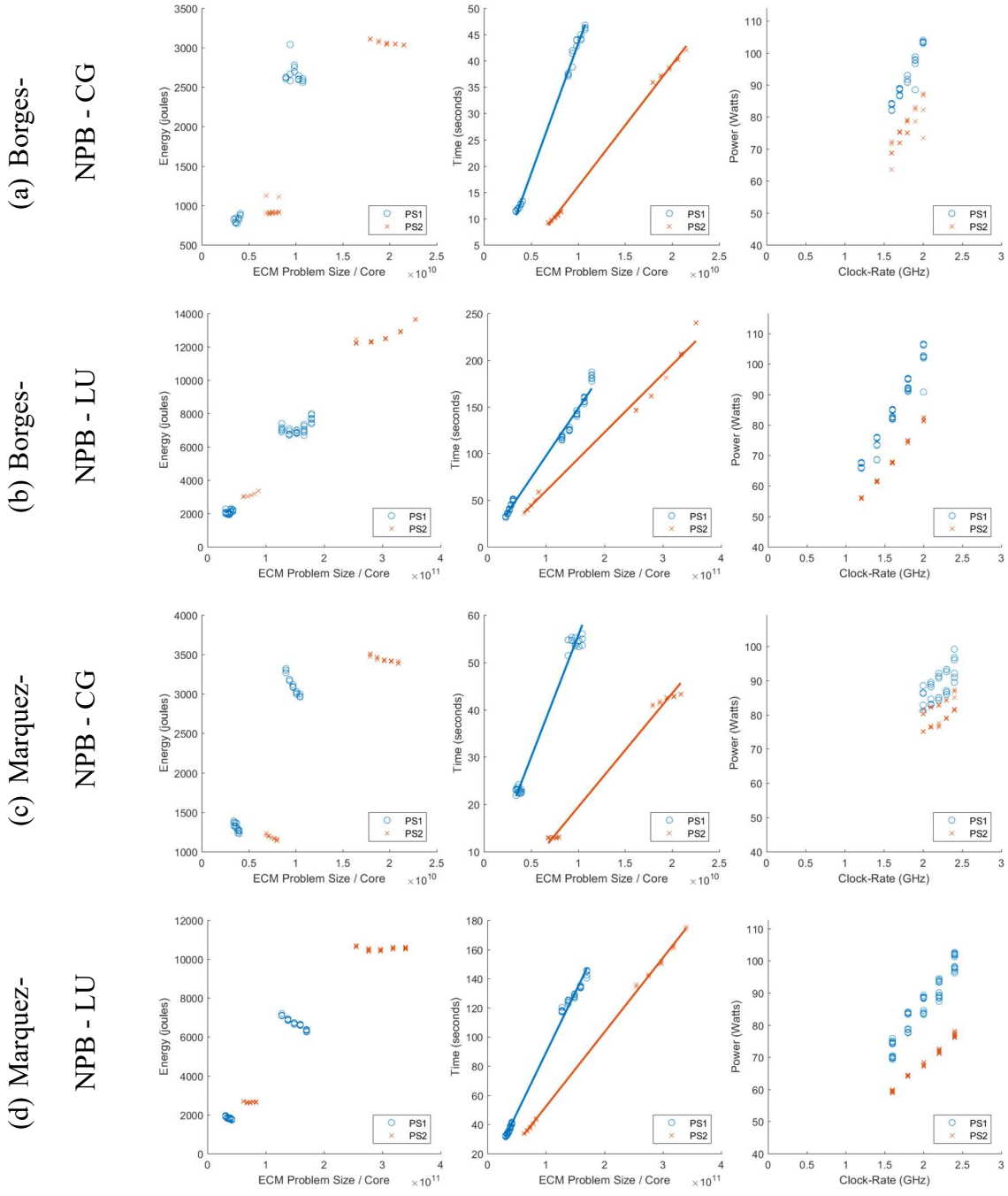


Fig. 31. Predicted vs. measured energy, time, and average power of CG and LU on Borges and Marquez.

TABLE XI

PREDICTED VS. MEASURED ENERGY, TIME, AND AVERAGE POWER FOR 3200 MILLION ATOMS (COMD) AND CLASS D (NPB – CG AND LU) ON BORGES.

	Energy		Time	Avg Power
CoMD-EAM	Measured	214752	2149.64	99.61
	Modeled	262802	2657.39	97.75
	Error	22.4%	23.6%	-1.9%
CoMD-LJ	Measured	135843	1393.69	96.79
	Modeled	128537	1300.20	96.67
	Error	-5.4%	-6.7%	-0.1%
NPB-CG	Measured	152655	1353.94	109.85
	Modeled	129783	1176.01	104.00
	Error	-15.0%	-13.1%	-5.3%
NPB-LU	Measured	261480	2260.84	114.57
	Modeled	259970	2421.47	106.52
	Error	-5.7%	7.1%	-7.0%

TABLE XII

PREDICTED VS. MEASURED ENERGY, TIME, AND AVERAGE POWER FOR 3200 MILLION ATOMS (COMD) AND CLASS D (NPB – CG AND LU) ON MARQUEZ.

	Energy		Time	Avg Power
CoMD-EAM	Measured	153553	1711.43	89.45
	Modeled	166320	1851.41	88.31
	Error	8.3%	8.2%	-1.3%
CoMD-LJ	Measured	95350	1072.08	88.54
	Modeled	88585	950.10	90.49
	Error	-7.1%	-11.4%	2.2%
NPB-CG	Measured	236406	2454.74	94.92
	Modeled	123447	1224.78	99.11
	Error	-47.8%	-50.1%	4.4%
NPB-LU	Measured	553854	5241.16	104.93
	Modeled	222934	2156.78	102.59
	Error	-59.7%	-58.8%	-2.2%

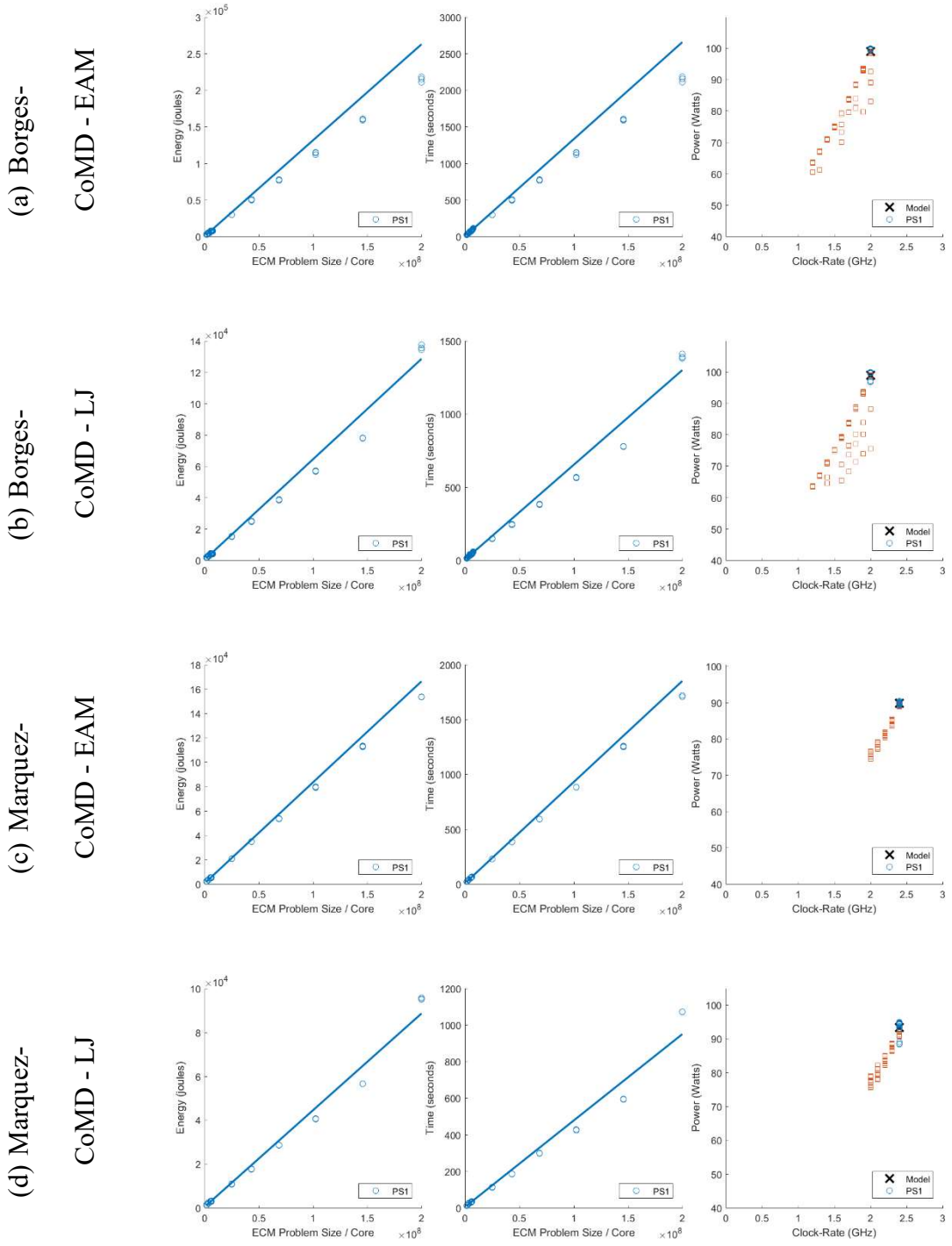


Fig. 32. Predicted vs. measured energy, time, and average power of CoMD on Borges and Marquez.

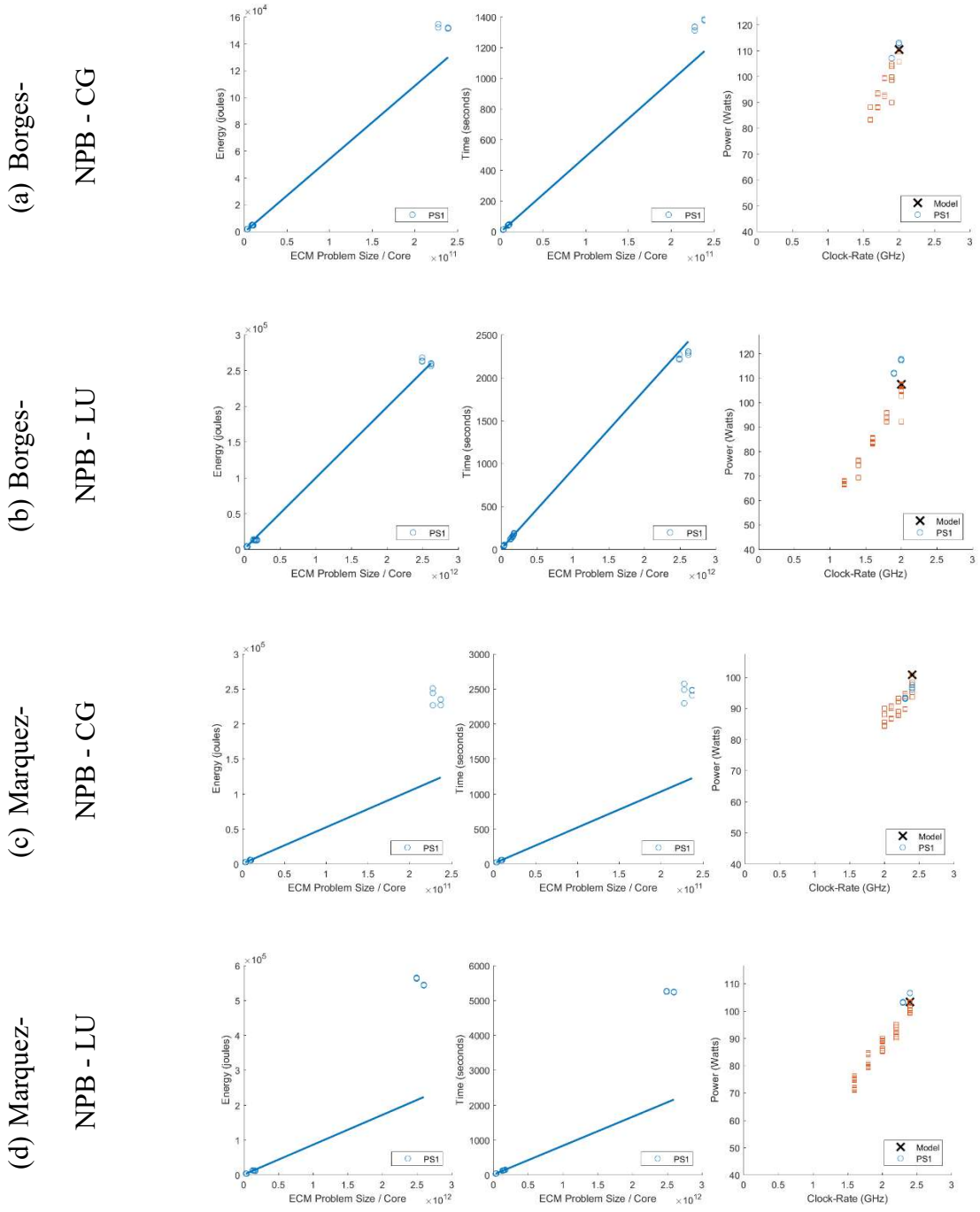


Fig. 33. Predicted vs. measured energy, time, and average power of CG and LU on Borges and Marquez.

6.4 Analysis

Consider the power traces and corresponding probability distributions shown in Figs. 34 and 35 for each application and computing platform. An annotation has been added for probabilities exceeding the specified boundaries of the plot. Each bar of the distribution represents a 5W bin, and each distribution is normalized by the total number of samples in the trace (except idle power).

Idle power measurements are excluded from the trace because the distribution represents the execution. Hence, the power traces have been cropped 4.8s from the start and 5.0s from the end of the trace, i.e., a majority of idle power measurements are ignored. Approximately 0.2s of idle power measurements remain at the beginning of the trace because there is a delay between issuing the command to measure power and the time required to make the first measurement. It may be possible to find a smaller value than 0.2s, however idle power measurements contribute less than 1% of overall distribution.

6.4.1 Distribution Analysis

The power distributions in Figs. 34 and 35 share a common trend, in that they may be modeled using a Normal distribution [56], which is represented in the figure as a solid line. Table XIII presents the Normal distribution parameters (mean μ, standard deviation σ) for the power traces shown in Figs. 34 and 35. The normal distribution is defined as:

$$f(x) = \frac{1}{\sqrt{2\times\pi\times\sigma^2}} \times \exp\left(\frac{-(x-\mu)^2}{2\times\sigma^2}\right), \tag{40}$$

where $f(x)$ is the resulting probability density function, σ is standard deviation, and μ is the mean.

Energy distributions for each application and system are presented in Figs. 36 and 37; like with the power traces, the energy traces show energy over time for the execution and the corresponding distribution. The distributions have been fit to a bimodal distribution [90], and the model parameters are presented in Table XIV. The bimodal distribution is defined as:

$$f(x) = \alpha_1 \times exp\left(\left(\frac{-(x-\mu_1)}{\sigma_1}\right)^2\right) + \alpha_2 \times exp\left(\left(\frac{-(x-\mu_2)}{\sigma_2}\right)^2\right), \quad (41)$$

where $f(x)$ is the resulting probability density function, α_1 and α_2 are approximately equal to $1/\sqrt{2\times\pi\times\sigma^2}$, and μ_1, μ_2, σ_1, and σ_2 are the mean and standard deviation of each peak of the distribution (labeled 1 and 2 respectively).

What is interesting about the distributions here is that power can be described generally using only one peak, but energy shows two (sometimes three, see Fig. 35) peaks, depending on the application. Also make note that the power measurements near idle power (and respective energies) are not representative of the distribution, and consist of fewer than 1% (0.01 normalized) of all samples; indeed these measurements are outliers, which supports the theory that measurements near idle are indicative of delays captured via power draw, see Fig. 35d. Generally power and energy for an application-platform are shown as a whole, meaning energy is the sum of the power samples over time, providing a single value for energy consumption.

Similarly, power is averaged such that a single value is representative of the trace. This is good, but it doesn't say much about how the application used that energy. The energy distribution shows this. It provides a range of energy values, and shows that given a

sampling rate (about 10ms measured), energy most commonly is used in specific increments of energy. In this way, applications and hardware can be more directly compared. And since energy is about equal to cost, this is also good for cost assessment.

6.4.2 EMD Residual and the QFR Model

Consider the QFR power models and EMD residuals presented in Figs. 38 and 39. EMD has been applied to each power trace in Figs. 34 and 35 and the corresponding residual (`Measured EMD`) is compared against the predicted quadratic fit residual (`Modeled QFR`), calculated as described in Eq. (42).

$$P(t) = at^2 + bt + c \tag{42}$$

$$a = \frac{-b}{T} \tag{43}$$

$$b = \frac{4 \times P_d}{T} \tag{44}$$

The coefficients a, b, and c are calculated using Eqs. (43) and (44) where coefficient c is idle power draw, dynamic power draw P_d is the difference between average power and idle, and time T is the time predicted by the ECM performance model. Idle power draw has been measured at 45.8 W for Borges and 33.3 W for Marquez (at the maximum clock-rate), which can be seen in the QFR's in Figs. 38 and 39.

Compare the EMD residual to the QFR. The error calculated prior (see Tables XI and XII) is captured in the relative length of each curve. Both the QFR and EMD residual also approach average power, which is expected since power was not modeled here. Focusing on the relative shape of the EMD residual vs. the QFR, notice that the EMD residual converges on average power draw as time increases.

For a shorter trace, the start and end of the EMD residual may be observed approaching idle power; this is explained by the amount of idle power samples in the trace. Since a fixed amount of idle measurements are included in the trace, independent of execution time, the influence of idle power on the residual of shorter traces is increased. The opposite occurs for larger traces (i.e. the curve is mostly influenced by dynamic power draw), hence the EMD residual converges on average power draw. Figure 34d provides an excellent example of a short trace, and Fig. 35c shows a long trace (> 1000 seconds).

Further, the EMD residual is more representative of the power trace than the QFR since EMD is based on the measurements. The minimum power exhibited by the residual is much larger than that of the QFR. This suggests that the observed static power draw for the execution is much higher than typically reported, since EMD is including the power required to keep cores active throughout the execution. Typically, the power required to keep cores active is tallied as dynamic power draw since this may be scaled with DVFS; however, if the power draw during execution does not dynamically change, this may be a misrepresentation of power. Based on the trend from EMD, a substantial amount of power may be concluded as static power draw.

A similar finding has been found by the authors in [28] who use linear regression to evaluate power models meant to investigate uncore power usage. They found that up to 74% of static power is due to uncore power draw, and up to 61% of total energy consumption on the Haswell CPU is due to uncore power draw. This finding is consistent with the findings for the Xeon Phi which has a static power draw of approximately 80W and most applications use up to 160W when loaded — in this case, static power contributes to 50% total energy consumption. Indeed, static power draw is the leading

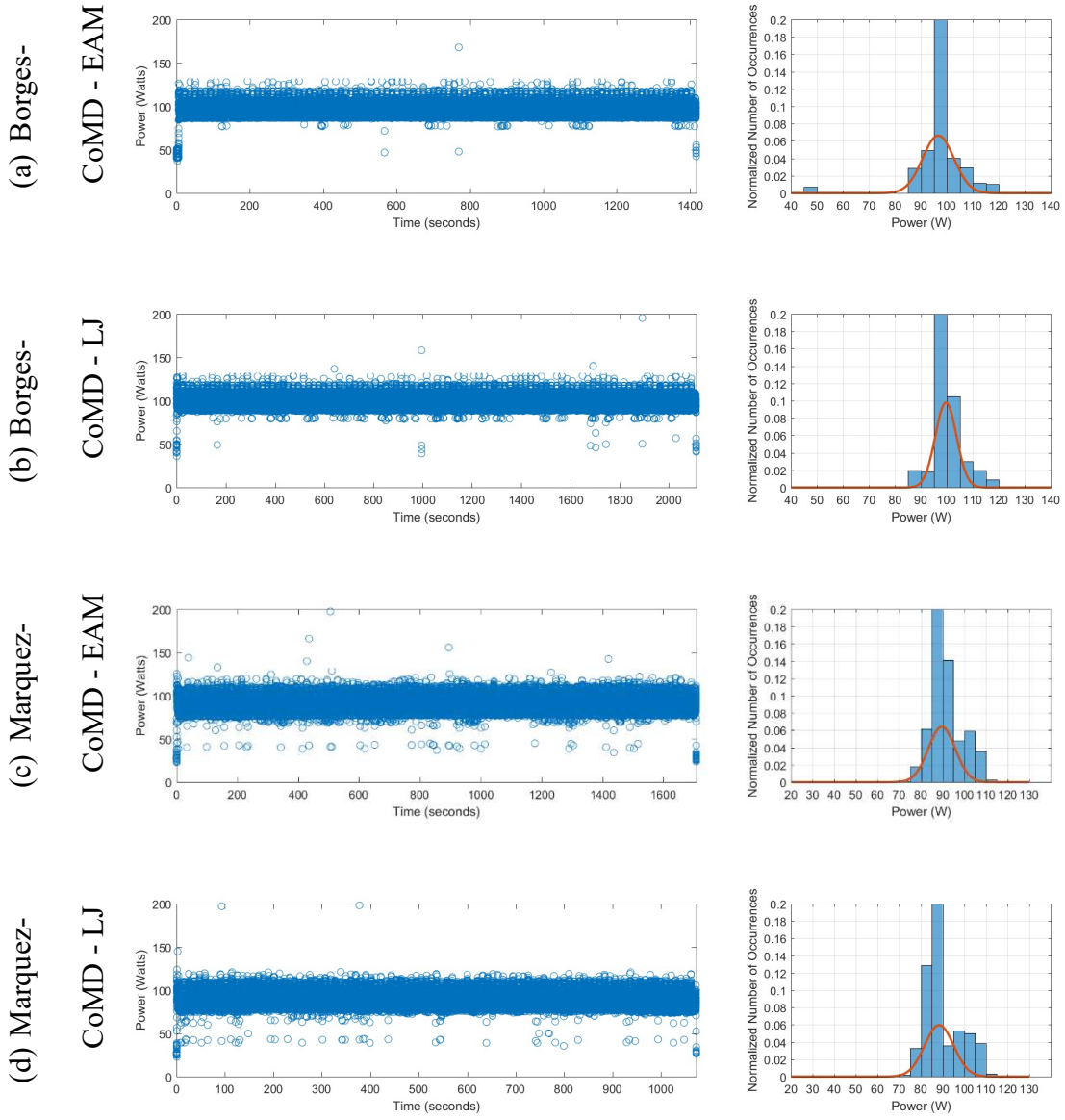


Fig. 34. Power traces for CoMD on Borges and Marquez at max problem size (3.2 billion atoms) with distribution of power samples normalized by the total number of samples in the trace.

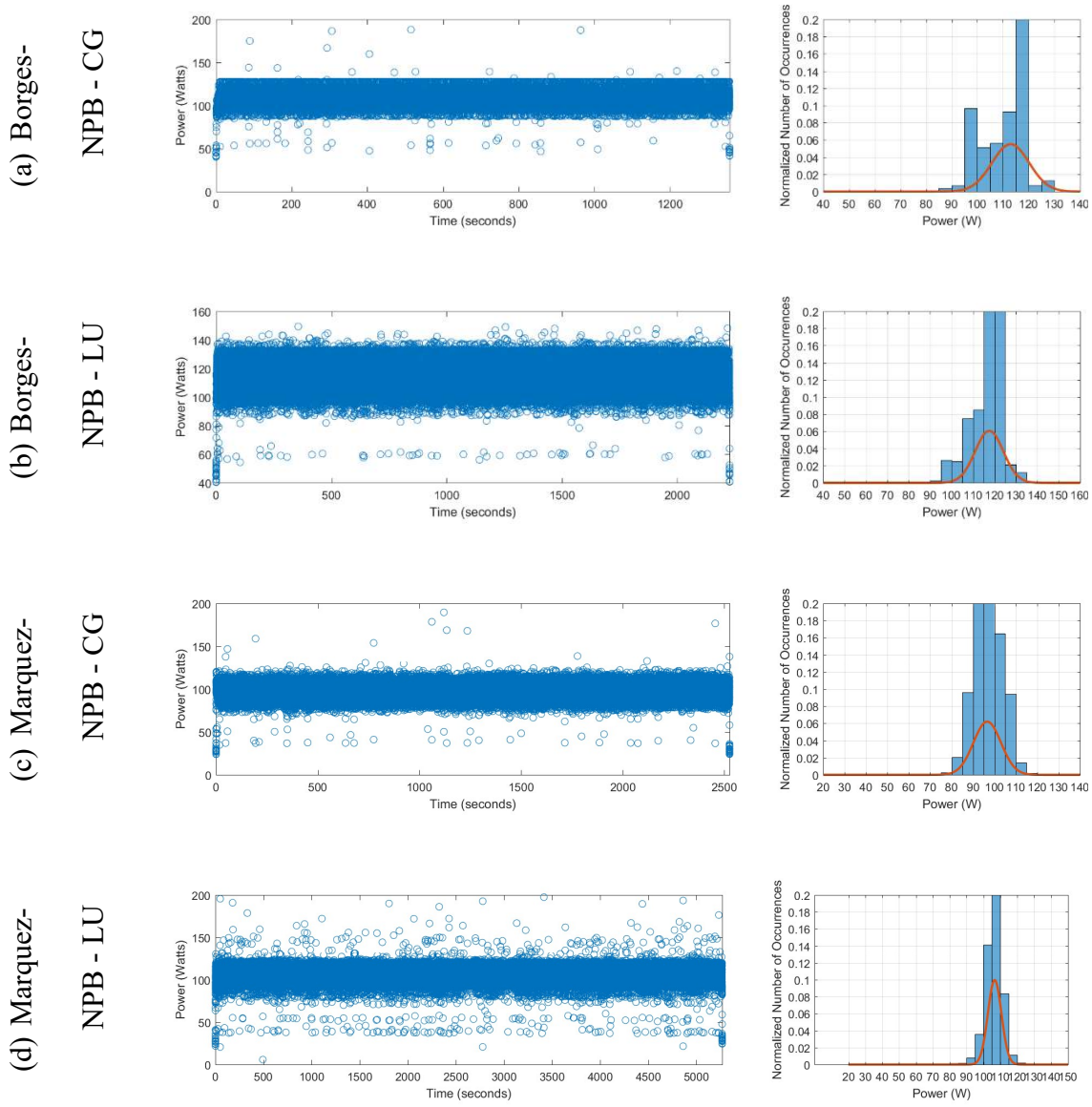


Fig. 35. Power traces for CG and LU on Borges and Marquez at max problem size (class D) with distribution of power samples normalized by the total number of samples in the trace.

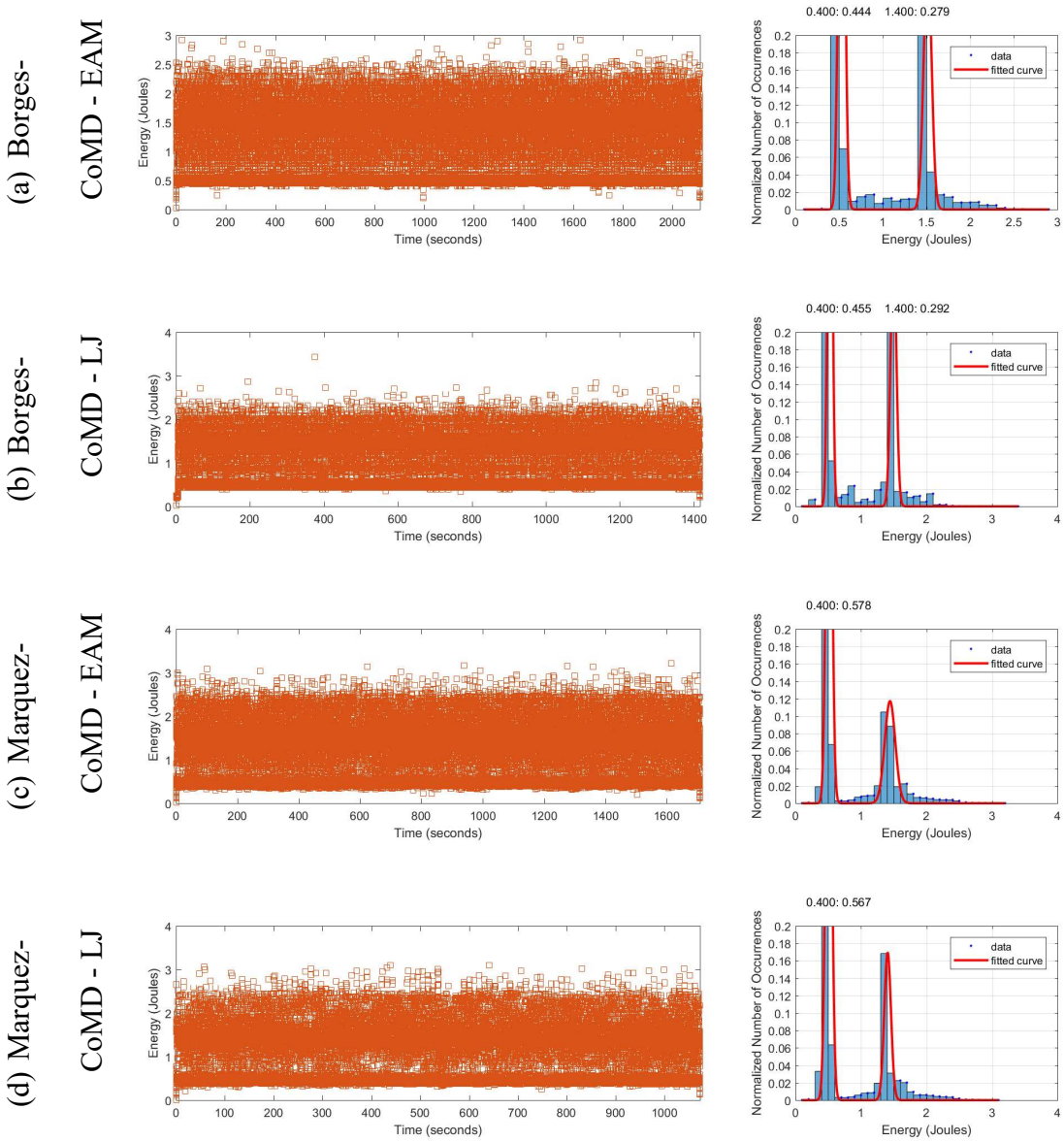


Fig. 36. Energy traces for CoMD on Borges and Marquez at max problem size (3.2 billion atoms) with distribution of power samples normalized by the total number of samples in the trace.

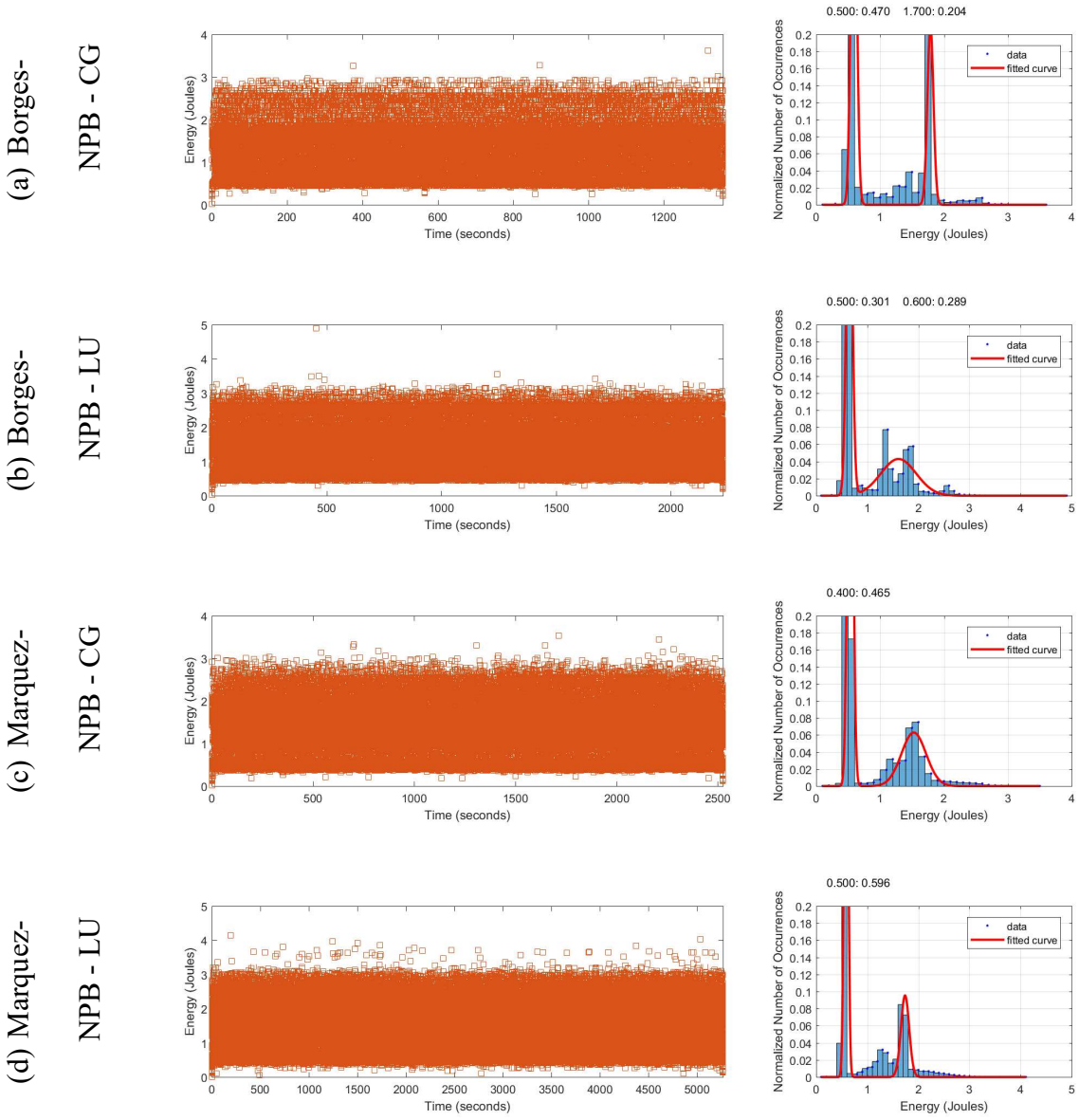


Fig. 37. Energy traces for CG and LU on Borges and Marquez at max problem size (class D) with distribution of power samples normalized by the total number of samples in the trace.

TABLE XIII

NORMAL DISTRIBUTION COEFFICIENTS OF POWER FOR COMD AND NPB (CG AND LU) ON THE BORGES AND MARQUEZ PLATFORMS.

System	Application	Mean μ	Std Dev σ
Borges	CoMD - EAM	99.6	4.069
	CoMD - LJ	96.7	6.017
	NPB - CG	113.0	7.242
	NPB - LU	117.6	6.579
Marquez	CoMD - EAM	89.6	6.223
	CoMD - LJ	88.4	6.696
	NPB - CG	6.429	96.6
	NPB - LU	106.6	3.996

TABLE XIV

BIMODAL DISTRIBUTION COEFFICIENTS OF ENERGY FOR COMD AND NPB (CG AND LU) ON THE BORGES AND MARQUEZ PLATFORMS.

System	Application	α_1	μ_1	σ_1	α_2	μ_2	σ_2
Borges	CoMD - EAM	0.610	0.528	0.049	0.290	1.513	0.063
	CoMD - LJ	0.601	0.525	0.048	0.294	1.495	0.062
	NPB - CG	0.485	0.589	0.063	0.211	1.788	0.067
	NPB - LU	0.067	0.649	0.079	0.043	1.610	0.488
Marquez	CoMD - EAM	0.600	0.511	0.060	0.117	1.442	0.114
	CoMD - LJ	0.573	0.507	0.063	0.169	1.406	0.072
	NPB - CG	0.646	0.533	0.058	0.063	1.527	0.271
	NPB - LU	0.647	0.585	0.051	0.095	1.738	0.114

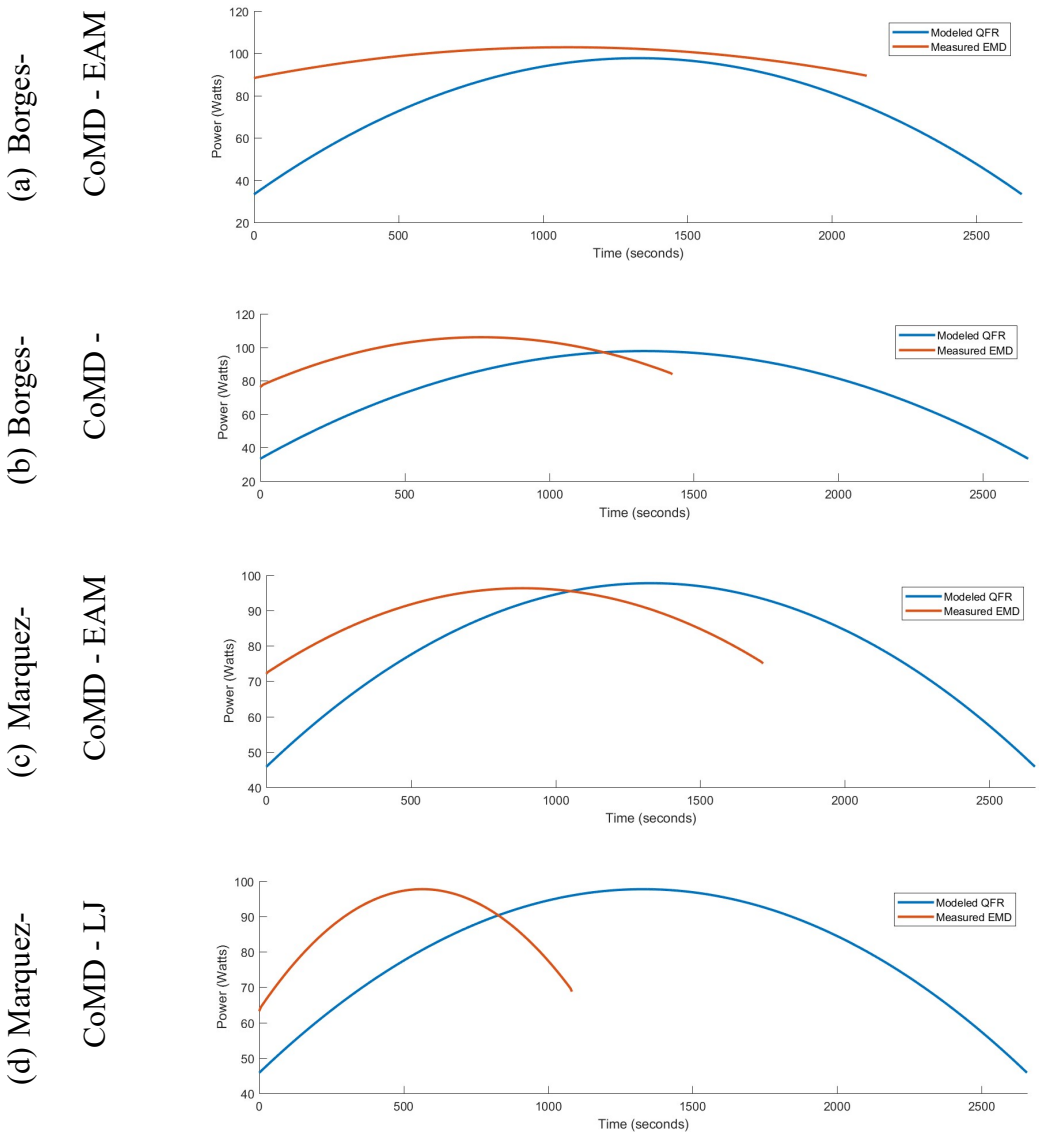


Fig. 38. EMD residual vs predicted QFR for CoMD on Borges and Marquez at max problem size (3.2 billion atoms).

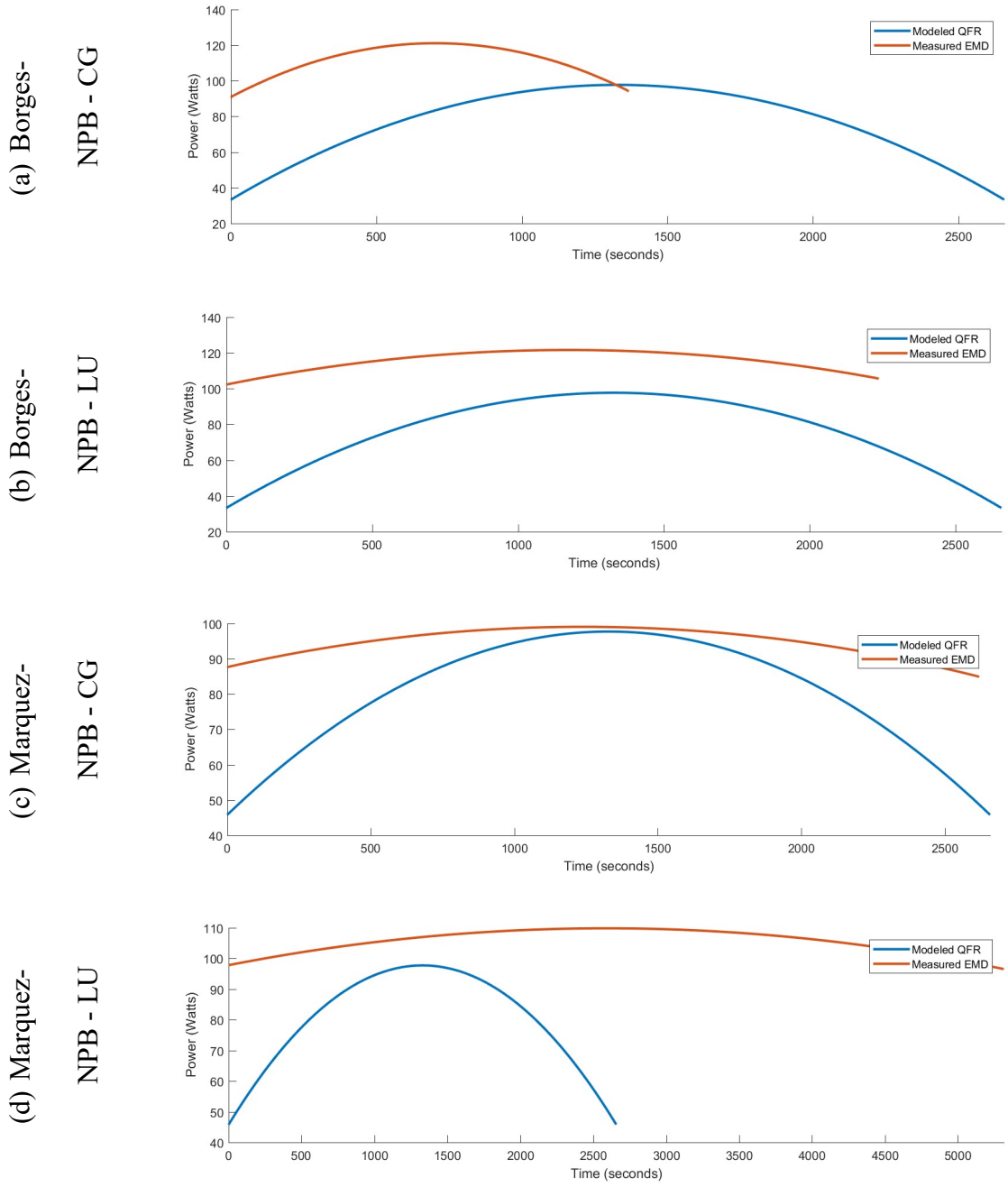


Fig. 39. EMD residual vs predicted QFR for CG and LU on Borges and Marquez at max problem size (class D).

limitation to energy-efficient computing.

Although the QFR model is based on the residual, certain assumptions had been made such that the model may be equated to time and power [60]. The first assumption is power draw is equal at the start and end of execution, thus time may then be easily defined when the y-intercept equal to coefficient c for a time > zero. The second assumption is coefficient c is idle power draw (a.k.a static power). As has been shown here, these assumptions placed limitations on the model, and for long traces (> 1000 seconds), the model began to fail. This is visible in Figs. 38 and 39 where the QFR does not cover nearly the amount of area as the EMD residual. The model could be more representative of the EMD residual if idle power measurements were collected for a time proportional to the execution time; however, this is wasteful and the distributions shown prior are a better representation of power and energy.

6.5 Relative Error Between Prediction Methods

Consider Fig. 40 which compares the relative error for each method to predict energy considered in this chapter; error in predicted energy is relative to measured energy. To recap, measurements were collected using small problem sizes to which the ECM performance model was fit. Using this definition of time, and the maximum value of average power measured, energy was predicted. Thus, the first method is `Avg Power`. Following came the introduction and discussion of the `Power Distribution` and `Energy Distribution` using unimodal and bimodal normal distributions. Finally, the `EMD Residual` and `QFR Model` were discussed.

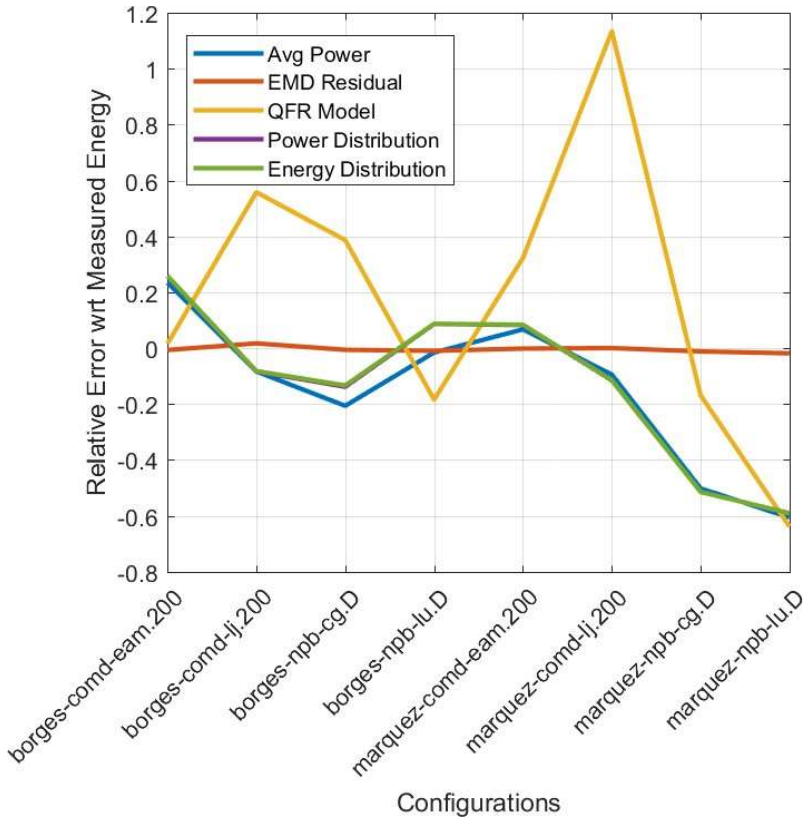


Fig. 40. Error in energy consumption for predicting energy using: average power, EMD residual, QFR model, power distribution, and energy distribution.

First, note in Fig. 40 that the baseline error is `Avg Power`, which was defined prior in Tables XI and XII. Because of the error in the execution time model, no other model here should predict energy better than average power, hence it is used as a baseline.

Also note that the EMD residual remains an excellent representation of energy consumption, although it is based on the measured results and thus is expected to have zero error. The QFR model shows high error, as expected based on the trends in Figs. 38 and 39. The purpose of the QFR model was to be a high-level interpretation of an execution such that applications and hardware could be compared in a way that was meaningful.

Inadvertently, analyzing the traces as distributions provides a better method for representing power and energy. The error shown in Fig. 40 for both the power and energy distributions closely matches the average power. This is to be expected, however, since the distributions incorporate mean power/energy into the model, so the results here could be misleading and further analysis of the method is required.

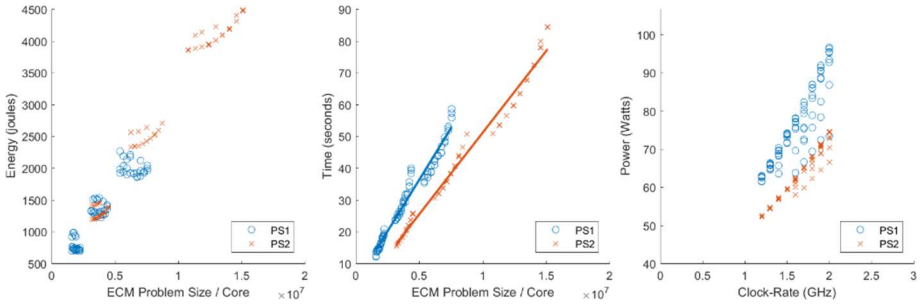
CHAPTER 7

MULTISOCKET AND MULTINODE ANALYSIS

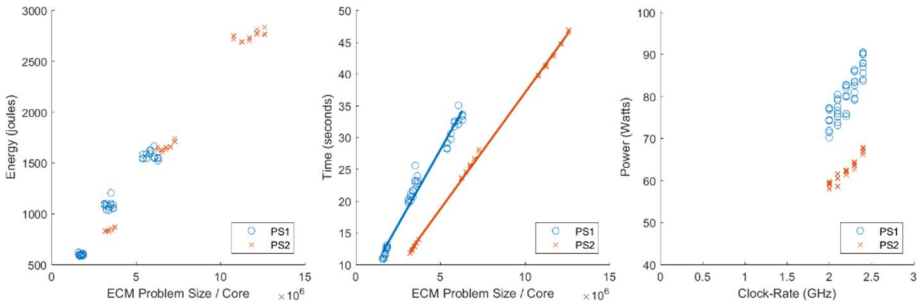
In this chapter, an investigation of hardware platforms featuring multiple sockets per node and multiple nodes is performed. This analysis is meant to bridge the gap between the analyses conducted thus far (primarily focused on single-node executions) and the applicability of the results to large-scale systems. More emphasis has been placed on individual nodes than large groups because these individual nodes make up the whole, and the variability of runs on a particular machine will impact the scaling behavior as additional nodes are added.

In the previous chapter, plots were presented that showed the raw measurements of time, power, and energy for various applications and platforms (in particular, the small problem sizes). The plots of interest have been duplicated here, see Fig. 41. Focus attention to power for each plot; notice that although a quadratic trend may be observed as clock-rate increases, outliers may be observed that draw 10W less than the trend (on Borges, and around 5-6W on Marquez). This has a bigger impact on the `PS1` measurements where 16 cores are used (as opposed to 8 in the `PS2` case). These variations may also be observed in energy (notice the `PS1` measurements are grouped together instead of following a trend as with `PS2`). Even with the trend, outliers are still present in energy (e.g. Borges). This is caused by latency in the system that can only be encountered at runtime. Thus, this behavior is to be expected in the multi-node environment and to greater extremes due to the added level of parallelism (network).

In the multi-node environment, latency due to data movement over the network and load imbalance between nodes will compound with the run-to-run variability in performance



(a) Borges - CoMD - LJ



(b) Marquez - CoMD - LJ

Fig. 41. Duplicate of the measured energy, time, and average power of CoMD on Borges and Marquez for several problem sizes as presented in the previous chapter.

and energy. Optimization of the application can reduce these bottlenecks, but no application can be completely rid of them. Thus, it becomes important to investigate how applications utilize multiple nodes. However, before jumping into multiple nodes, one feature of each node has been overlooked; that is each node is often comprised of multiple sockets and each socket, if measured using RAPL as is done in this work, provide multiple sources of measurement, i.e., total package, core, and DRAM.

7.1 Multisocket Analysis

Many computing platforms designed today feature multiple sockets to increase the available resources per node. This is an important caveat, since one may assume that all cores may share the same cache, however often there are multiple processors and the node is represented as the total amount of resources (e.g. cores, memory, cache levels, bandwidth, power draw, etc.).

Intel processors allow for energy to be measured using the running average power limit (RAPL) interface [20] and more recent processors utilize the Linux Power Capping Framework [2] to obtain measurements. Most important here is that each socket reports energy independently, and offers energy measured for the core, DRAM memory, and sometimes the package (to indirectly include uncore).

In the author's previous works [58–60] EMD has been applied to a *power* trace, consisting of total power (the sum of all power sources). In this section, an investigation of applying EMD to the various sources of energy is conducted; specifically, EMD is applied to power traces for the following measurement sources: total and per source totals for core, DRAM, and uncore.

7.1.1 Power Traces

The power traces presented in Figs. 42 and 43 show two problem sizes for the GAMESS chemistry application, namely *1L2Y* and *20w*, while varying the MPI thread affinity between *bunch* and *scatter*. The traces shown here have been collected on the Haswell system, *Marquez*, because Haswell and newer processors include DVFS for DRAM and the impact of thread affinity is more obvious.

GAMESS is different from the other applications shown in this dissertation (CoMD, and any NPB benchmark) because it has been designed to be *task-parallel* as opposed to the other applications which are *data-parallel*. Data parallel is very easy to implement; generally, one or more loops index one or more datasets and computations are performed on them. Task parallel is less specific, in that a "task" could represent any number of operations, such as computation (or a series of computations), writing to the hard disk (I/O), and communication.

In GAMESS, an equal number of MPI processes are created to accomplish two types of tasks; one set for computations and communication, and the other set for I/O. The advantage of this application design is that data I/O can overlap computation, although the code is more complex to accommodate this optimization. And since half of the tasks perform very little work, again data I/O, over-subscribing cores is commonly used for this application. In the traces presented in Figs. 42 and 43, half of the total cores are dedicated to computation and the other half to data I/O.

Notice in Figs. 42 and 43 that the DRAM power is flat on the second Socket for the Bunch affinity, and activity is observed for both Sockets for the Scatter affinity. Next, notice that core power is fairly constant throughout the execution on both sockets, no matter the affinity. This is interesting, since power traces on older generations of Xeon CPU's show a lot of variability in the CPU power and constant power draw for DRAM.

Consider Fig. 44 which shows GAMESS-1L2Y under the bunch and scatter affinities on Sandy-Bridge. What is most interesting for this architecture is that there is not a difference in power draw between sockets while varying affinity. Since uncore and DRAM power do not change throughout execution, all variability in the trace occurs on the core

power rail. It is unexpected that the bunch affinity would cause power draw to vary on both sockets. A similar study in [93] showed that the DVFS granularity is the cause for the power draw on both sockets. Note, this is also observed for configurations with fewer cores (8 used in the execution in Fig. 44).

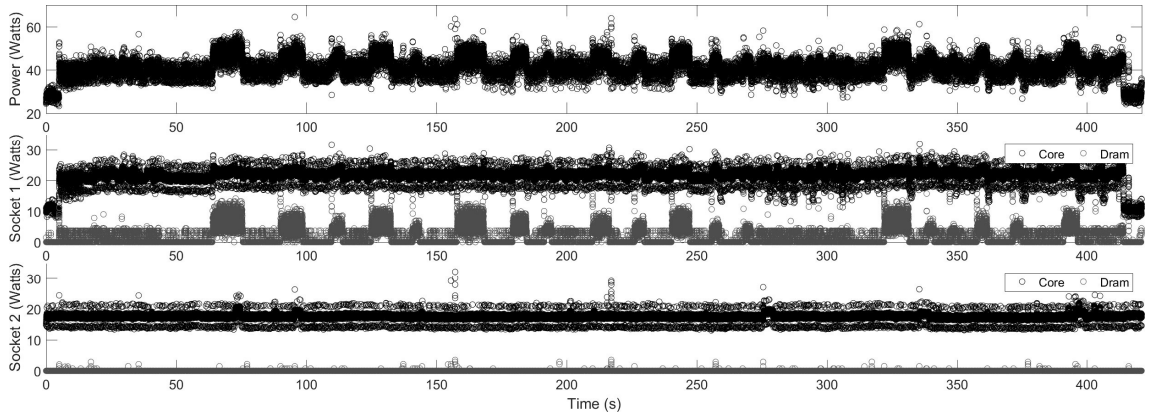
7.1.2 EMD on Socket Traces

When EMD is applied to power traces, it provides a wealth of information yet to be completely explored. A question may be raised; *what are the differences when EMD is applied to the total power versus the individual traces?* To answer this question, the IMF's for the total power of all nodes, of each node, and of each source are considered. The method has been extended to per-source and per-socket analysis, but that is not shown here for the sake of exposition.

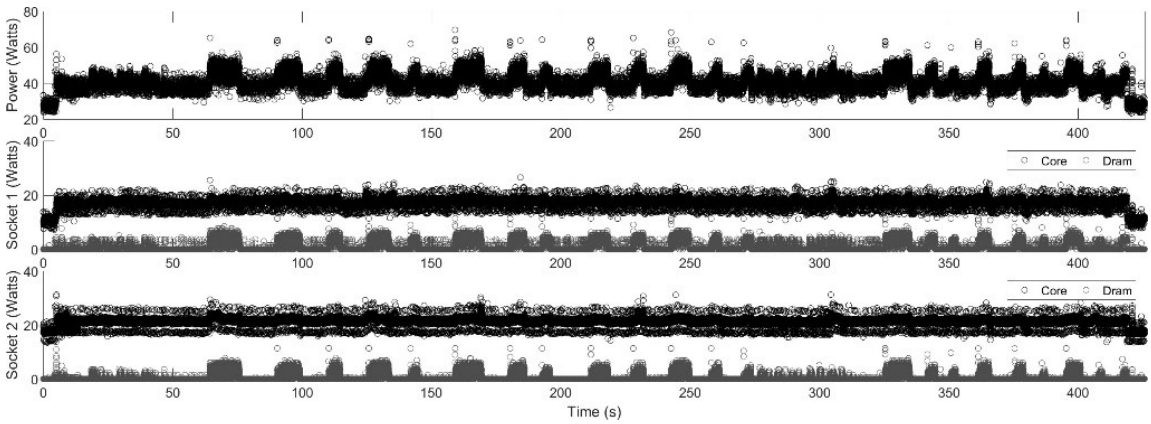
7.1.2.1 Per Source Traces

Consider the IMF's generated for the traces in Fig. 42 which are presented in Figs. 45 to 48. Here, the total power trace is compared against the core and DRAM power traces (sum of both sockets). And for the sake of exposition, only the 1L2Y problem size is discussed since there is more variation in power as a function of execution time.

To maximize the visibility of the data, IMF's have been separated into two groups for each trace; Figs. 45 and 46 present the IMF's for the bunch affinity and Figs. 47 and 48 for the scatter affinity. Total power is shown in black, core power is shown in light gray, and DRAM power is shown as dark gray; for reference, the original power traces are presented in Figs. 45 and 47, respectively.

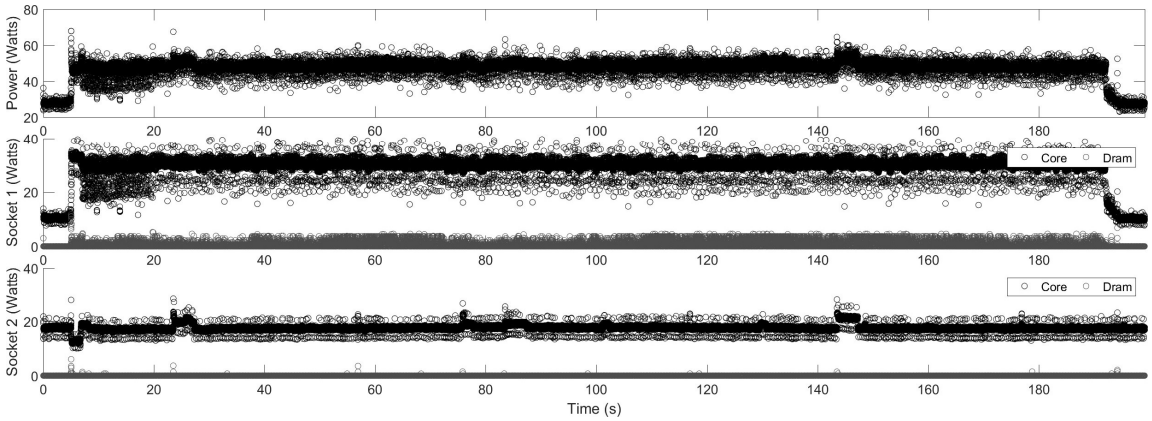


(a) Marquez - GAMESS - 1L2Y - 4 cores –Bunch

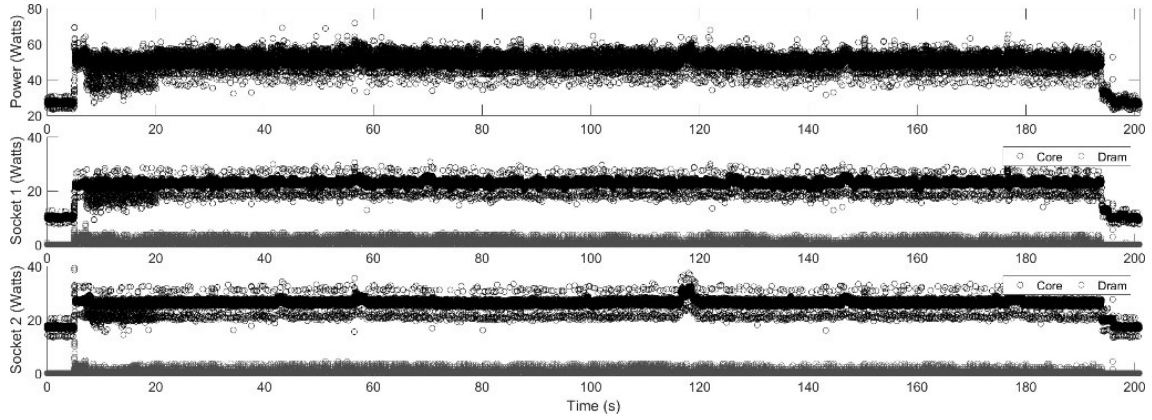


(b) Marquez - GAMESS - 1L2Y - 4 cores -Scatter

Fig. 42. Power traces of GAMESS-1L2Y collected on Haswell showing total vs. per socket, per source traces while varying MPI thread affinity. For each plot (a & b), there are three subplots: top shows total power, middle and bottom shows socket 1 & 2 power respectively where core power is shown in black, and DRAM power in gray.

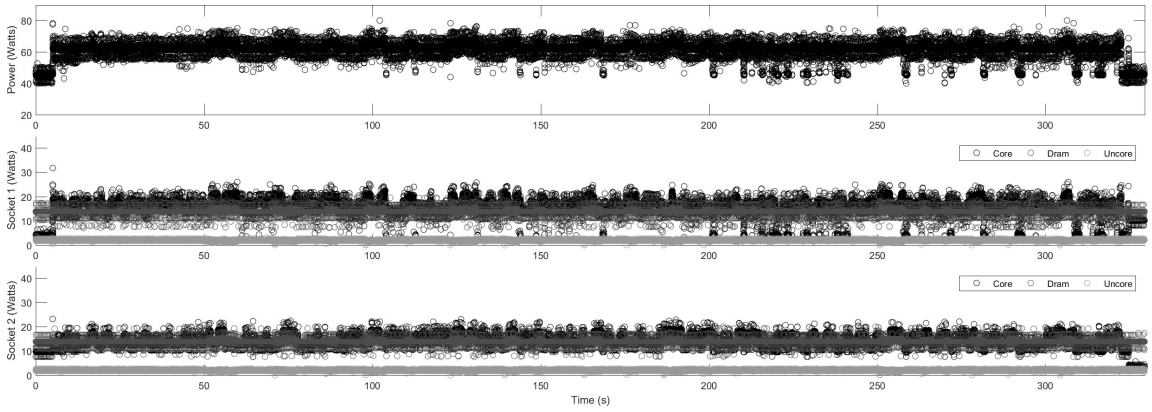


(a) Marquez - GAMESS - 20w - 8 cores –Bunch

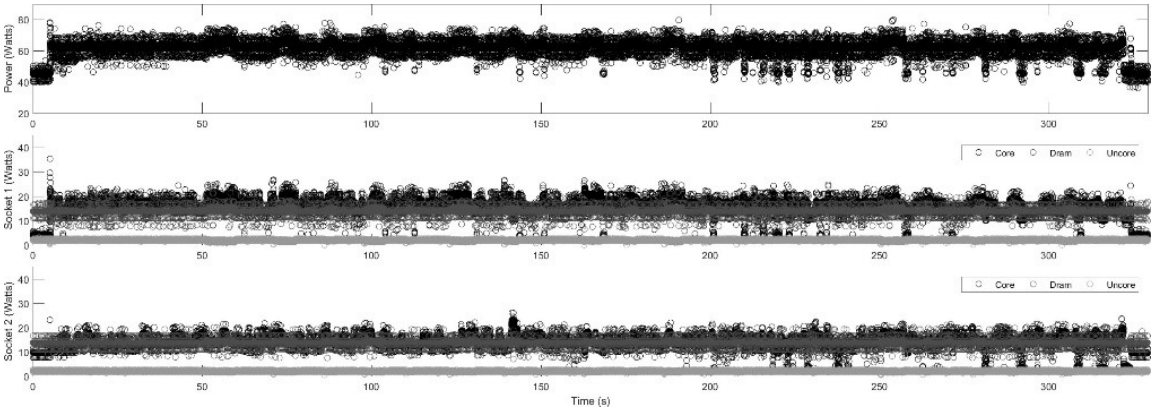


(b) Marquez - GAMESS - 20w - 8 cores -Scatter

Fig. 43. Power traces of GAMESS-20w collected on Haswell showing total vs. per socket, per source traces while varying MPI thread affinity. For each plot (a & b), there are three subplots: top shows total power, middle and bottom shows socket 1 & 2 power respectively where core power is shown in black, and DRAM power in gray.



(a) Borges - GAMESS - 1L2Y - 8 cores – Bunch



(b) Borges - GAMESS - 1L2Y - 8 cores - Scatter

Fig. 44. Power traces of GAMESS-1L2Y collected on Sandy-Bridge showing total vs. per socket, per source traces while varying MPI thread affinity. For each plot (a & b), there are three subplots: top shows total power, middle and bottom shows socket 1 & 2 power respectively where core power is shown in black, DRAM power in dark gray, and uncore in light gray.

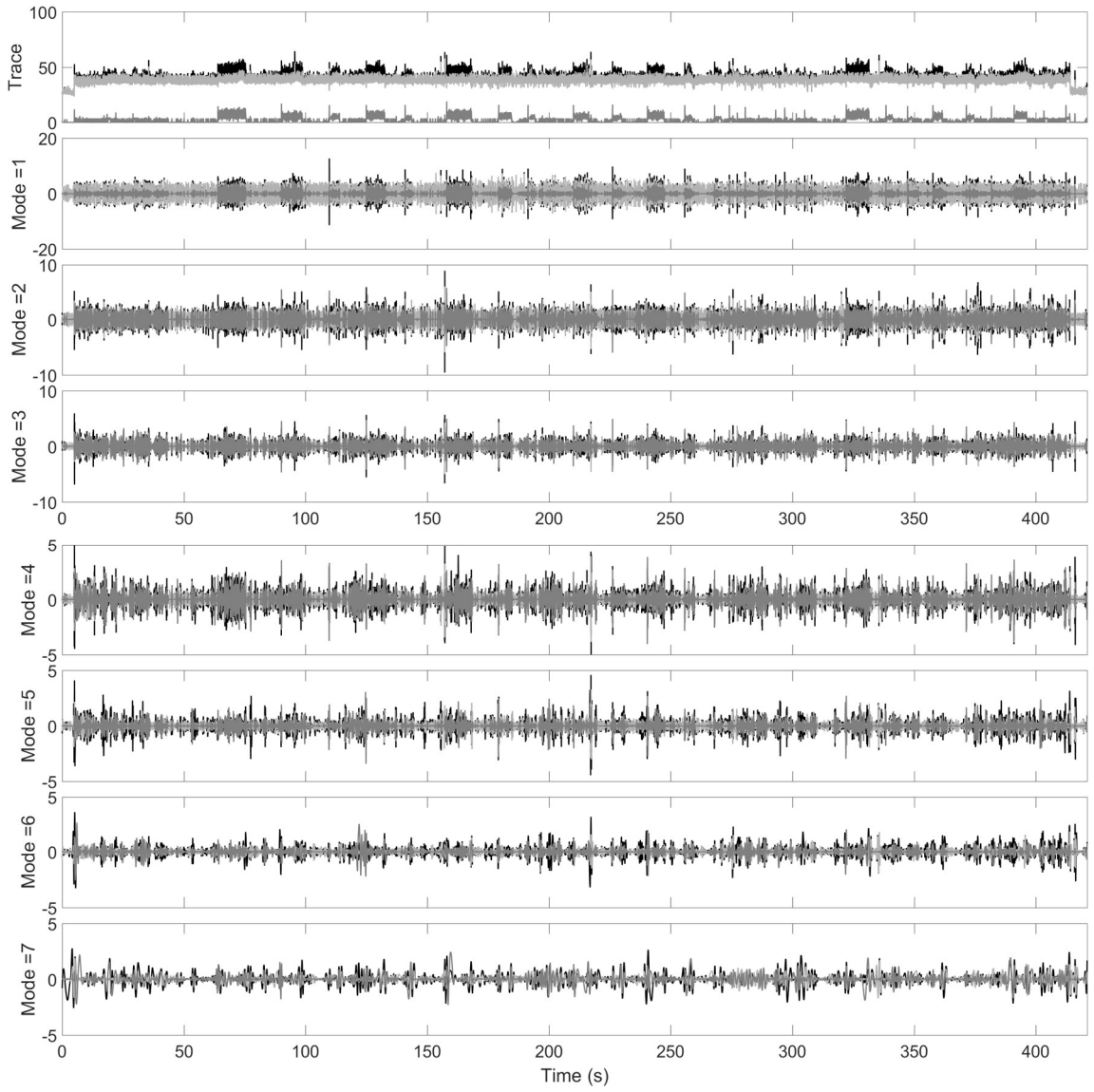


Fig. 45. Power trace and the first half of the IMF's for GAMESS-1L2Y collected on Haswell with the bunch affinity. In each plot, total power (black) is shown with core (light gray) and DRAM (gray) power traces (the total of both sockets).

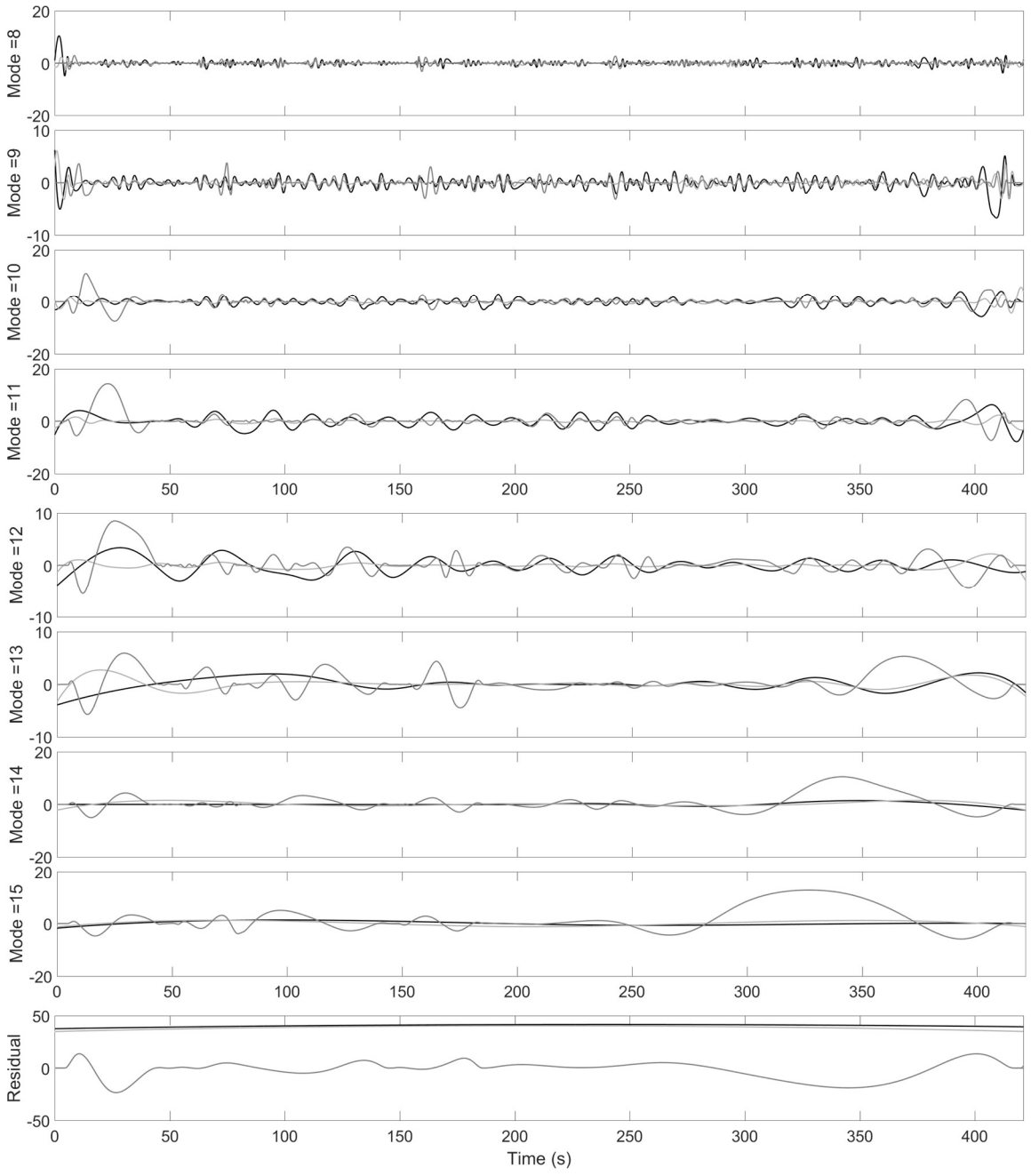


Fig. 46. Residual and the second half of the IMF's for GAMESS-1L2Y collected on Haswell with the bunch affinity. In each plot, total power (black) is shown with core (light gray) and DRAM (gray) power traces (the total of both sockets).

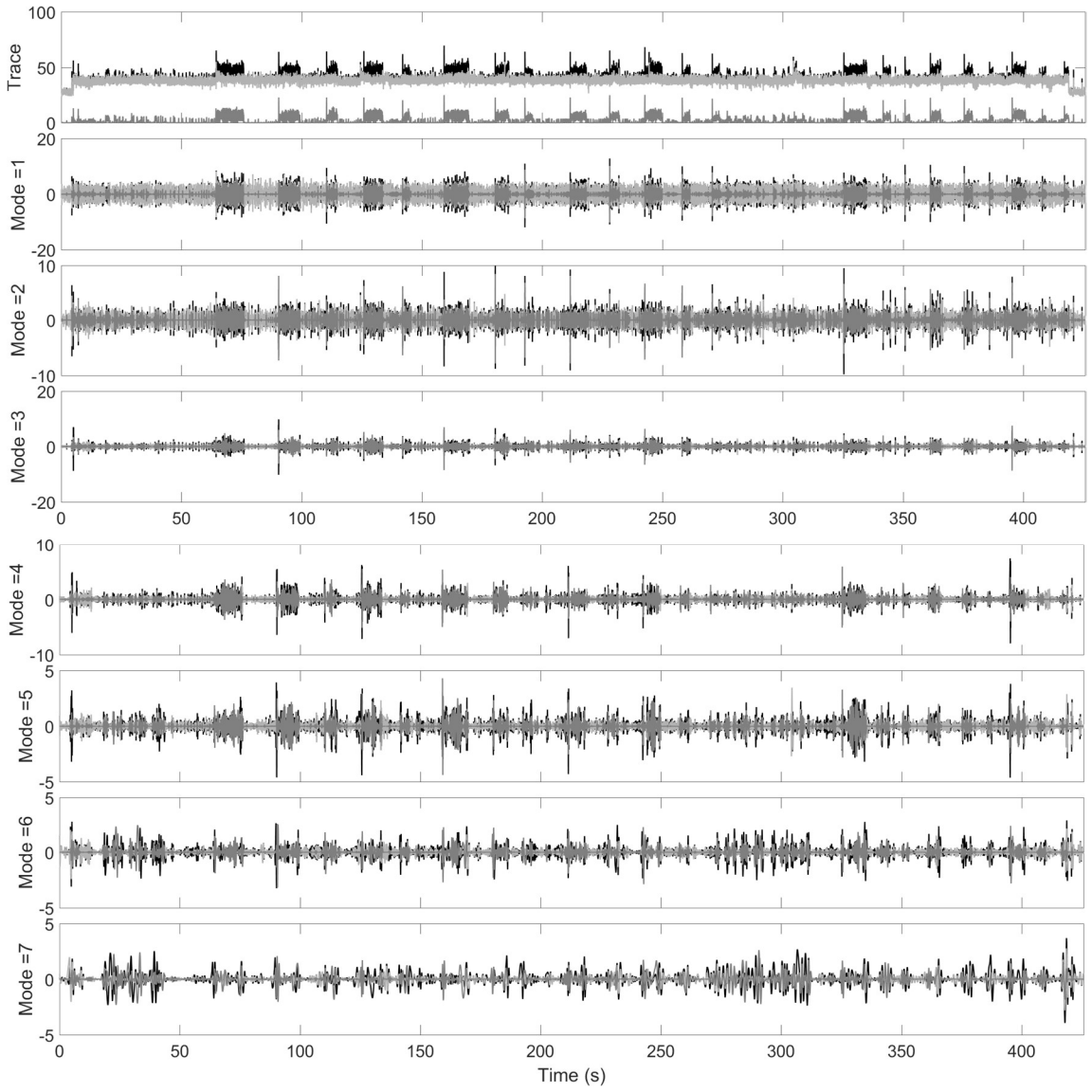


Fig. 47. Power trace and the first half of the IMF's for GAMESS-1L2Y collected on Haswell with the scatter affinity. In each plot, total power (black) is shown with core (light gray) and DRAM (gray) power traces (the total of both sockets).

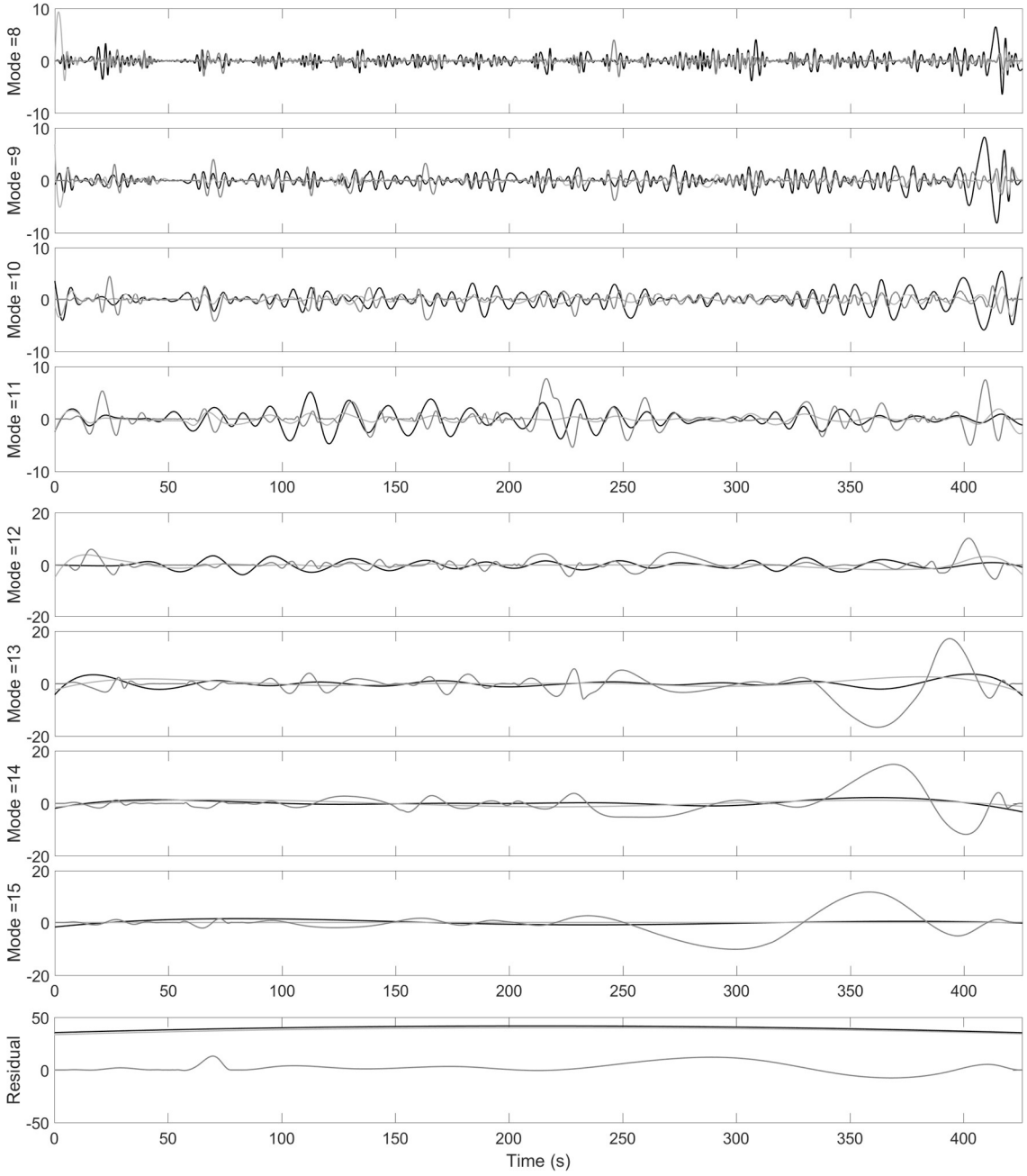


Fig. 48. Residual and the second half of the IMF's for GAMESS-1L2Y collected on Haswell with the scatter affinity. In each plot, total power (black) is shown with core (light gray) and DRAM (gray) power traces (the total of both sockets).

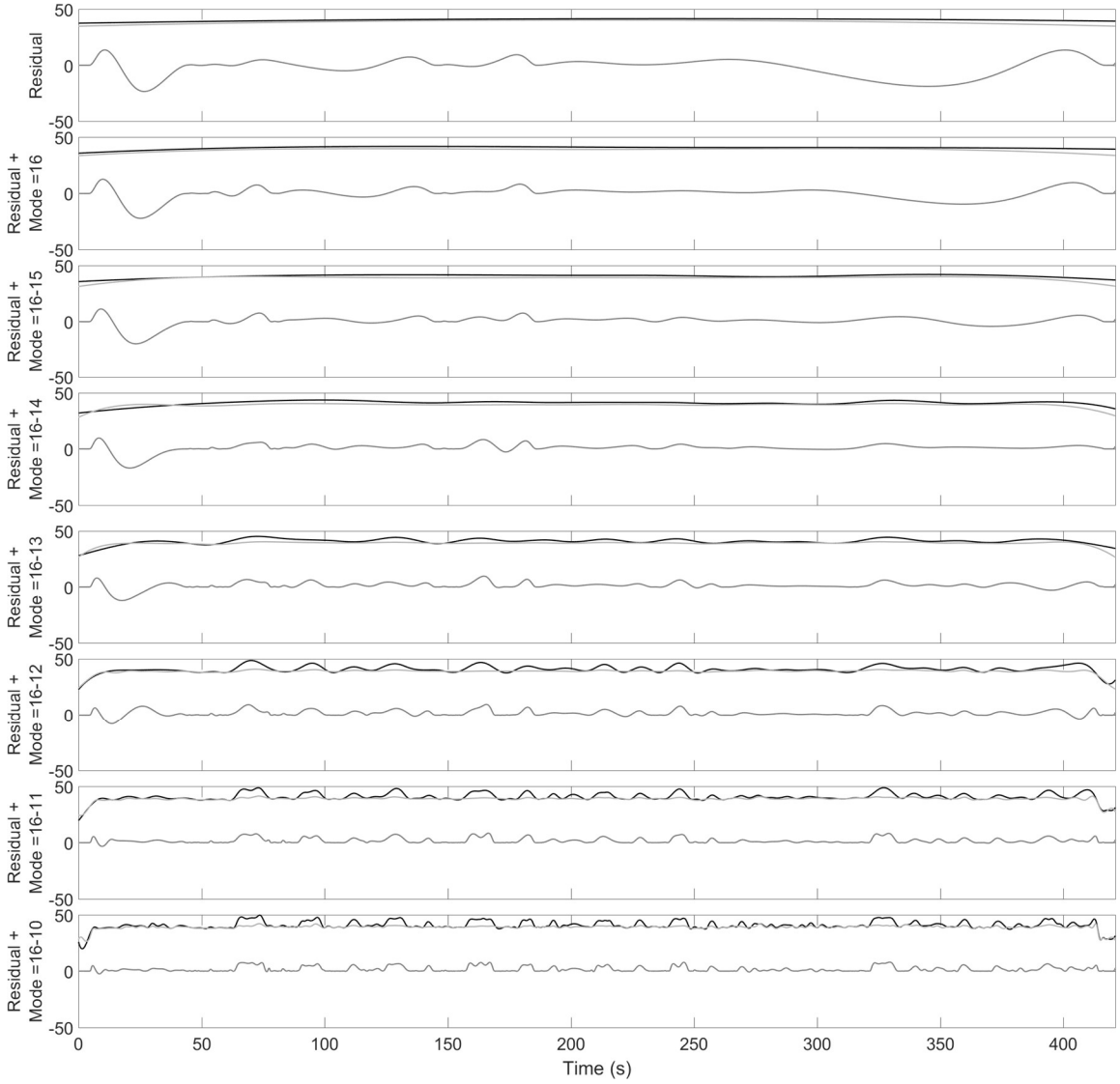


Fig. 49. IMF reconstruction (first half) for GAMESS-1L2Y collected on Haswell with the bunch affinity. In each plot, total power (black) is shown with core (light gray) and DRAM (gray) power traces (the total of both sockets).

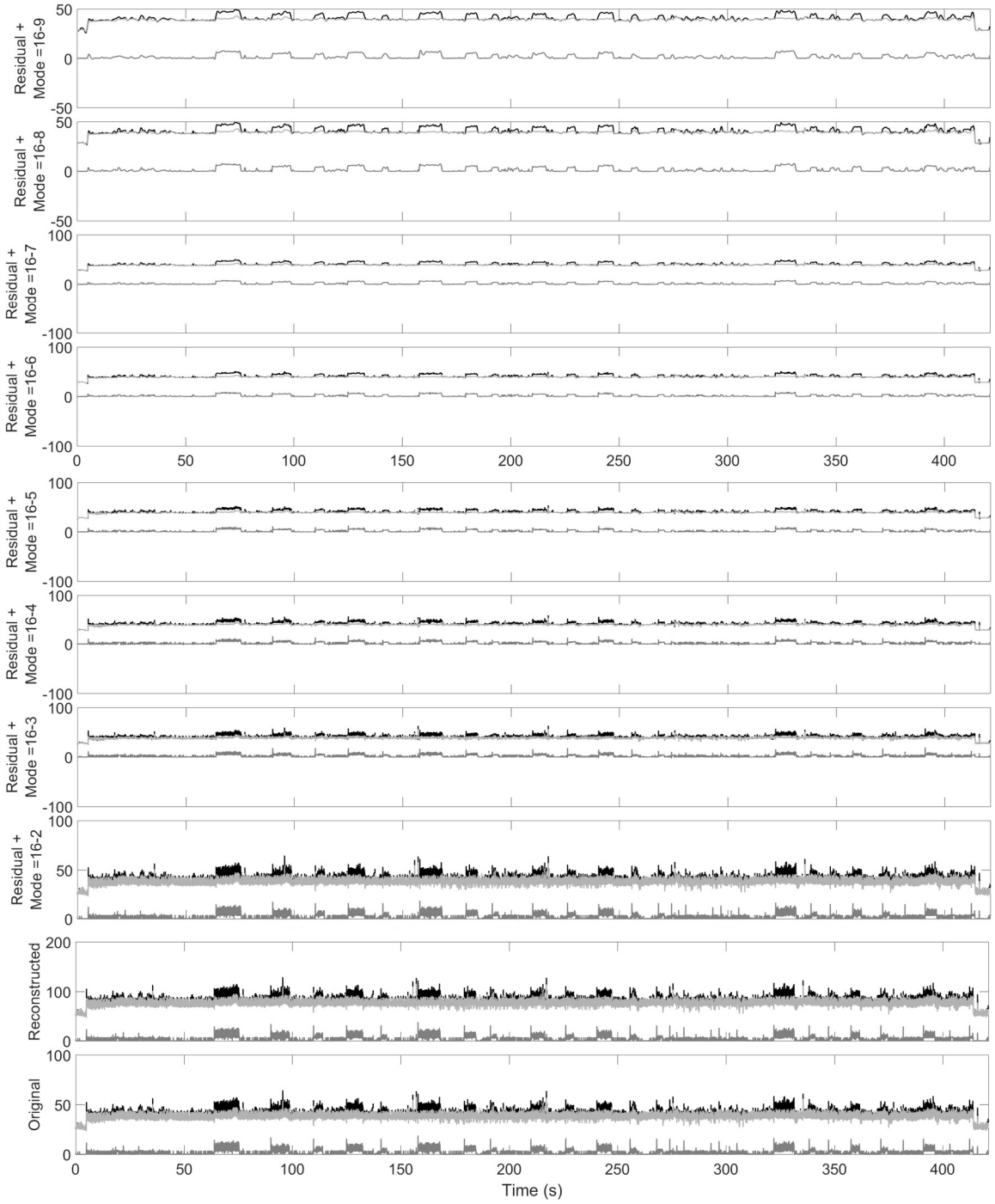


Fig. 50. IMF reconstruction (second half) for GAMESS-1L2Y collected on Haswell with the bunch affinity. In each plot, total power (black) is shown with core (light gray) and DRAM (gray) power traces (the total of both sockets).

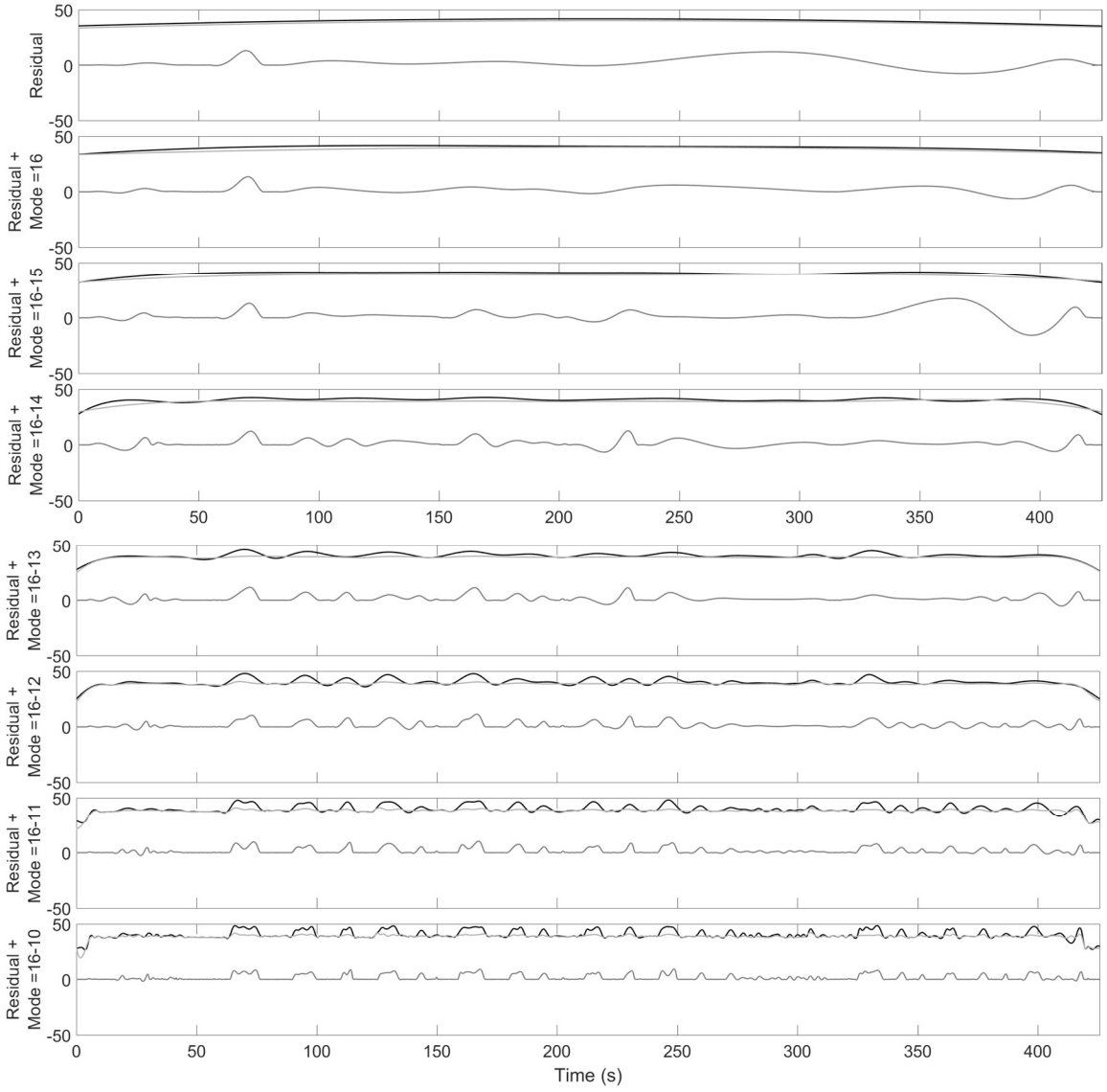


Fig. 51. IMF reconstruction (first half) for GAMESS-1L2Y collected on Haswell with the scatter affinity. In each plot, total power (black) is shown with core (light gray) and DRAM (gray) power traces (the total of both sockets).

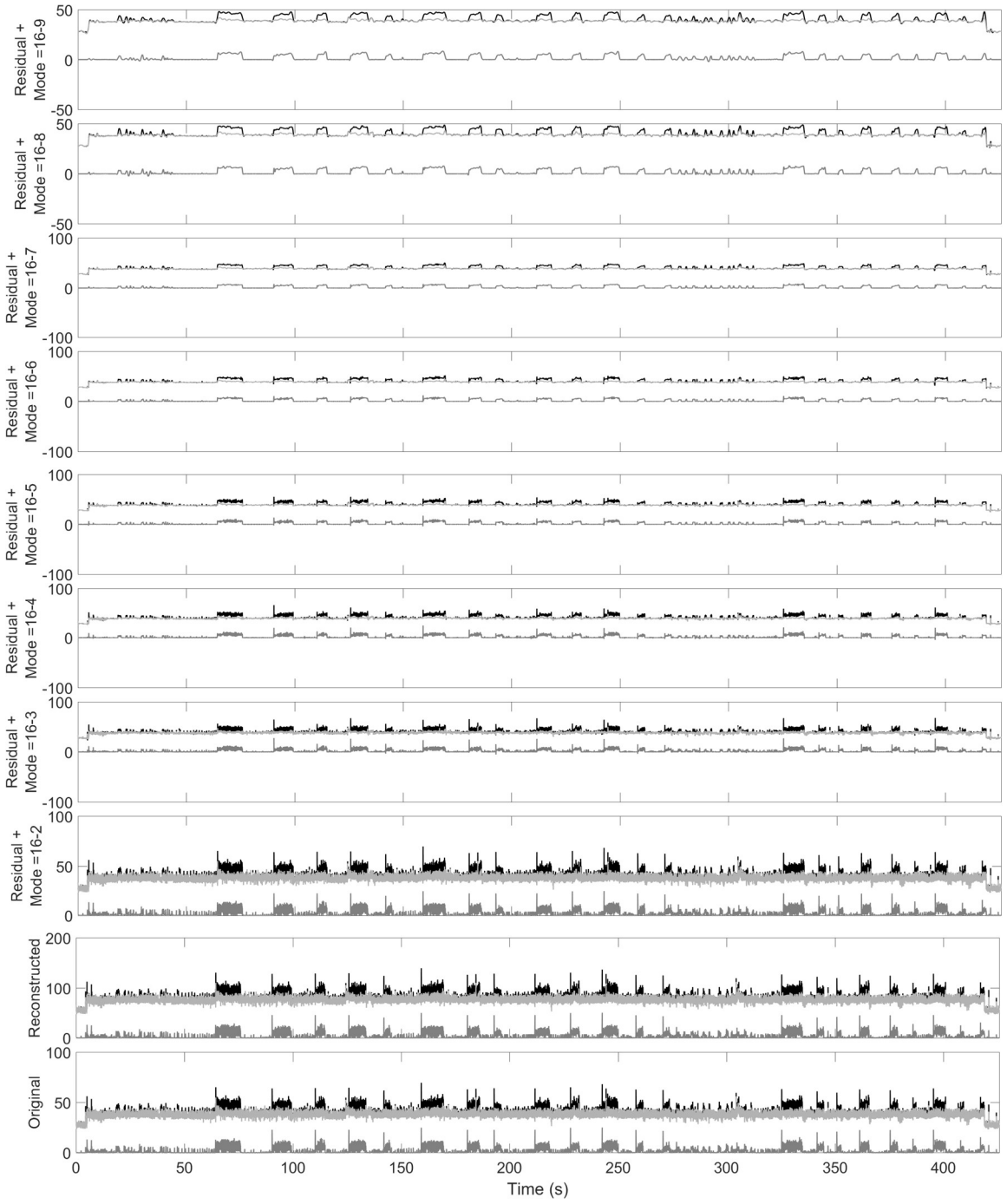


Fig. 52. IMF reconstruction (second half) for GAMESS-1L2Y collected on Haswell with the scatter affinity. In each plot, total power (black) is shown with core (light gray) and DRAM (gray) power traces (the total of both sockets).

The IMF's show that the scatter affinity is encountering power fluctuations more often than the bunch affinity. This may be observed in the original power traces, where the scatter affinity has many spikes in power just as the DRAM power is ramping up; this occurs during every period where DRAM power is high, although this does not occur under the bunch affinity, thus it may be caused by communication between each socket. In the IMF's, modes 1 and 2 for the scatter affinity have many more peaks which coincide with the peaks observed in the original trace. Far fewer peaks may be observed in the bunch affinity. In fact, most of the IMF's for the scatter affinity are larger (high amplitude) than those for bunch.

Considering it was shown that DRAM power was indeed coming from only one socket with the bunch affinity, it may be concluded that the scatter affinity is encountering additional conflicts as a result of synchronization between sockets. However, it may also be concluded that this performance loss is negligible, since the time-to-solution is equivalent when using bunch or scatter affinity.

It is also of interest to note here that the number of modes for each source trace was the same. For all IMF's explored in this dissertation, that is, for CoMD, GAMESS, and NPB across several platforms – Borges, Turing, Bolt, Marquez, KNC, and KNL – the number of modes for each respective source of power for the trace: core, DRAM, and uncore are always the same. However, different configurations (such as varying the number of cores, and clock-rate) can result in a different number of modes, although this may be a factor of time-to-solution since EMD is dependent on the total number of samples (which is dependent on time for a constant sampling rate, as used in this work).

7.1.2.2 Trace Reconstruction

The EMD analysis procedure refines a time-series into a collection of IMF's; the sum of these IMF's is the original time-series to the precision of the floating-point computations, although this error is negligible as discussed in [40]. Each IMF may be considered as the difference in amplitude from the mean trend (a.k.a. the residual) for processes occurring on the time-scale of the IMF. The time-scale is the difference in time between inflection points in the IMF, see [40].

The idea of accumulating IMF's is of interest since the sum of several IMF's may be more meaningful than an individual IMF. Consider Figs. 49 to 52 the reconstruction of the IMF's for GAMESS 1L2Y under the bunch and scatter affinities; total, core, and DRAM power are shown similar to Figs. 45 and 47. As shown here, the core and DRAM IMF's reconstruct into each respective trace, and the sum of the reconstructed traces, whether as the sum of the IMF's for total power or the sum of the IMF's for core and DRAM power (or even per socket-source), the resulting time-series matches the original. Thus, applying EMD to the individual source traces (and then adding the respective IMF's) results in the same IMF's as if decomposing the total power trace. Of course, decomposing the total power trace cannot distinguish between core and DRAM power, although the net sum of each will be represented in the resulting IMF's for total power, as shown in the prior section.

7.2 Multinode Analysis

As shown in the previous section, EMD may be successfully applied to individual traces representing the same physical process. In this section, the same analysis is

performed for executions with multiple nodes. The difference between multisocket and multinode executions is subtle; both include multiple devices working in unison to accomplish a task, however, the multisocket measurements are collected from the same timer and the multinode measurements are collected from different timers, one for each node. Since the multisocket traces were all the same length and sampled at the same time, those traces were easier to analyze. The multinode traces are not the same length, thus they will need to be aligned such that the traces may be summed to create the total power trace of the execution.

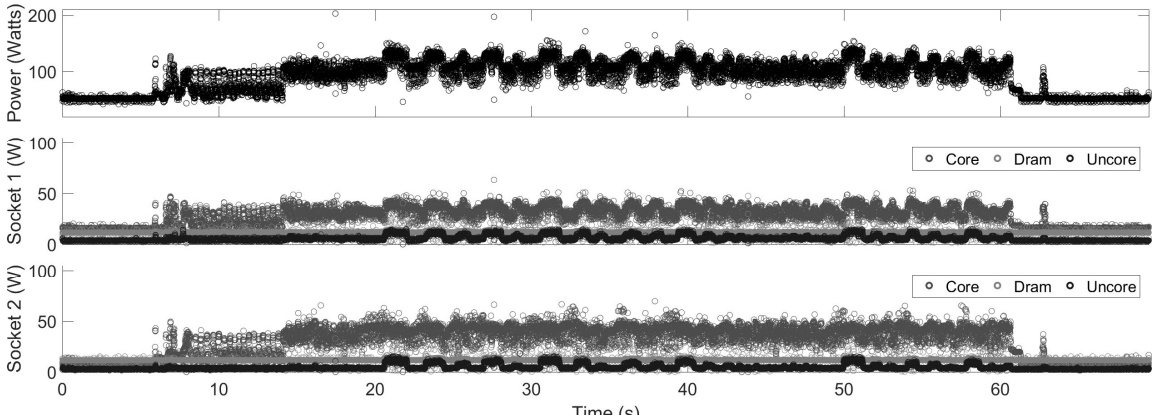
7.2.1 Power Traces

Figures 53 and 54 presents the four traces measured for the execution of GAMESS - 1L2Y on four nodes of the Turing cluster (20 cores per node); Figs. 55 and 56 shows the four traces for GAMESS-20w. From the view given in these plots, it would appear that each trace is already aligned; however, as will be shown next, slight adjustments are necessary.

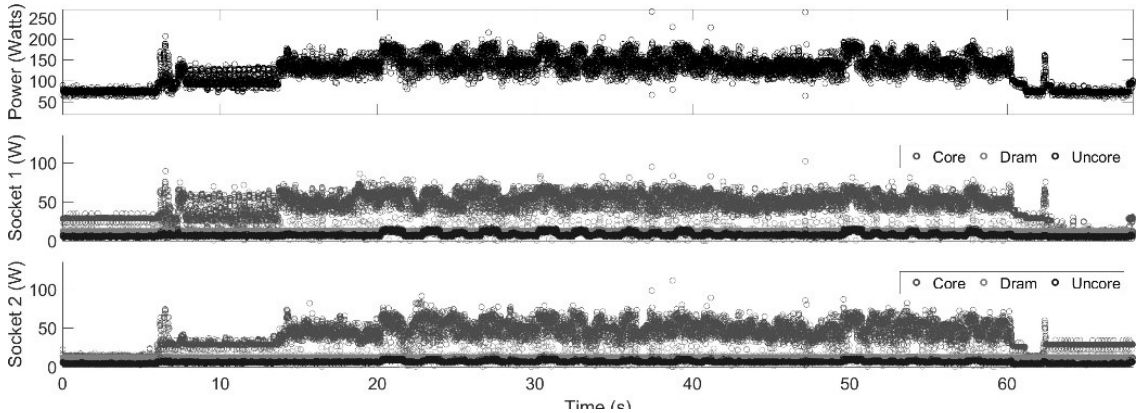
7.2.2 Trace Synchronization

To synchronize the traces, a delay was introduced to the power trace on each node except the first, i.e. the shorter traces. Figure 57 presents the power traces for GAMESS on Turing using 4 nodes for the 1L2Y and 20w problems. Each plot (in a & b) shows the alignment of each respective node: the top plot shows Node 2 vs Node 1, middle shows Node 3 vs Node 1, and the last shows Node 4 vs Node 1.

To align the traces, cross-correlation was attempted. The use of cross-correlation

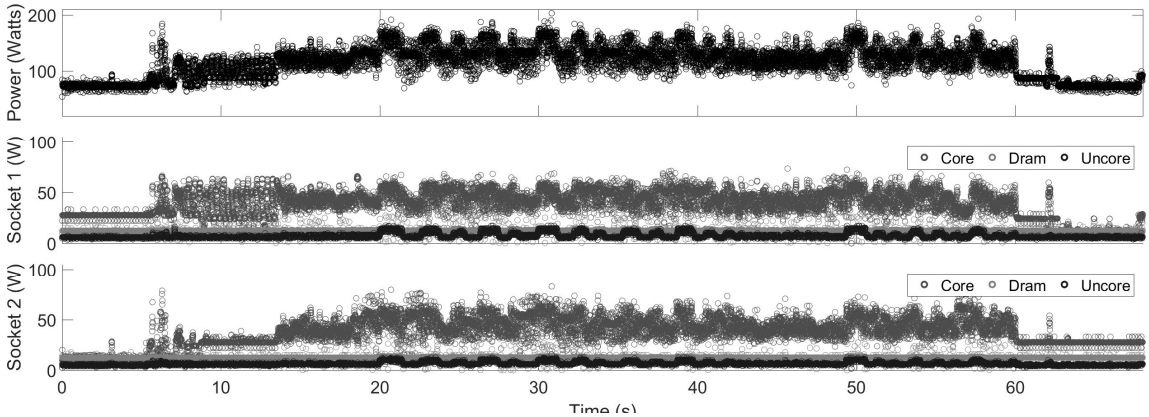


(a) Turing - GAMESS - 1L2Y - Scatter - Node 1

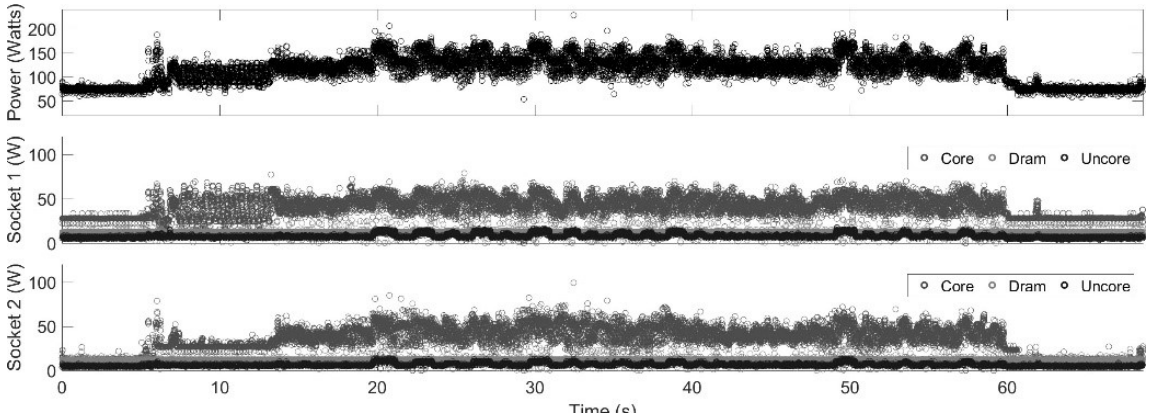


(b) Turing - GAMESS - 1L2Y - Scatter - Node 2

Fig. 53. Power traces of GAMESS-1L2Y collected on 4 Ivy-Bridge nodes showing total vs. per socket, per source traces while varying MPI thread affinity. For each plot (a & b), there are three subplots: top shows total power, middle and bottom shows socket 1 & 2 power respectively where core power is shown in gray, DRAM power in light gray, and uncore in black.

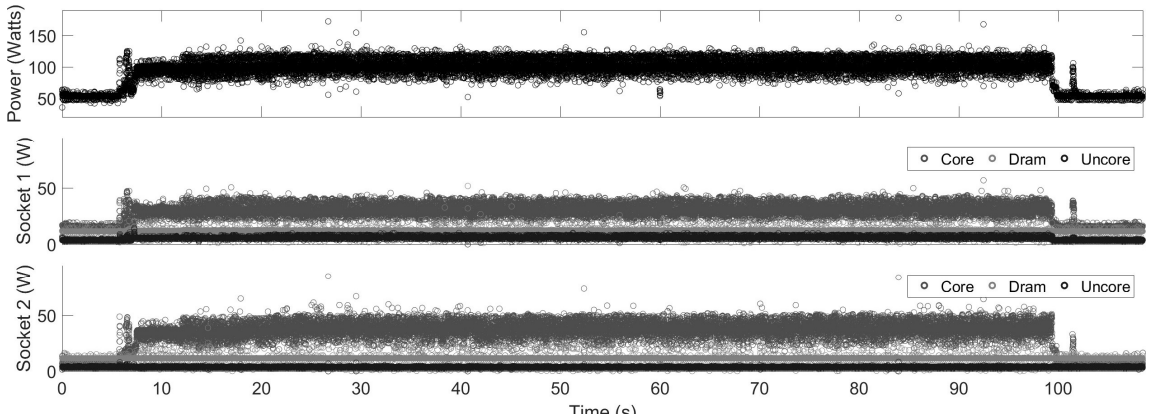


(a) Turing - GAMESS - 1L2Y - Scatter - Node 3

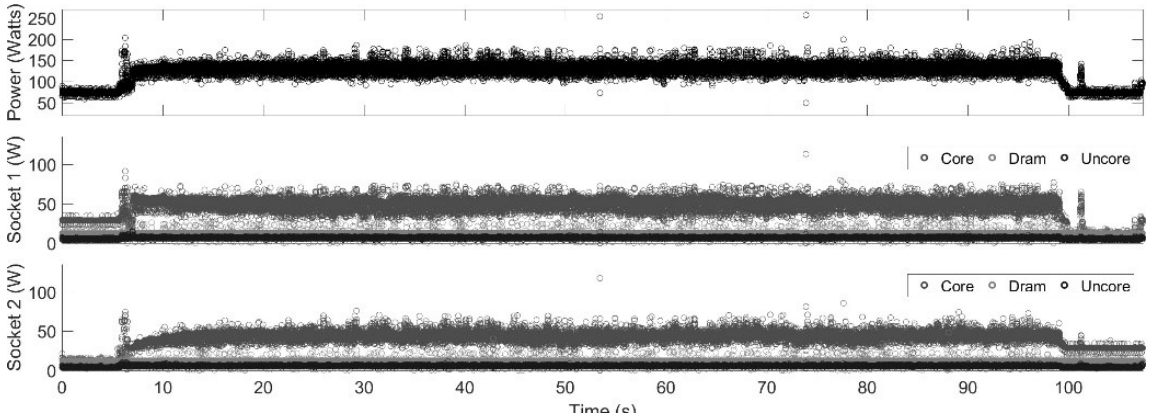


(b) Turing - GAMESS - 1L2Y - Scatter - Node 4

Fig. 54. Power traces of GAMESS-1L2Y collected on 4 Ivy-Bridge nodes showing total vs. per socket, per source traces while varying MPI thread affinity. For each plot (a & b), there are three subplots: top shows total power, middle and bottom shows socket 1 & 2 power respectively where core power is shown in black, and DRAM power in gray.

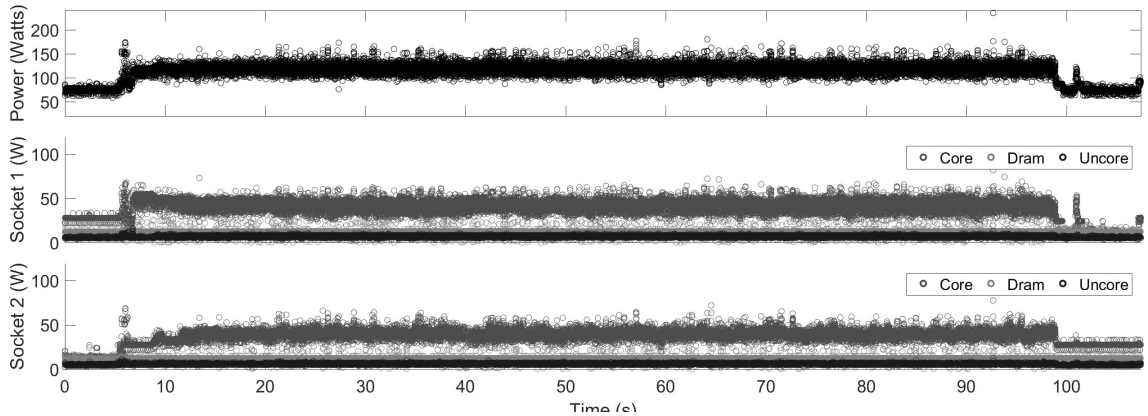


(a) Turing - GAMESS - 20w - Scatter - Node 1

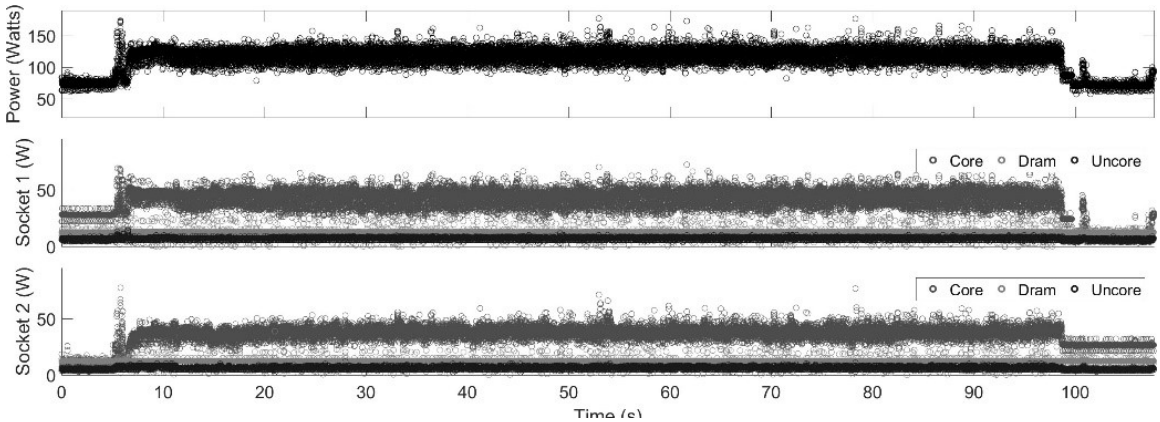


(b) Turing - GAMESS - 20w - Scatter - Node 2

Fig. 55. Power traces of GAMESS-20w collected on 4 Ivy-Bridge nodes showing total vs. per socket, per source traces while varying MPI thread affinity. For each plot (a & b), there are three subplots: top shows total power, middle and bottom shows socket 1 & 2 power respectively where core power is shown in gray, DRAM power in light gray, and uncore in black.

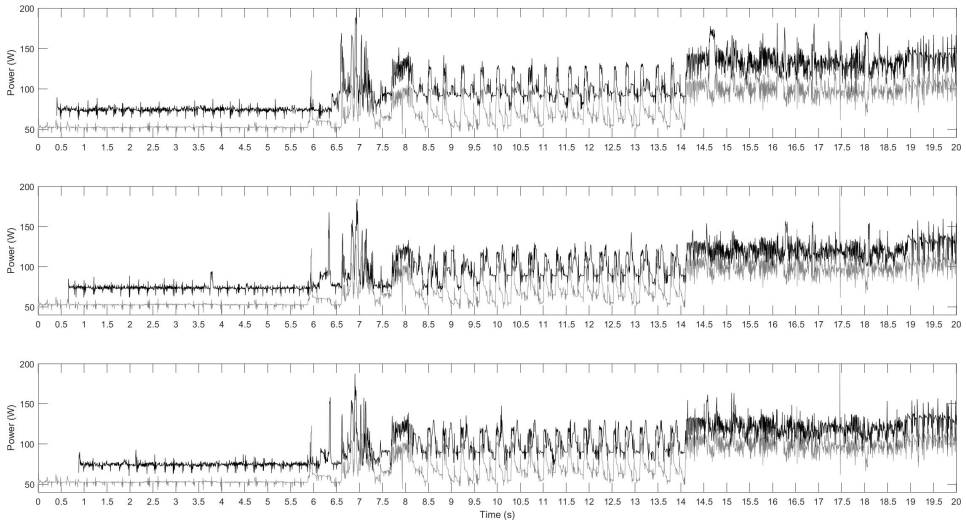


(a) Turing - GAMESS - 20w - Scatter - Node 3

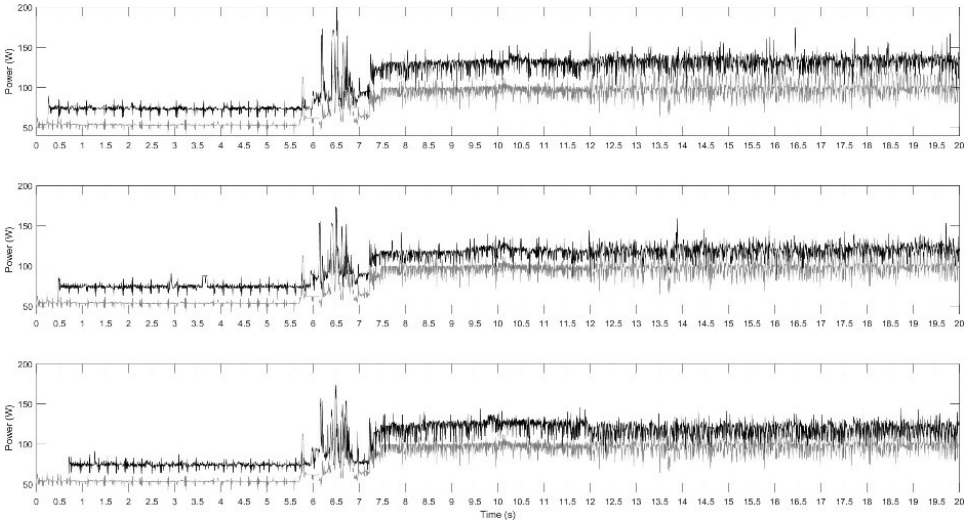


(b) Turing - GAMESS - 20w - Scatter - Node 4

Fig. 56. Power traces of GAMESS-20w collected on 4 Ivy-Bridge nodes showing total vs. per socket, per source traces while varying MPI thread affinity. For each plot (a & b), there are three subplots: top shows total power, middle and bottom shows socket 1 & 2 power respectively where core power is shown in black, and DRAM power in gray.

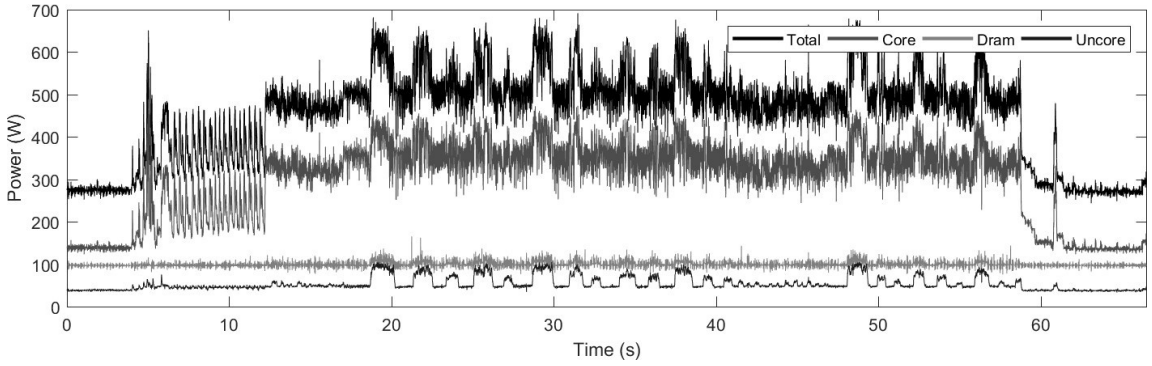


(a) Turing - GAMESS - 1L2Y

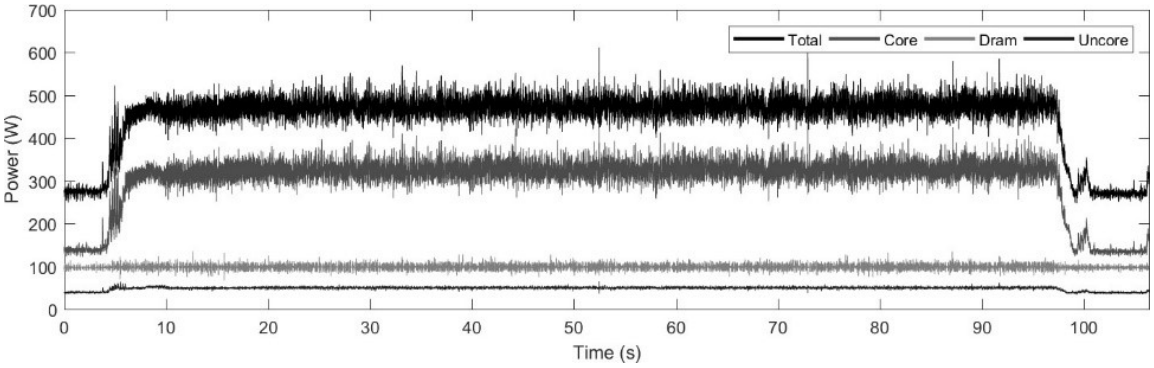


(b) Turing - GAMESS - 20w

Fig. 57. Power traces of GAMESS, 1L2Y and 20w, collected on Ivy-Bridge on 4 nodes. Each plot shows node power (2, 3, or 4) in black compared to the power on the first node in gray. Traces have been manually aligned.



(a) Total Power – Turing - GAMESS - 1L2Y



(b) Total Power – Turing - GAMESS - 20w

Fig. 58. Total power traces for GAMESS, 1L2Y and 20w, collected on 4 Ivy-Bridge nodes after alignment and cropping. Total power is shown in black, core in gray, DRAM in light gray, and uncore in dark gray.

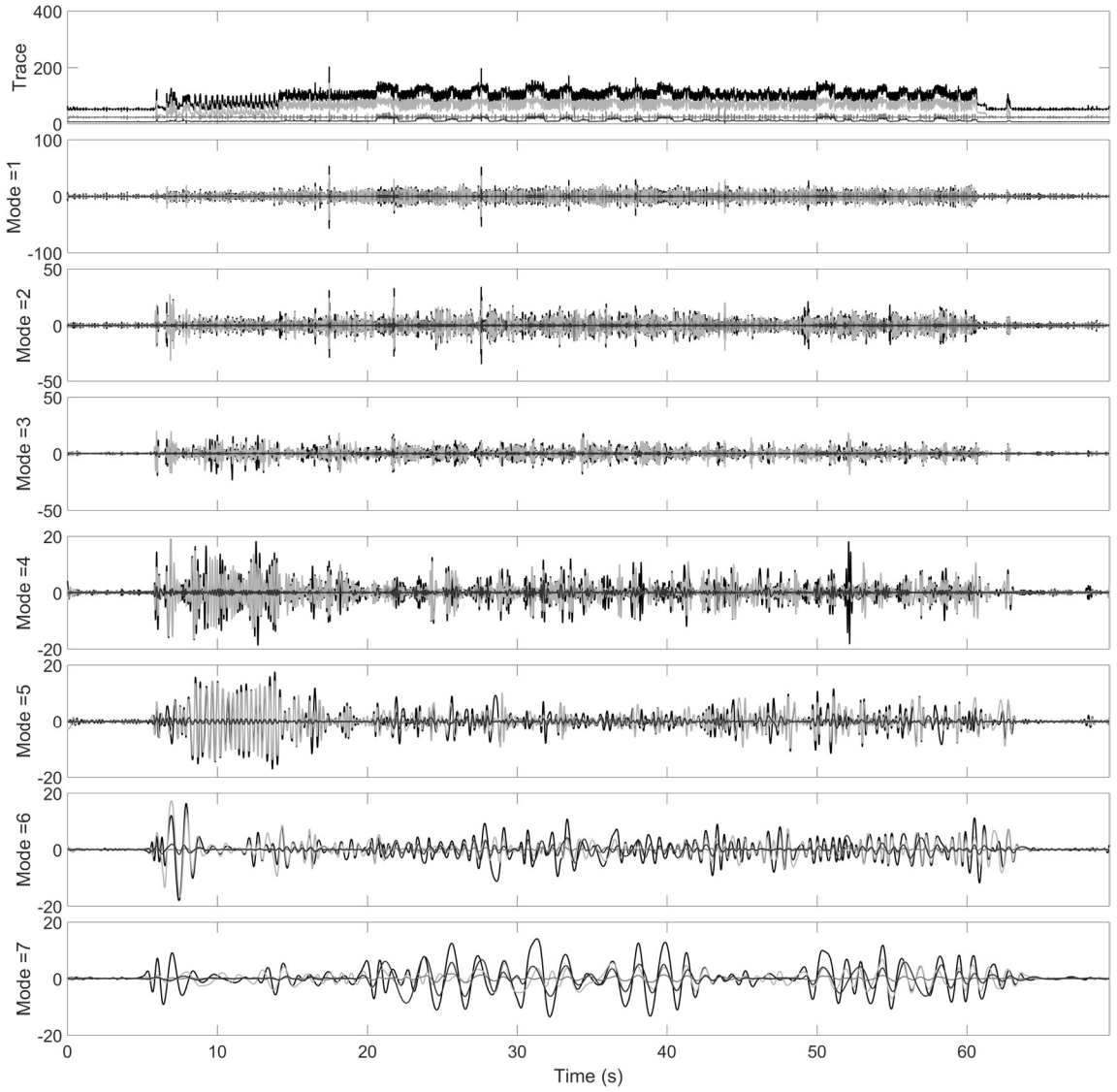


Fig. 59. (Node 1) Power trace and the first half of the IMF's for GAMESS-1L2Y collected on 4 Ivy-Bridge nodes. In each plot, total power (black) is shown with core (light gray), DRAM (gray), and uncore (dark gray) power traces.

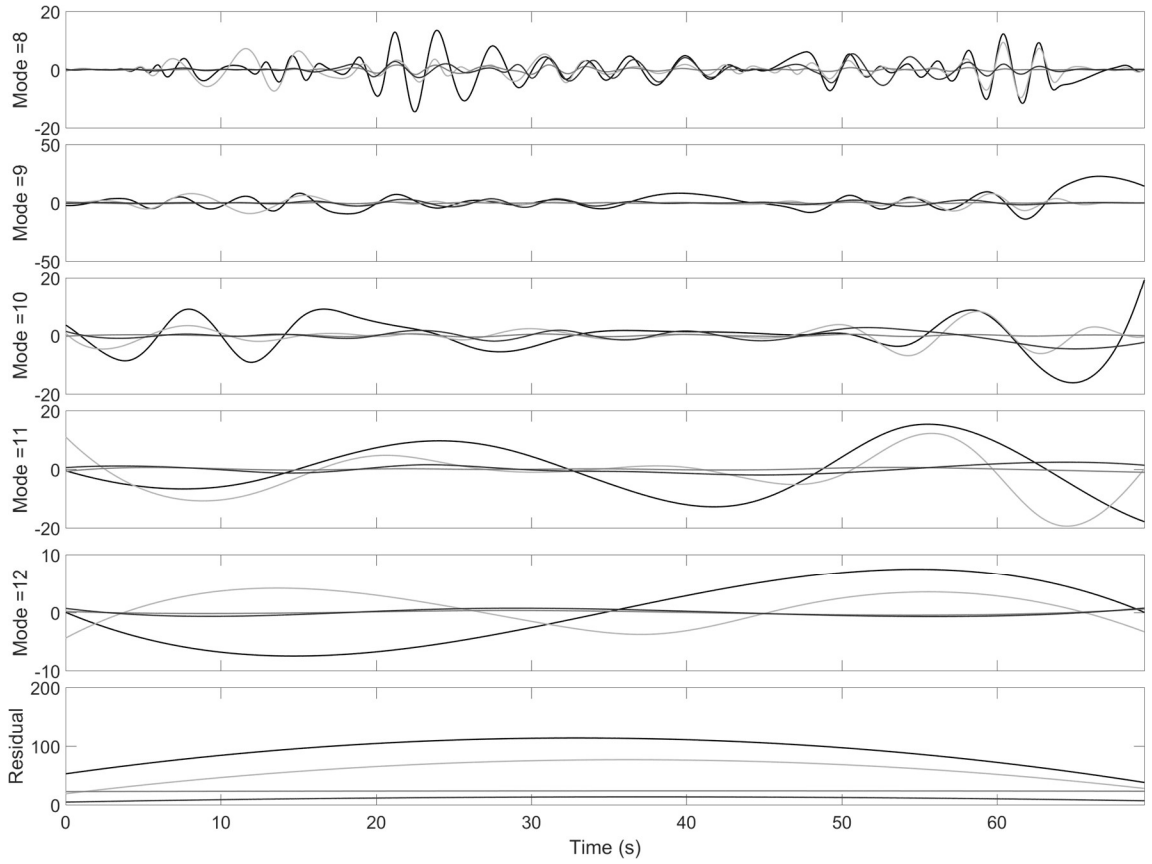


Fig. 60. (Node 1) Residual and the second half of the IMF's for GAMESS-1L2Y collected on 4 Ivy-Bridge nodes. In each plot, total power (black) is shown with core (light gray), DRAM (gray), and uncore (dark gray) power traces.

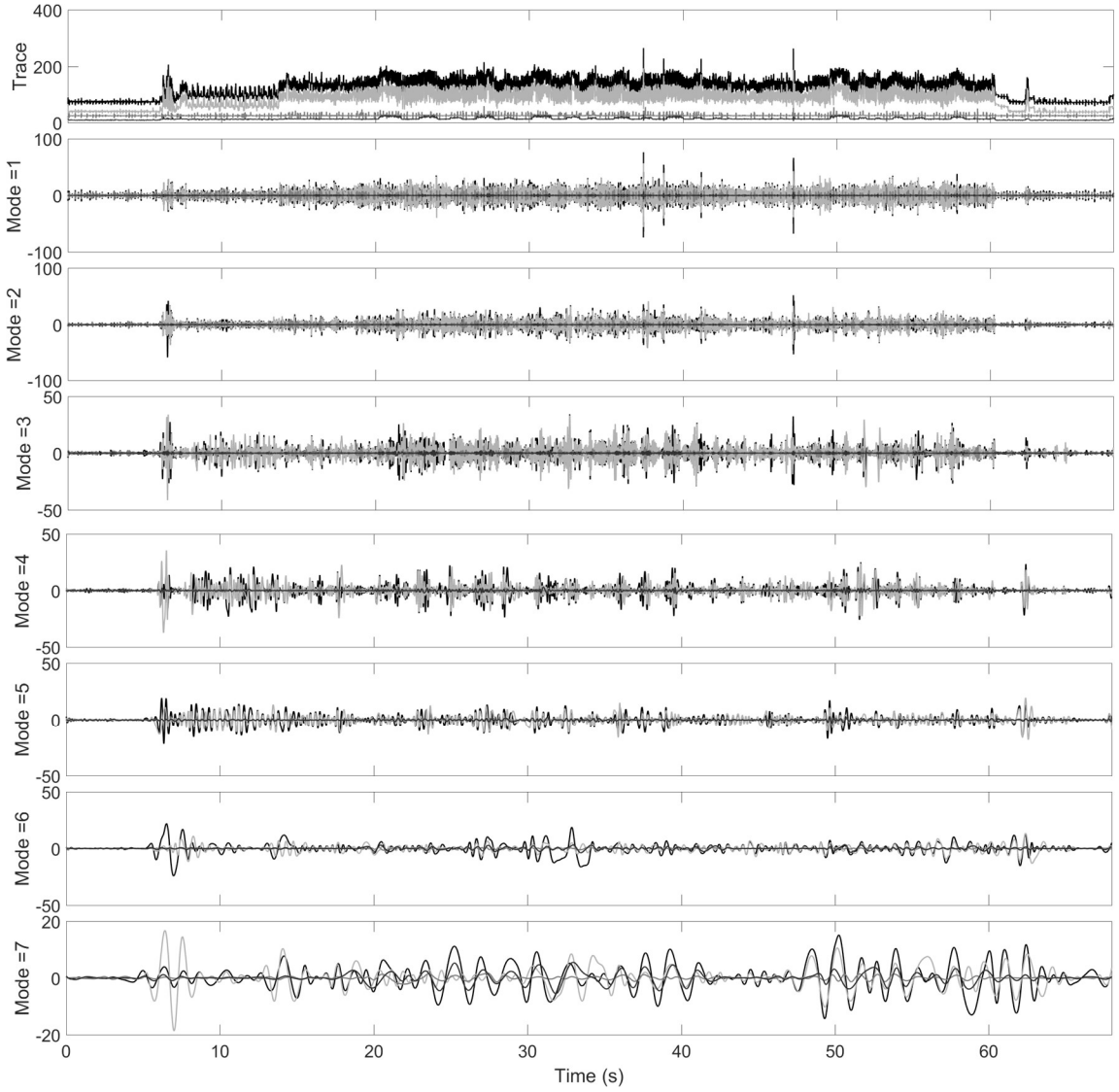


Fig. 61. (Node 2) Power trace and the first half of the IMF's for GAMESS-1L2Y collected on 4 Ivy-Bridge nodes. In each plot, total power (black) is shown with core (light gray), DRAM (gray), and uncore (dark gray) power traces.

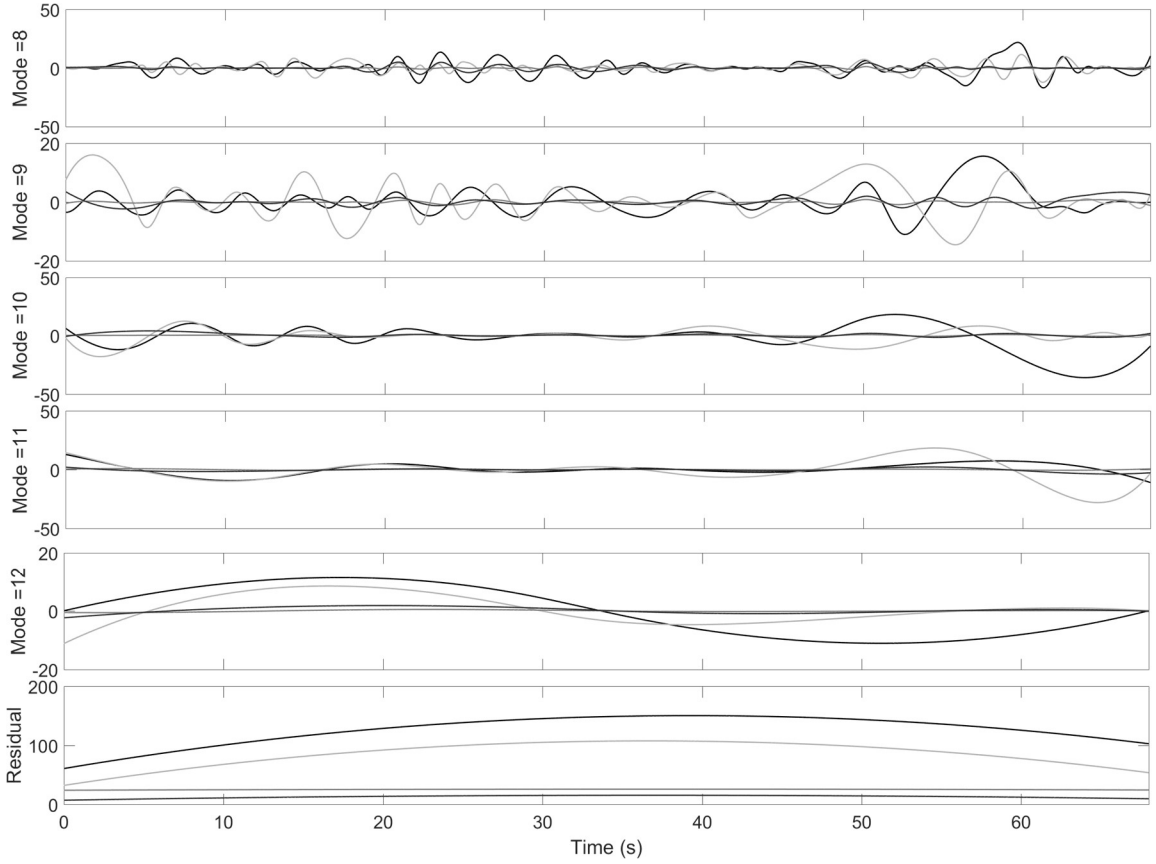


Fig. 62. (Node 2) Residual and the second half of the IMF's for GAMESS-1L2Y collected on 4 Ivy-Bridge nodes. In each plot, total power (black) is shown with core (light gray), DRAM (gray), and uncore (dark gray) power traces.

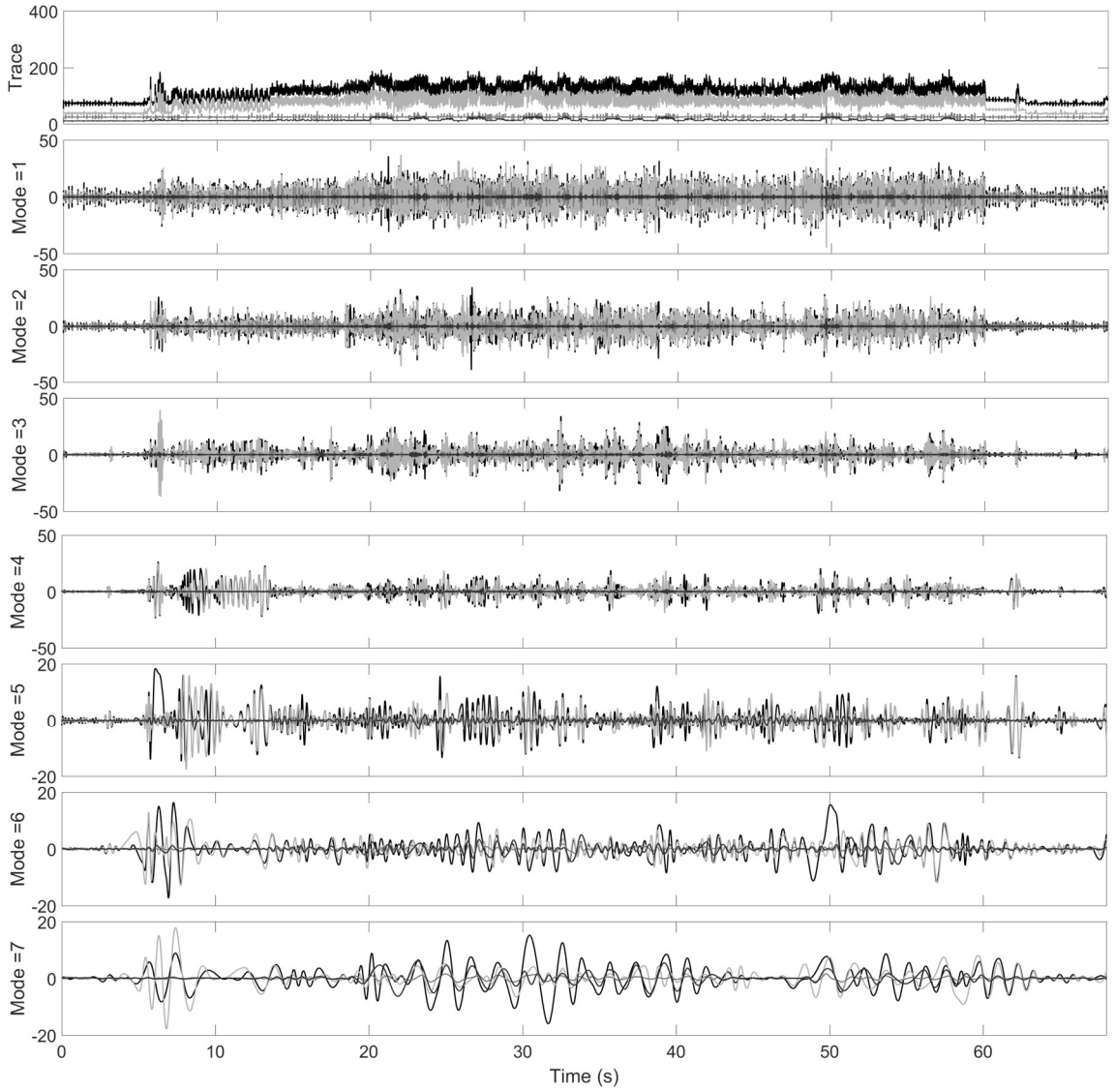


Fig. 63. (Node 3) Power trace and the first half of the IMF's for GAMESS-1L2Y collected on 4 Ivy-Bridge nodes. In each plot, total power (black) is shown with core (light gray), DRAM (gray), and uncore (dark gray) power traces.

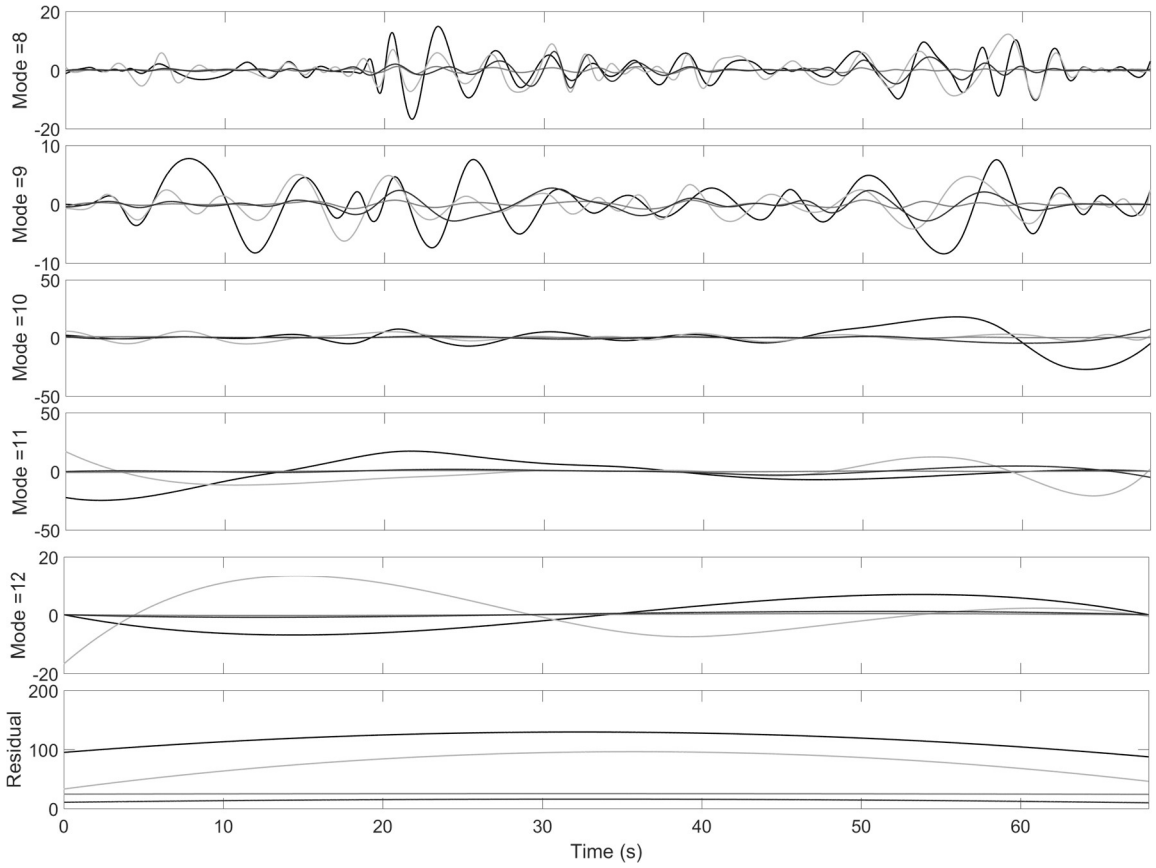


Fig. 64. (Node 3) Residual and the second half of the IMF's for GAMESS-1L2Y collected on 4 Ivy-Bridge nodes. In each plot, total power (black) is shown with core (light gray), DRAM (gray), and uncore (dark gray) power traces.

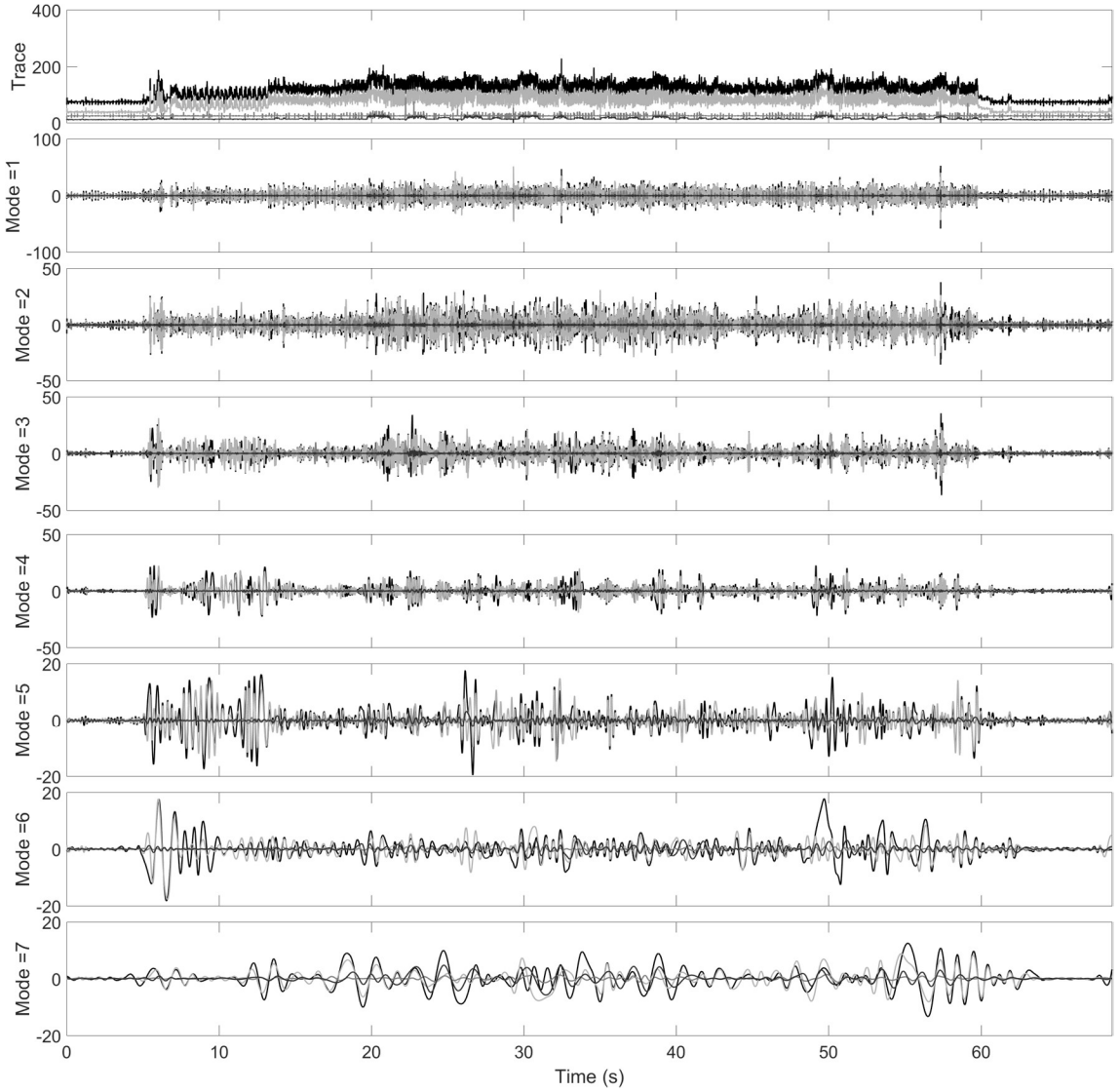


Fig. 65. (Node 4) Power trace and the first half of the IMF's for GAMESS-1L2Y collected on 4 Ivy-Bridge nodes. In each plot, total power (black) is shown with core (light gray), DRAM (gray), and uncore (dark gray) power traces.

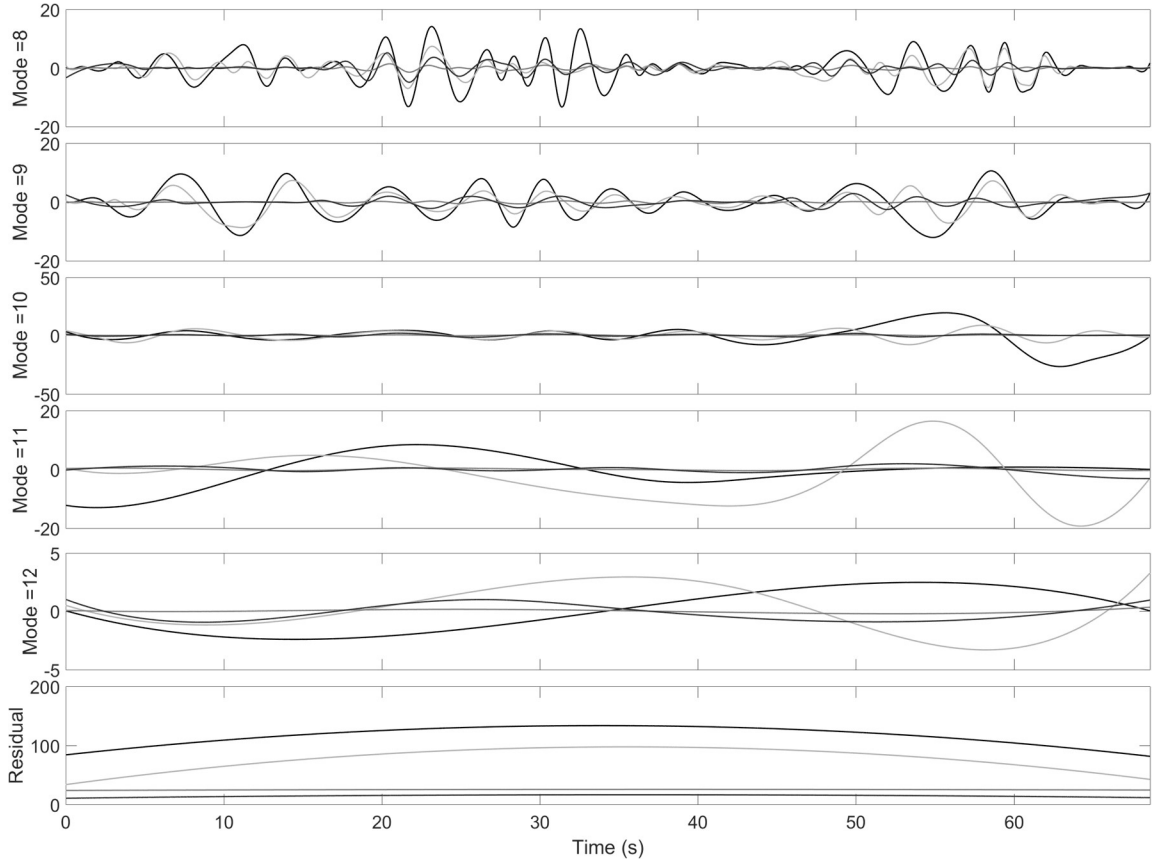


Fig. 66. (Node 4) Residual and the second half of the IMF's for GAMESS-1L2Y collected on 4 Ivy-Bridge nodes. In each plot, total power (black) is shown with core (light gray), DRAM (gray), and uncore (dark gray) power traces.

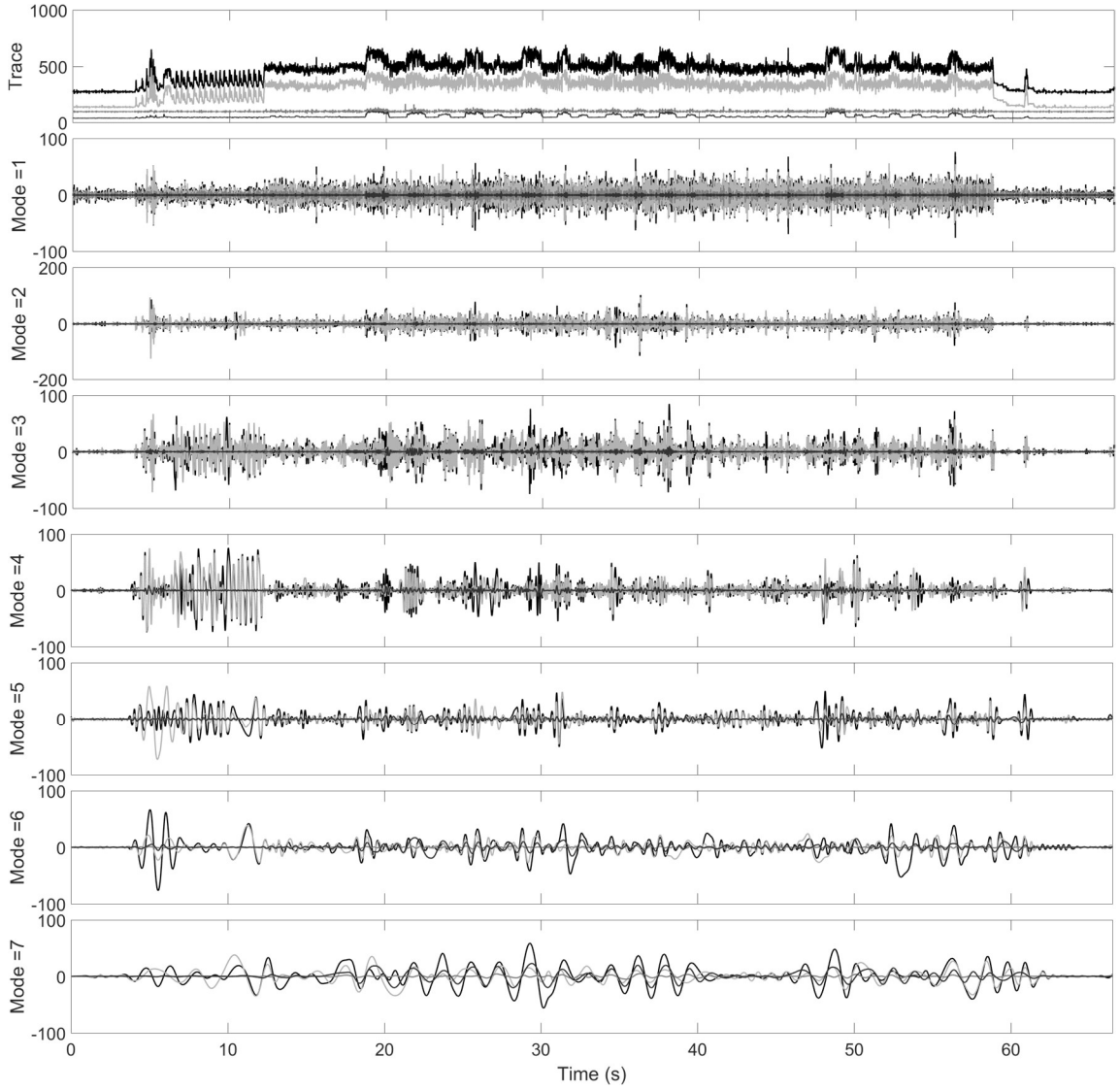


Fig. 67. (Total) Power trace and the first half of the IMF's for GAMESS-1L2Y collected on 4 Ivy-Bridge nodes. In each plot, total power (black) is shown with core (light gray), DRAM (gray), and uncore (dark gray) power traces.

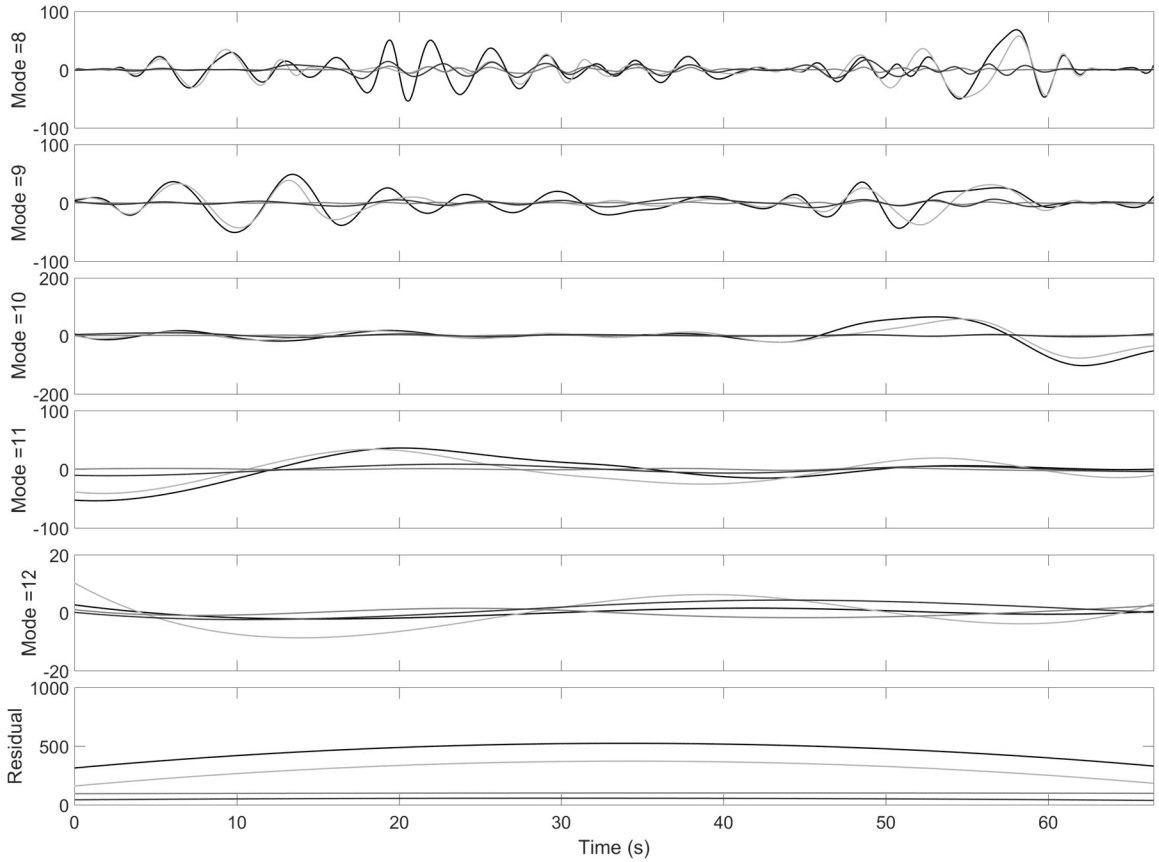


Fig. 68. (Total) Residual and the second half of the IMF's for GAMESS-1L2Y collected on 4 Ivy-Bridge nodes. In each plot, total power (black) is shown with core (light gray), DRAM (gray), and uncore (dark gray) power traces.

to synchronize signals is suggested in Matlab [73]. However, power traces have been found to be difficult to properly align using this method because large spurious correlations occur frequently. Cross-correlation would have provided a method for automating the synchronization process. Instead, the alignment delay was found by increasing the delay until a feature, e.g. a rapid increase in power draw, is aligned for the two traces; this was performed by hand. For 1L2Y in Fig. 57, the synchronization feature is just past 14 seconds,

and for 20w, the feature is between 7 and 7.5 seconds.

Once the traces have been synchronized, they may be cropped to ensure each trace has the same number of samples. Then the traces for each node may be accumulated to create the total execution power trace, shown in Fig. 58. With the traces prepared, EMD may now be applied.

7.2.3 EMD on Multinode Traces

Figures 59 and 60 present the IMF's calculated after EMD was applied to the total, core, DRAM, and uncore power traces for node one. Figures 61 and 62 present the IMF's calculated after EMD was applied to the power traces for node two. Figures 63 and 64 present the IMF's calculated after EMD was applied to the power traces for node three. Figures 65 and 66 present the IMF's calculated after EMD was applied to the power traces for node four. Figures 67 and 68 present the IMF's calculated after EMD was applied to the total, core, DRAM, and uncore power traces for the total power summed for all four nodes.

By separating the power trace into sources, the role of each component with respect to the IMF's becomes clearer, although interpreting the meaning of each IMF remains unclear. Take note that the total power trace (Figs. 67 and 68) closely resembles the traces for each of the four nodes, so discussions of individual modes will pertain to this set of IMFs' although the same features may be found in the IMF's for each node as well.

There are two features for the 1L2Y problem size that makes GAMESS a particularly interesting application to investigate. The first can be found between 6 to 12 seconds in Figs. 67 and 68, where there are sharp *ridges* that have formed. This feature is clearly visible in the core power trace. The second feature occurs many times throughout execution

and is first observed near 20 seconds (2nd tick mark along x-axis) and may be described as *humps*. This feature is clearly visible in the uncore power trace.

Consider the ridges; the IMF's for this feature have manifested in all of the modes, but are most striking in modes 3, 4, and 5. What is most interesting about this feature is that it shows a portion of execution with a very high flop-per-byte ratio. Compare the core to the DRAM and uncore power traces. There are few large latencies incurred, indicated by the fact that DRAM and uncore power remain low, yet the core power throttles to form the ridges.

Now consider the humps. In modes 3 and 4, groups of high amplitude oscillations are visible for the duration of the *hump* in uncore power; further, the uncore IMF's in these modes show consistent oscillations (although the amplitude is small compared to core). In modes 7 and 8, the oscillations for uncore are more pronounced (and correlate well with the IMF's for total power). Depending on the operations occurring during these periods, these modes may be useful for a forecasting model which could be used to predict/detect when to apply DVFS during execution for minimal impact; since uncore power is increasing dramatically, it is expected that a significant amount of data movement is involved, and DVFS is most beneficial during periods of high data movement.

7.3 Concluding Remarks

In this chapter, an in-depth view of EMD on power traces has been performed. Total power was compared to the parts that compose it: core, DRAM, and uncore power. For the individual sockets, EMD can be applied to the traces at any level (per socket & source, per socket, per source, or total). Of course, EMD on the total trace cannot distinguish between

core and DRAM power draws, however, whether EMD is applied to the traces (e.g. core, DRAM, uncore) and then summed or the traces are summed and then EMD is applied, the 172 resulting IMF's will be the same and these IMF's can be used to reconstruct the original trace. If a forecasting model were applied to one or more IMF's, it could be used for predicting core, DRAM, and uncore power consumption during the execution. This could then be used to aid a DVFS strategy, or predict the overall energy consumption of the execution. The need to predict energy may be required by future Exascale HPC systems adopt user-specified energy budgets to run large-scale applications.

CHAPTER 8

CONCLUSIONS

This dissertation has investigated power and energy consumption on a number of hardware platforms, parallel applications, and for a number of execution characteristics. Homogeneous and heterogeneous platforms have been considered, as well as multicore and manycore devices.

8.1 Summary

Power capping and energy saving techniques using DVFS on CPU and Xeon Phi accelerators have been explored. The findings in this work agree with the literature, in that DVFS is beneficial for applications that are not bounded by floating-point performance and may be used sparingly otherwise to reduce the energy consumption of the execution. However, DVFS is fickle and often introduces performance bottlenecks; these may be due to state switching or the reduced clock-rate, and often is not worth the performance loss. Thus, a better strategy is to implement energy capping where applications must abide by an energy cap, but may use any amount of power in between. However, given an energy cap, optimization will be more important as to not waste energy and a system will need a metric to weigh the general energy efficiency of an application.

This dissertation investigated available power and performance models and applied them to benchmarks and real-world parallel applications for heterogeneous and homogeneous multicore and manycore computing platforms to predict energy consumption. A model was proposed which described an execution by the phases, either computation or data movement, for heterogeneous and homogeneous platforms. Although

the model was able to capture the trends of the executions considered, measurement was not easily extendable to large-scale applications; particularly cases where code has been optimized for different architectures and must be inserted within each optimized version, thereby introducing performance bottlenecks. It was clear a less intrusive method was needed.

The Empirical Mode Decomposition and Hilbert-Huang Transform analysis technique has been applied in innovative ways to model, analyze, and visualize power and energy measurements. The approach has been used to visualize power traces using the relation of energy, frequency, and time. The technique has been used to identify overlap between computation and communication and quantify contributions of each for a specific application-platform combination. The approach has also been used to analyze segmented power measurements, and model the general trend of an execution. Further, it is shown in this work that the EMD method may be applied to the total power (sum of individual power sources), or the individual power traces (core, DRAM, uncore for each socket/device) so long as they come from the same execution (e.g. multiple sockets, nodes).

Probability distributions were introduced in this work to represent power and energy traces, thereby providing an alternative means of modeling power and energy consumption. The distribution models retain the fact that power is not constant over time, and also retains the fact that average power is an excellent approximation for most workloads and systems. Further, these distributions may be used to define the explicit costs of a workload for a given computing platform.

8.2 Findings

The findings of this work are as follows:

- Static power draw is the leading cause of all power/energy consumption in HPC – this includes power consumed independent of the number of cores, such as the power needed for memory and underlying components, such as data buses
  - Improving measurement of these components will be needed in future systems (some chips only provide core and DRAM power traces and neglect uncore, for example)
  - Future hardware will need to be more adaptive in how power is consumed for optimal energy-efficiency; minimizing the power consumption for idle hardware is the best way to improve efficiency
  - Future software will need to be more parallel to better utilize the hardware; this includes lowering the memory-footprint required per core
- DVFS is an effective power capping tool, however, has significant potential to impede performance if used incorrectly
  - An example of "correct" usage is as follows: apply DVFS during data transfer phases of execution, such as transfers between devices or communication between nodes, although the challenge remains to "detect" these phases in real-time
  - An example of "incorrect" usage is to blindly apply DVFS to the entire execution; performance loss will be higher than 10% (the maximum accepted loss in this area of research)
- EMD provides a wealth of data by decomposing the power/energy trace into a series of intrinsic mode functions, or oscillations, where each new series spans an interval of time

(from 2× the sampling rate (lowest mode) up to the total time elapsed (residual))

- This is the first time that EMD has been applied to power/energy traces
- More analysis is needed to uncover the explicit meaning of these IMF's for the purposes of phase detection and in the interest of power and energy capping
- Cross-correlation is a good first step to this further analysis, but these traces will also need to be processed by other time-series analysis techniques as well, such as discrete wavelet analysis and the Fourier decomposition method.

- Power and energy traces (the time-series) can be represented using multi-modal Normal distributions
  - Distributions be used to predict energy and compare application usage across platforms, or the typical energy consumption of a hardware platform for a wealth of applications
  - This representation of power/time traces is novel

8.3 Future Work

There is much to be done in the future along this line of research. The EMD/HHT method decomposes a time-series into intrinsic mode functions, but the interpretation of these modes is not straightforward. For this reason, the residual was primarily investigated and modeled; however, a wealth of information is still hidden in these modes. The Energy-Frequency-Time plots can be used to show when performance-bottlenecks have been encountered due to the relative energy consumption.

Further analysis into the combination of IMF modes is needed. As shown in the EMD reconstruction, the sum of even a few modes brings shape to the trace by removing

noise (including other IMF modes considered "noise" based on the time-scale(s) of interest). As proven in other research efforts (such as sea level rise analysis), analysis of combinations of IMF's proves to be more effective than individual modes. This would be the best next step toward understanding power and energy using EMD/HHT analysis.

REFERENCES

[1] perf: Linux profiling with performance counters, 2015. https://perf.wiki.kernel.org/index.php/Main_Page.

[2] Power capping framework, 2016. https://www.kernel.org/doc/Documentation/power/powercap/powercap.txt.

[3] L. Adhianto and B. Chapman. Performance modeling of communication and computation in hybrid *MPI* and Openmp applications. *Simulation Modelling Practice and Theory*, 15(4):481 – 491, 2007. Performance Modelling and Analysis of Communication Systems.

[4] E. Apra, M. Klemm, and K. Kowalski. Efficient implementation of many-body quantum chemical methods on the Intel® Xeon Phi™ coprocessor. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '14, pages 674–684, Piscataway, NJ, USA, 2014. IEEE Press. http://dx.doi.org/10.1109/SC.2014.60.

[5] Michaela Barth. Best practice guide Intel Xeon Phi v1.1, 2014. http://www.prace-ri.eu/IMG/pdf/Best-Practice-GuideIntel-Xeon-Phi.pdf.

[6] M. Bernaschi, M. Bisson, and F. Salvadore. Multi-kepler GPU vs. multi-intel MIC for spin systems simulations. *Computer Physics Communications*, 185(10):2495 – 2503, 2014.

[7] M. Bernaschi and F. Salvadore. Multi-kepler gpu vs. multi-intel mic: A two test case performance study. In *High Performance Computing Simulation (HPCS), 2014 International Conference on*, pages 1–8, July 2014.

[8] A. Birkland. Cornell virtual workshop, 2013. https://www.cac.cornell.edu/vw/mic/default.aspx.

[9] F. Broquedis, J. C. Ortega, S. Moreaud, N. Furmento, B. Goglin, G. Mercier, S. Thibault, and R. Namyst. hwloc: A generic framework for managing hardware affinities in HPC applications. In *Parallel, Distributed and Network-Based Processing (PDP), 2010 18th Euromicro International Conference on*, pages 180 –186, Feb. 2010.

[10] S. Browne, J. Dongarra, N. Garner, G. Ho, and P. Mucci. A portable programming interface for performance evaluation on modern processors. *Int. J. High Perform. Comput. Appl.*, 14(3):189–204, August 2000. http://dx.doi.org/10.1177/109434200001400303.

[11] S. Cepeda. Optimization and performance tuning for Intel Xeon Phi coprocessors, part 2: Understanding and using hardware events, 2012. https://software.intel.com/en-us/articles/.

[12] Shannon Cepeda. Optimization and performance tuning for Intel Xeon Phi coprocessors, part 2: Understanding and using hardware events, 2012.

[13] S. Cho and R. Melhem. Corollaries to Amdahl's law for energy. *IEEE Comput. Archit. Lett.*, 7:25–28, Jan. 2008. http://dl.acm.org/citation.cfm?id=1383041.1383084.

[14] J. Choi, M. Mukhan, X. Liu, and R. Vuduc. Algorithmic time, energy, and power on candidate HPC compute building blocks. In *2014 IEEE 28th International Symposiumon Parallel Distributed Processing (IPDPS)*, Arizona, USA, May 2014.

[15] J. W. Choi, D. Bedard, R. Fowler, and R. Vuduc. A roofline model of energy. In *Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pages 661–672, May 2013.

[16] K. Choi, R. Soma, and M. Pedram. Fine-grained dynamic voltage and frequency scaling for precise energy and performance tradeoff based on the ratio of off-chip access to on-chip computation times. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 24(1):18–28, Jan 2005.

[17] Intel Corporation. Developer zone, 2013. https://www.software.intel.com.

[18] Intel Corporation. Intel Xeon Phi coprocessor performance monitoring units., 2013. http://software.intel.com/sites/default/files/forum/278102/intelr-xeon-phitm-pmu-rev1.01.pdf.

[19] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a realistic model of parallel computation. In *Proceedings of the Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPOPP '93, pages 1–12, New York, NY, USA, 1993. ACM. http://doi.acm.org/10.1145/155332.155333.

[20] H. David, E. Gorbatov, U. R. Hanebutte, R. Khannal, and C. Le. RAPL: memory power estimation and capping. In *Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design*, ISLPED'10, pages 189–194, New York, NY, USA, 2010. ACM. http://doi.acm.org/10.1145/1840845.1840883.

[21] E. de Sturler and H. A. van der Vorst. Reducing the effect of global communication in GMRES(m) and CG on parallel distributed memory computers. *Appl. Numer. Math.*, 18(4):441–459, October 1995. http://dx.doi.org/10.1016/0168-9274(95)00079-A.

[22] DOE. Co-design, 2013. http://science.energy.gov/ascr/research/scidac/co-design/.

[23] ExMatEx. CoMD proxy application, 2012. http://www.exmatex.org/comd.html.

[24] T. Ezer. EEMD/HHT, 2015. http://www.ccpo.odu.edu/~tezer/HHT/.

[25] T. Ezer, L. P. Atkinson, W. B. Corlett, and J. L. Blanco. Gulf stream's induced sea level rise and variability along the U.S. mid-Atlantic coast. *Journal of Geophysical Research: Oceans*, 118(2):685–697, 2013. http://dx.doi.org/10.1002/jgrc.20091.

[26] T. Ezer and W. Corlett. Is sea level rise accelerating in the Chesapeake Bay? A demonstration of a novel new approach for analyzing sea level data. *GeophysicalResearch Letters*, 39(19), 2012. http://dx.doi.org/10.1029/2012GL053435.

[27] D. G. Fedorov, R. M. Olson, K. Kitaura, M. S. Gordon, and S. Koseki. A new hierarchical parallelization scheme: Generalized distributed data interface (GDDI), and an application to the fragment molecular orbital method (FMO). *Journal of Computational Chemistry*, 25, Issue 6:872–880, 2004. http://onlinelibrary.wiley.com/doi/10.1002/jcc.20018/full.

[28] B. Goel and S. McKee. A methodology for modeling dynamic and static power consumption for multicore processors. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 273–282, May 2016.

[29] M. S. Gordon, D. G. Fedorov, S. R. Pruitt, and L. V. Slipchenko. Fragmentation methods: A route to accurate calculations on large systems. *Chemical Reviews*, 112(1):632–672, 2012. PMID: 21866983.

[30] M. S. Gordon and M. W. Schmidt. Advances in electronic structure theory: GAMESS a decade later, 2005.

[31] A. Grama, A. Gupta, and V. Kumar. Isoefficiency: measuring the scalability of parallel algorithms and architectures. *IEEE Parallel Distributed Technology: Systems Applications*, 1(3):12–21, Aug 1993.

[32] R. Green. OpenMP thread affinity control, 2012. https://software.intel.com/en-us/articles/openmp-thread-affinity-control.

[33] W. Gropp, L. Olson, and P. Samfass. Modeling MPI communication performance on SMP nodes: Is it time to retire the ping pong test, volume 25-28-September-2016, pages 41–50. Association for Computing Machinery, 9 2016.

[34] G. Hager. Energy vs. performance: Introducing the z-plot, 2016. https://blogs.fau.de/hager/archives/tag/energy.

[35] G. Hager, J. Treibig, J. Habich, and G. Wellein. Exploring performance and power properties of modern multicore chips via simple machine models. *CoRR*, abs/1208.2908, 2012. http://arxiv.org/abs/1208.2908.

[36] R. Hayashi and S. Horiguchi. Domain decomposition scheme for parallel molecular dynamics simulation. In *High Performance Computing on the Information Superhighway, 1997. HPC Asia '97*, pages 595–600, Apr 1997.

[37] A. Heinecke, K. Vaidyanathan, M. Smelyanskiy, A. Kobotov, R. Dubtsov, G. Henry, A.G. Shet, G. Chrysos, and P. Dubey. Design and implementation of the Linpack benchmark for single and multi-node systems based on Intel Xeon Phi coprocessor. In *2013 IEEE 27th International Symposium on Parallel Distributed Processing(IPDPS)*, pages 126–137, May 2013.

[38] J. Hofmann and D. Fey. An ECM-based energy-efficiency optimization approach for bandwidth-limited streaming kernels on recent Intel Xeon processors. In *Proceedings of the 4th International Workshop on Energy Efficient Supercomputing*, E2SC '16, pages 31–38, Piscataway, NJ, USA, 2016. IEEE Press. https://doi.org/10.1109/E2SC.2016.16.

[39] M. Hohnerbach, A. E. Ismail, and P. Bientinesi. The vectorization of the tersoff multi-body potential: An exercise in performance portability. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '16, pages 7:1–7:13, Piscataway, NJ, USA, 2016. IEEE Press. http://dl.acm.org.proxy.lib.odu.edu/citation.cfm?id=3014904.3014914.

[40] N. Huang, Z. Shen, S. Long, M. Wu, H. Shih, Q. Zheng, N. Yen, Chi C. Tung, and H. Liu. The empirical mode decomposition and the Hilbert spectrum for nonlinear and non-stationary time series analysis. *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 454(1971):903–995, 1998.

[41] ICL:UT. Performance application programming interface PAPI, 2015. http://icl.cs.utk.edu/papi/.

[42] A. Ilic, F. Pratas, and L. Sousa. Beyond the roofline: Cache-aware power and energy efficiency modeling for multi-cores. *IEEE Transactions on Computers*, 66(1):52–58, Jan 2017.

[43] Intel. Intel Xeon Phi coprocessor: Datasheet, 2015. http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-coprocessordatasheet.html.

[44] Intel Corporation. Intel C++ Compiler 12.1 user and reference guides, 2013.

[45] Intel Corporation. Intel Xeon Phi coprocessor system software developers guide, 2013. http://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/xeon-phi-coprocessor-systemsoftware-developers-guide.pdf.

[46] Intel Forums. how to measure energy consumption on the coprocessor?, 2013. https://software.intel.com/en-us/forums/topic/488782?language=en.

[47] H. Jordan, P. Thoman, J. Durillo, S. Pellegrini, P. Gschwandtner, T. Fahringer, and H. Moritsch. A multi-objective auto-tuning framework for parallel codes. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 10:1–10:12, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press. http://dl.acm.org/citation.cfm?id=2388996.2389010.

[48] S. Kamil, C. Chan, L. Oliker, J. Shalf, and S. Williams. An auto-tuning framework for parallel multicore stencil computations. In *2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS)*, pages 1–12, April 2010.

[49] M. Kee, H. Lim, G. Park, and S. Cho. An analytical model based on performance demand of workload for energy-efficient heterogeneous multicore systems. *Journal of Parallel and Distributed Computing*, 100:172 – 180, 2017.

[50] T. Kidd. Intel Xeon Phi coprocessor power management configuration: Using the micsmc command-line interface, 2014. https://software.intel.com/en-us/blogs/2014/01/31/intel-xeon-phi-coprocessor-powermanagement-\configuration-using-the-micsmc-command.

[51] D. Kim. Introduction to EMD (Empirical Mode Decomposition) with application to a scientific data, 2006. http://dasan.sejong.ac.kr/˜dhkim/main/research/talks/EMDintroSeminar.pdf.

[52] R. Krishnaiyer, E. Kultursay, P. Chawla, S. Preis, A. Zvezdin, and H. Saito. Compiler based data prefetching and streaming non-temporal store generation for the Intel Xeon Phi coprocessor. In *2013 IEEE 27th International Symposium on Parallel Distributed Processing (IPDPS)*, pages 1575–1586, May 2013.

[53] D. Kusnezov, S. Binkley, B. Harrod, and B. Meisner. DOE exascale initiative, 2013. http://www.industry-academia.org/download/20130913-SEAB-DOE-Exascale-Initiative.pdf.

[54] C. Lai, Z. Hao, M. Huang, X. Shi, and H. You. Comparison of parallel programming models on intel mic computer cluster. In *Parallel Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International*, pages 925–932, May 2014.

[55] J. Laros. Sandia national laboratories high performance computing power application programming interface (API) specification, 2016. http://powerapi.sandia.gov/.

[56] Averill Law. *Simulation Modeling and Analysis (McGraw-Hill Series in Industrial Engineering and Management)*. McGraw-Hill Science/Engineering/Math, 2006.

[57] G. Lawson. Waterfall plot renderer, 2015. https://dl. dropboxusercontent.com/u/21348775/WebBuild/WebBuild.html.

[58] G. Lawson, M. Sosonkina, T. Ezer, and Y. Shen. Applicability of the empirical mode decomposition for power traces of large-scale applications. The International Journal of High Performance Computing Applications, 2017. Accepted June 28, 2017, to appear.

[59] G. Lawson, M. Sosonkina, T. Ezer, and Y. Shen. Applying EMD/HHT analysis to power traces of applications executed on systems with Intel Xeon Phi. The International Journal of High Performance Computing Applications, 2017. Accepted July 8, 2017, to appear.

[60] G. Lawson, M. Sosonkina, T. Ezer, and Y. Shen. Empirical mode decomposition for modeling of parallel applications on Intel Xeon Phi processors. In *Proceedings of the 2nd International Workshop on Theoretical Approaches to Performance Evaluation, Modeling and Simulation*, TAPEMS ’17, 2017.

[61] G. Lawson, M. Sosonkina, and Yuzhong S. Energy evaluation for applications with different thread affinities on the Intel Xeon Phi. In *Computer Architecture and High Performance Computing Workshop (SBAC-PADW), 2014 International Symposium on*, Oct 2014.

[62] G. Lawson, M. Sosonkina, and Y. Shen. Performance and energy evaluation of CoMD on Intel Xeon Phi co-processors. In *Proceedings of the 1st International Workshop on Hardware-Software Co-Design for High Performance Computing*, Co-HPC ’14, Piscataway, NJ, USA, 2014. IEEE Press. http://dx.doi.org/10.1109/CoHPC.2014.12.

[63] G. Lawson, M. Sosonkina, and Y. Shen. Changing CPU frequency in CoMD proxy application offloaded to Intel Xeon Phi co-processors. *Procedia Computer Science*, 51(0):100 – 109, 2015. International Conference On Computational Science, ICCS 2015 Computational Science at the Gates of Nature.

[64] G. Lawson, M. Sosonkina, and Y. Shen. Towards modeling energy consumption of Xeon Phi. *CoRR*, abs/1505.06539, 2015. http://arxiv.org/abs/1505.06539.

[65] G. Lawson, V. Sundriyal, M. Sosonkina, and Y. Shen. Experimentation procedure for offloaded mini-apps executed on cluster architectures with Xeon Phi accelerators, 2015. http://arxiv.org/abs/1509.02135.

[66] G. Lawson, V. Sundriyal, M. Sosonkina, and Y. Shen. Modeling performance and energy for applications offloaded to Intel Xeon Phi. In *Proceedings of the 2Nd International Workshop on Hardware-Software Co-Design for High Performance Computing*, Co-HPC ’15, pages 7:1–7:8, New York, NY, USA, 2015. ACM. http://doi.acm.org/10.1145/2834899.2834903.

[67] G. Lawson, V. Sundriyal, M. Sosonkina, and Y. Shen. Runtime power limiting of parallel applications on Intel Xeon Phi processors. In *Proceedings of the 4th International Workshop on Energy Efficient Supercomputing*, E2SC ’16, pages 39–45, Piscataway, NJ, USA, 2016. IEEE Press. https://doi.org/10.1109/E2SC.2016.9.

[68] B. Li, H. Chang, S. L. Song, C. Su, T. Meyer, J. Mooring, and K. Cameron. The power-performance tradeoffs of the Intel Xeon Phi on HPC applications, 2014. http://scape.cs.vt.edu/wp-content/uploads/2014/06/lspp14-Li.pdf.

[69] X. Liu, S. Peng, C. Yang, C. Wu, H. Wang, Q. Cheng, W. Zhu, and J. Wang. mamber: Accelerating explicit solvent molecular dynamic with Intel Xeon Phi many-integrated core coprocessors. In *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on*, pages 729–732, May 2015.

[70] M. Graham Lopez, Jeffrey Young, Jeremy S. Meredith, Philip C. Roth, Mitchel Horton, and Jeffrey S. Vetter. Examining recent many-core architectures and programming models using shoc. In *Proceedings of the 6th International Workshop on Performance Modeling, Benchmarking, and Simulation of High Performance Computing Systems*, PMBS '15, pages 3:1–3:12, New York, NY, USA, 2015. ACM. http://doi.acm.org/10.1145/2832087.2832090.

[71] U. Lopez-Novoa, A. Mendiburu, and J. Miguel-Alonso. A survey of performance modeling and simulation techniques for accelerator-based computing. *IEEE Transactions on Parallel and Distributed Systems*, 26(1):272–281, Jan 2015.

[72] M. Luo, M. Li, A. Venkatesh, X. Lu, and D. Panda. Upc on mic: Early experiences with native and symmetric modes. *Proceedings of the 7th International Conference on PGAS Programming Models*, 2013.

[73] Matlab. Align signals using cross-correlation, 2017. `https://www.mathworks.com/help/signal/ug/align-signals-using-cross-correlation.html`.

[74] M. Meswani, L. Carrington, D. Unat, A. Snavely, S. Baden, and S. Poole. Modeling and predicting performance of high performance computing applications on hardware accelerators. In *2012 IEEE 26th International Parallel and Distributed ProcessingSymposium Workshops PhD Forum*, pages 1828–1837, May 2012.

[75] NASA. Problem sizes and parameters in NAS parallel benchmarks, 2012. https://www.nas.nasa.gov/publications/npb_problem_sizes.html.

[76] NASA. NAS parallel benchmarks, 2013. http://www.nas.nasa.gov/publications/npb.html.

[77] Chris J. Newburn, Serguei Dmitriev, Ravi Narayanaswamy, John Wiegert, Ravi Murty, Francisco Chinchilla, Rajiv Deodhar, and Russ McGuire. Offload compiler runtime for the Intel Xeon Phi™ coprocessor. In *Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing Workshops and PhD Forum*, IPDPSW '13, pages 1213–1225, Washington, DC, USA, 2013. IEEE Computer Society. http://dx.doi.org/10.1109/IPDPSW.2013.251.

[78] J. Park, G. Bikshandi, K. Vaidyanathan, P. Tang, P. Dubey, and D. Kim. Tera-scale 1d FFT with low-communication algorithm and Intel Xeon Phi coprocessors. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, pages 34:1–34:12, New York, NY, USA, 2013. ACM.

http://doi.acm.org/10.1145/2503210.2503242.

[79] Open MPI Project. Portable hardware locality (hwloc), 2016. https://www.open-mpi.org/projects/hwloc/.

[80] M. Quinn. *Parallel Programming in C with MPI and OpenMP*. McGraw-Hill Education Group, 2003.

[81] S. Ramos and T. Hoefler. Modeling communication in cache-coherent smp systems: A case-study with xeon phi. In *Proceedings of the 22Nd International Symposium on High-performance Parallel and Distributed Computing*, HPDC '13, pages 97–108, New York, NY, USA, 2013. ACM. http://doi.acm.org/10.1145/2462902.2462916.

[82] J. Reinders and J. Jeffers. *High Performance Parallelism Pearls Volume Two*. MK Publishers, 2015. http://lotsofcores.com/.

[83] S. Saini, H. Jin, D. Jespersen, S. Cheung, J. Djomehri, J. Chang, and R. Hood. Early multi-node performance evaluation of a knights corner (knc) based nasa supercomputer. In *Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2015 IEEE International*, pages 57–67, May 2015.

[84] F. Sainz, J. Belln, V. Beltran, and J. Labarta. Collective offload for heterogeneous clusters. In *2015 IEEE 22nd International Conference on High Performance Computing (HiPC)*, pages 376–385, Dec 2015.

[85] E. Saule, K. Kaya, and U. C¸ ataly ¨ urek. ¨ *Performance Evaluation of Sparse Matrix Multiplication Kernels on Intel Xeon Phi*, pages 559–570. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014. http://dx.doi.org/10.1007/978-3-642-55224-3 52.

[86] M. W. Schmidt, K. K. Baldridge, J. A. Boatz, S. T. Elbert, M. S. Gordon, J. H. Jensen, S. Koseki, N. Matsunaga, K. A. Nguyen, S. Su, T. L. Windus, M. Dupuis, and Jr. J. A. Montgomery. General atomic and molecular electronic structure system. *J. Comput. Chem.*, 14:1347–1363, Nov. 1993. http://portal.acm.org/citation.cfm?id=163483.163497.

[87] Y. S. Shao and D. Brooks. Energy characterization and instruction-level energy model of Intel's Xeon Phi processor, 2013. http://www.eecs.harvard.edu/˜shao/papers/shao2013-islped.pdf.

[88] H. Shoukourian, T. Wilde1, A. Auweterl, and A. Bode. Predicting the energy and power consumption of strong and weak scaling HPC applications. *Supercomputing Frontiers and Innovations*, (2), 2014. http://superfri.org/superfri/article/view/9/105.

[89] A. Sodani, R. Gramunt, J. Corbal, H. S. Kim, K. Vinod, S. Chinthamani, S. Hutsell, R. Agarwal, and Y. C. Liu. Knights landing: Second-generation Intel Xeon Phi product. *IEEE Micro*, 36(2):34–46, Mar 2016.

[90] StackOverflow. Fit bimodal normal distribution to a vector of values, not its histogram, 2015. https://stackoverflow.com/questions/30374594/fit-bimodal-normal-distribution-to-a-vector-of-valuesnot-its-histogram.

[91] E. de Sturler. A performance model for krylov subspace methods on mesh-based parallel computers. *Parallel Comput.*, 22(1):57–74, January 1996. http://dx.doi.org/10.1016/0167-8191(95)00057-7.

[92] K. Suksomboon, N. Matsumoto, M. Fukushima, S. Okamoto, and M. Hayashi. Towards performance prediction of multicore software routers. *International Journal of Network Management*, 27(2):e1966–n/a, 2017. http://dx.doi.org/10.1002/nem.1966.

[93] V. Sundriyal, A. Gaenko, M. Sosonkina, and Z. Zhang. Energy saving strategies for parallel applications with point-to-point communication phases. *Journal of Parallel and Distributed Computing*, 73(8):1157–1169, August 2013.

[94] V. Sundriyal and M. Sosonkina. Initial investigation of a scheme to use instantaneous CPU power consumption for energy savings format. In *Proceedings of the 1st International Workshop on Energy Efficient Supercomputing*, New York, NY, USA, 2013. ACM. http://doi.acm.org/10.1145/2536430.2536433.

[95] V. Sundriyal and M. Sosonkina. Analytical modeling of the CPU frequency to minimize energy consumption in parallel applications. *Submitted for publication to: Elsevier*, 2015.

[96] V. Sundriyal and M. Sosonkina. Joint frequency scaling of processor and dram. *The Journal of Supercomputing*, 72(4):1549–1569, 2016. http://dx.doi.org/10.1007/s11227-016-1680-4.

[97] V. Sundriyal, M. Sosonkina, and Z. Zhang. Automatic runtime frequency- scaling system for energy savings in parallel applications. *The Journal of Supercomputing*, 68(2):777–797, 2014. http://dx.doi.org/10.1007/s11227-013-1062-0.

[98] G. Teodoro, T. Kurc, J. Kong, L. Cooper, and J. Saltz. Comparative performance analysis of Intel Xeon Phi, GPU, and CPU: A case study from microscopy image analysis. In *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*, pages 1063–1072, May 2014.

[99] Top500. Top 500 supercomputer sites, 2013. http://www.top500.org/.

[100] Unity3D, 2015. https://unity3d.com/.

[101] P. Wang. Ready to run applications from multicore platform onto Intel Xeon Phi coprocessor, 2013. https://software.intel.com/en-us/blogs/2013/01/08/ready-to-run-applications-from-multicore-platformonto-intel-xeon-phi-coprocessor.

[102] S. Wang, B. Luo, W. Shi, and D. Tiwari. Application configuration selection for energy-efficient execution on multicore systems. *Journal of Parallel and Distributed Computing*, 87:43 – 54, 2016.

[103] V. Weaver. Reading RAPL energy measurements from linux, 2011. http://web.eece.maine.edu/~vweaver/projects/rapl/.

[104] Wikipedia. Quadratic function, 2017. https://en.wikipedia.org/wiki/Quadratic_function.

[105] Wikipedia. Sunway taihulight, 2017. https://en.wikipedia.org/wiki/Sunway_TaihuLight.

[106] Wikipedia. Titan (supercomputer), 2017. https://en.wikipedia.org/wiki/Titan_(supercomputer).

[107] S. Williams, A. Waterman, and D. Patterson. Roofline: An insightful visual performance model for multicore architectures. *Commun. ACM*, 52(4):65–76, April 2009. http://doi.acm.org/10.1145/1498765.1498785.

[108] J. Wood, Z. Zong, Q. Gu, and R. Ge. Energy and power characterization of parallel programs running on Intel Xeon Phi. In *2014 43rd International Conference on Parallel Processing Workshops*, pages 265–272, Sept 2014.

[109] Z. Wu and N. Huang. Ensemble empirical mode decomposition: A noise-assisted data analysis method. *Advances in Adaptive Data Analysis*, 01(01):1–41, 2009. http://www.worldscientific.com/doi/abs/10.1142/S1793536909000047.

[110] Z. Zhang and J. M. Chang. A cool scheduler for multi-core systems exploiting program phases. *IEEE Trans. Comput.*, 63(5):1061–1073, May 2014.

[111] Victoria Zhislina. Why has CPU frequency ceased to grow?, 2014. https://software.intel.com/en-us/blogs/2014/02/19/why-has-cpu-frequency-ceased-to-grow.

Gary D Lawson Jr

1625 Skyline Drive
Norfolk, VA 23518
Mobile: (757) 292-3528
Email: gary.lawson.hpc@gmail.com

Summary of Qualifications

Extensive experience with modeling power/energy and performance of high-performance computing systems, heterogeneous and homogeneous computing platforms. Extensive experience with Intel Xeon Phi accelerators (processor and coprocessor), and some experience with NVIDIA GPU. Proficient with MPI, MPI-3 Shared Memory, and OpenMP in C/C++ and Fortran. Self-motivated professional, capable of working independently or collaboratively with a team, and driven to aspire to management and leadership positions.

Professional Experience

NASA, Poquoson, VA
Contractor, Aug 2016 – Oct 2017

- Designed mini-app to test shared memory paradigms (MPI-3 Shared Memory and OpenMP) on routine(s) of interest from VULCAN
- Improved performance using loop-blocking and shared memory parallelism in the mini-app with 75% efficiency (15x speedup using 20 processors)
- Successfully implemented MPI-3 in VULCAN on routine(s) of interest
- Delegated work among 2 under-graduate students who aided with the mini-app portion of the project.

Old Dominion University, Norfolk, VA
Research Assistant, Sept 2013 – Present

- Time-series Analysis using Empirical Mode Decomposition of power measurements
- Power and performance modeling of various applications (benchmark, mini-app, and real-world) across several computing platforms (CPU, Accelerator, and CPU + Accelerator configurations)
- Energy prediction using well-known power and performance models
- Languages: C/C++/C#, Fortran, Matlab, Python, Bash
- Libraries: MPI, OpenMP, CUDA, Sandia PowerAPI, micmgmt API (KNC)

Education

Old Dominion University, Norfolk, VA
2010 – Present

- Doctor of Philosophy, Modeling Simulation and Visualization Engineering
- Master of Science, Modeling Simulation and Visualization Engineering
- Bachelor of Science, Electrical Engineering