E-AMOM: AN ENERGY-AWARE MODELING AND OPTIMIZATION METHODOLOGY FOR SCIENTIFIC APPLICATIONS ON MULTICORE SYSTEMS

A Dissertation

by

CHARLES WESLEY LIVELY III

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 2012

Major Subject: Computer Engineering

E-AMOM: An Energy-Aware Modeling and Optimization Methodology for Scientific Applications on Multicore Systems

E-AMOM: AN ENERGY-AWARE MODELING AND OPTIMIZATION METHODOLOGY FOR SCIENTIFIC APPLICATIONS ON MULTICORE SYSTEMS

A Dissertation

by

CHARLES WESLEY LIVELY III

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Approved by:

Chair of Committee,	Valerie Elaine Taylor
Committee Members,	Karen Butler-Purry
	Eun Jung Kim
	Tiffani Williams
Head of Department,	Duncan Walker

May 2012

Major Subject: Computer Engineering

ABSTRACT

E-AMOM: An Energy-Aware Modeling and Optimization Methodology for Scientific Applications on Multicore Systems. (May 2012)

Charles Wesley Lively III,

B.S.E., Mercer University;

M.S., Texas A&M University

Chair of Advisory Committee: Dr. Valerie Elaine Taylor

Power consumption is an important constraint in achieving efficient execution on High Performance Computing Multicore Systems. As the number of cores available on a chip continues to increase, the importance of power consumption will continue to grow. In order to achieve improved performance on multicore systems scientific applications must make use of efficient methods for reducing power consumption and must further be refined to achieve reduced execution time.

In this dissertation, we introduce a performance modeling framework, E-AMOM, to enable improved execution of scientific applications on parallel multicore systems with regards to a limited power budget. We develop models for each application based upon performance hardware counters. Our models utilize different performance counters for each application and for each performance component (runtime, system power consumption, CPU power consumption, and memory power consumption) that are selected via our performance-tuned principal component analysis method. Models developed through E-AMOM provide insight into the performance characteristics of

each application that affect performance for each component on a parallel multicore system. Our models are more than 92% accurate across both Hybrid (MPI/OpenMP) and MPI implementations for six scientific applications.

E-AMOM includes an optimization component that utilizes our models to employ run-time Dynamic Voltage and Frequency Scaling (DVFS) and Dynamic Concurrency Throttling to reduce power consumption of the scientific applications. Further, we optimize our applications based upon insights provided by the performance models to reduce runtime of the applications. Our methods and techniques are able to save up to 18% in energy consumption for Hybrid (MPI/OpenMP) and MPI scientific applications and reduce the runtime of the applications up to 11% on parallel multicore systems.

DEDICATION

This dissertation is dedicated to my loving and supportive parents, Charles W. Lively Jr. and Irene S. Lively.

ACKNOWLEDGEMENTS

Graduate school has been an enriching journey that has helped to shape my life and thought process. For this, I would first like to give thanks to my Lord and Savior Jesus Christ for guiding me through out this journey and in life. My mother, Irene, has been an endless source of encouragement and support since birth and I could never repay her for always encouraging my intellectual interests. My family and friends have been a great support system over the years and so I would like to thank them for always offering kind words of encouragement and support (Charles Lively Jr., Vidal Lively, Charles Beverley Jr., Courtney Carey, Jesse Dukes, Jacqueline Hodge, Carla Marsh, and Antoinette Davis)

I have to thank my academic family for support throughout this time. My advisor and mentor, Dr. Valerie Taylor, has taught me what it truly takes to be an excellent researcher through constant encouragement, hard work, and "refinement". I would also like to give thanks to my second advisor, Dr. Xingfu Wu, for always providing encouraging feedback and support. Special thanks are also in order for my past research group members Dr. Ayodeji Coker and Dr. Sameh Sharkawi.

NOMENCLATURE

MPI	Message Passing Interface
HPC	High Performance Computing
DVFC	Dynamic Voltage and Frequency Scaling
DCT	Dynamic Concurrency Throttling

TABLE OF CONTENTS

Page

ABSTRACT iii

DEDICATION v

ACKNOWLEDGEMENTS vi

NOMENCLATURE vii

TABLE OF CONTENTS viii

LIST OF FIGURES x

LIST OF TABLES xv

1. INTRODUCTION 1

    1.1 Research Challenges on Multicore Systems 8
    1.2 Modeling Infrastructure 11
    1.3 Related Work 16

2. PROPOSED PERFORMANCE MODELING SCHEME 27

    2.1 Energy-Aware Modeling and Optimization Methodology (E-AMOM) 27
    2.2 Performance-Tuned Principal Component Analysis Method 30
    2.3 Application Optimization Methods 34
    2.4 Modeling Approaches Leveraged 37

3. PERFORMANCE-POWER TRADE-OFFS OF MPI AND HYBRID APPLICATIONS 50

    3.1 Parallel Multicore Systems 50
    3.2 Experimental Environment 52
    3.3 Experimental Results 53
    3.4 Summary 68

Page

4. POWER-AWARE PERFORMANCE MODELS OF SCIENTIFIC APPLICATIONS..... 70

4.1 Performance-Tuned Principal Component Analysis Methodology..... 70
4.2 HPC Applications 81
4.3 Experimental Results 85
4.4 Summary 149

5. OPTIMIZATION OF HYBRID AND MPI SCIENTIFIC APPLICATIONS.... 152

5.1 Software-Based Power Reduction Methods 152
5.2 Optimization Methodology for Application Kernels..... 153
5.3 Loop Optimizations 160
5.4 Experimental Results 161
5.5 Summary 184

6. SUMMARY AND FUTURE WORK 185

6.1 Summary 185
6.2 Future Work 186

REFERENCES 189

VITA 199

LIST OF FIGURES

FIGURE		Page
1	Hex-Core AMD Opteron [60]	2
2	Predicted Power Requirements for Exascale Systems [38]	4
3	Multiple Metrics Modeling Infrastructure [51]	12
4	Prophesy Framework [63]	13
5	PAPI Framework [56]	14
6	PowerPack Framework	15
7	E-AMOM Schema	29
8	E-AMOM Integration into MuMMI	30
9	Initial Hierarchical Multicore Modeling Framework	38
10	Modeling Assertions Framework	41
11	Matrix-Matrix Multiply MA Profile	42
12	Matrix-Matrix Multiply Model	43
13	Matrix-Matrix Sensitivity Analysis	44
14	Power for MPI BT Executed on 1 Node	55
15	Power for Hybrid BT on 1 Node	56
16	Spearman and Pearson Correlation Comparison	72
17	PCA Analysis Example	78
18	Average Error of BT-MZ Hybrid	87
19	Scatterplot of BT-MZ Hybrid for Runtime	88

Page

20	Scatterplot of BT-MZ Hybrid for System Power Consumption	89
21	Scatterplot of BT-MZ Hybrid for CPU Power Consumption	90
22	Scatterplot of BT-MZ Hybrid for Memory Power Consumption	91
23	Average Error of BT-MZ MPI	92
24	Scatterplot of BT-MZ MPI for Runtime	93
25	Scatterplot of BT-MZ MPI for System Power Consumption	94
26	Scatterplot of BT-MZ MPI for CPU Power Consumption	95
27	Scatterplot of BT-MZ MPI for Memory Power Consumption	96
28	Average Error of SP-MZ Hybrid	97
29	Scatterplot of SP-MZ Hybrid for Runtime	99
30	Scatterplot of SP-MZ Hybrid for System Power Consumption	100
31	Scatterplot of SP-MZ Hybrid for CPU Power Consumption	101
32	Scatterplot of SP-MZ Hybrid for Memory Power Consumption	102
33	Average Error of SP-MZ MPI	103
34	Scatterplot of SP-MZ MPI for Runtime	104
35	Scatterplot of SP-MZ MPI for System Power Consumption	105
36	Scatterplot of SP-MZ MPI for CPU Power Consumption	106
37	Scatterplot of SP-MZ MPI for Memory Power Consumption	107
38	Average Error of LU-MZ Hybrid	109
39	Scatterplot of LU-MZ Hybrid for Runtime	110
40	Scatterplot of LU-MZ Hybrid for System Power Consumption	111

Page

41 Scatterplot of LU-MZ Hybrid for CPU Power Consumption 112

42 Scatterplot of LU-MZ Hybrid for Memory Power Consumption 113

43 Average Error of LU-MZ MPI 115

44 Scatterplot of LU-MZ MPI for Runtime 116

45 Scatterplot of LU-MZ MPI for System Power Consumption 117

46 Scatterplot of LU-MZ MPI for CPU Power Consumption 117

47 Scatterplot of LU-MZ MPI for Memory Power Consumption 118

48 Average Error of GTC Hybrid 120

49 Scatterplot of GTC Hybrid for Runtime 121

50 Scatterplot of GTC Hybrid for System Power Consumption 122

51 Scatterplot of GTC Hybrid for CPU Power Consumption 123

52 Scatterplot of GTC Hybrid for Memory Power Consumption 124

53 Average Error of GTC MPI 125

54 Scatterplot of GTC MPI for Runtime 126

55 Scatterplot of GTC MPI for System Power Consumption 127

56 Scatterplot of GTC Hybrid for CPU Power Consumption 128

57 Scatterplot of GTC Hybrid for Memory Power Consumption 129

58 Average Error of PMLB Hybrid 130

59 Scatterplot of PMLB Hybrid for Runtime 131

60 Scatterplot of PMLB Hybrid for System Power Consumption 132

61 Scatterplot of PMLB Hybrid for CPU Power Consumption 133

Page

62 Scatterplot of PMLB Hybrid for Memory Power Consumption 134

63 Average Error of PMLB MPI 135

64 Scatterplot of PMLB MPI for Runtime 136

65 Scatterplot of PMLB MPI for System Power Consumption..... 137

66 Scatterplot of PMLB MPI for CPU Power Consumption..... 138

67 Scatterplot of PMLB MPI for Memory Power Consumption..... 139

68 Average Error of Parallel EqDyna Hybrid..... 140

69 Scatterplot of EqDyna Hybrid for Runtime 141

70 Scatterplot of EqDyna Hybrid for System Power Consumption 142

71 Scatterplot of EqDyna Hybrid for CPU Power Consumption 143

72 Scatterplot of EqDyna Hybrid for Memory Power Consumption 144

73 Average Error of EqDyna MPI 145

74 Scatterplot of EqDyna MPI for Runtime 146

75 Scatterplot of EqDyna MPI for System Power Consumption 147

76 Scatterplot of EqDyna MPI for CPU Power Consumption 148

77 Scatterplot of EqDyna MPI for Memory Power Consumption 149

78 Average Error of All Hybrid Applications 150

79 Average Error of All MPI Applications..... 151

80 Overview of Optimization Scheme..... 156

81 Example Application Control Flow 157

82 Per Kernel Predictions for Applications 159

Page

83 Applying Optimizations to an Application 162

LIST OF TABLES

TABLE		Page
1	Top 500 Supercomputers in the World (Top 500 List) [64]	3
2	Hardware Performance Counters	32
3	Kernel Coupling Values for M-M Kernel	47
4	System Configuration of Dori	53
5	Overview of HPC Applications	54
6	Runtime and Energy Comparison for OpenMP and MPI BT on 1 Node	57
7	Energy and Runtime Comparison of MPI and Hybrid BT	58
8	Energy and Runtime Comparison of MPI and Hybrid PMLB Application	60
9	Energy and Runtime Comparison of MPI and Hybrid GTC Application	62
10	MPI and Hybrid BT on 4x4 (16 Cores) Using Frequency Scaling	64
11	GTC Power Profiling on 4x4 (16 Cores) Using Frequency Scaling	66
12	Function Comparison of GTC Using Frequency Scaling	67
13	Reduced Performance Counters and Correlation Value	73
14	Reduced Performance Counters and Regression Coefficients	75
15	Reduced Performance Counters and Regression Coefficients-Step 5	76
16	Final Multivariate Regression Model	79
17	Overview of HPC Applications	82
18	Regression Coefficients for BT-MZ Hybrid	86
19	Regression Coefficients for BT-MZ MPI	92

Page

20 Regression Coefficients for SP-MZ Hybrid 97

21 Regression Coefficients for SP-MZ MPI..... 103

22 Regression Coefficients for LU-MZ Hybrid..... 108

23 Regression Coefficients for LU-MZ MPI..... 114

24 Regression Coefficients for GTC Hybrid 119

25 Regression Coefficients for GTC MPI 125

26 Regression Coefficients for PMLB Hybrid 130

27 Regression Coefficients for PMLB MPI..... 135

28 Regression Coefficients for Parallel EqDyna Hybrid..... 140

29 Regression Coefficients for Parallel EqDyna MPI 145

30 Performance of Hybrid BT-MZ Application and Optimization (Class C) 163

31 Performance of Hybrid BT-MZ Application and Optimization (Class D) 164

32 Performance of MPI BT-MZ Application and Optimization (Class C) 165

33 Performance of MPI BT-MZ Application and Optimization (Class D) 166

34 Performance of Hybrid SP-MZ Application and Optimization (Class C). 167

35 Performance of Hybrid SP-MZ Application and Optimization (Class D). 168

36 Performance of MPI SP-MZ Application and Optimization (Class C) 169

37 Performance of MPI SP-MZ Application and Optimization (Class D)..... 170

38 Performance of Hybrid LU-MZ Application and Optimization (Class C) 171

39 Performance of Hybrid LU-MZ Application and Optimization (Class D) 172

40 Performance of Hybrid GTC Application and Optimization (50ppc) 173

Page

41 Performance of Hybrid GTC Application and Optimization (100ppc) 174

42 Performance of MPI GTC Application and Optimization (50ppc) 176

43 Performance of MPI GTC Application and Optimization (100ppc) 177

44 Performance of Hybrid PMLB Application and Optimization (128) 178

45 Performance of Hybrid PMLB Application and Optimization (256) 179

46 Performance of MPI PMLB Application and Optimization (128) 180

47 Performance of MPI PMLB Application and Optimization (256) 180

48 Performance of Hybrid EqDyna Application and Optimization (50ppc) .. 182

49 Performance of MPI EqDyna Application and Optimization (50ppc) 183

1. INTRODUCTION

In high performance computing, the current trend makes use of chip multiprocessors (multicore processors) for computing systems. The incorporation of uniprocessors in computing has reached both performance and physical limitations. For example, the processing speeds for uniprocessors are no longer able to scale with Moore's law [34]. Therefore, the use of multicore processors has been sought as an alternative avenue to maintain the gains in performance that have occurred previously in the computing field. Recently, interconnect technologies have posed limits on the capabilities of systems to continue the scaling of Moore's law.

It is expected that the number of cores available on a chip will continue to increase and the hierarchical nature of parallel systems will also continue to increase. As we move toward increased performance of high-performance parallel computing systems, it is expected that the complexity of the organization will increase as well as the power consumption [34][37]. In this work, we propose a methodology, called E-AMOM, to model and analyze the performance characteristics of scientific applications on high-end parallel systems with multicore processors.

Within the past decade, several chip manufacturers have introduced multicore processors. Since their introduction, multicore chips have been utilized in parallel computing environments by scientific laboratories, data centers, and academic chip, was introduced in 2001 with original clock speeds in the range of 1.1 Ghz to 1.3 Ghz. The

---

This dissertation follows the style of *IEEE Trans. on Parallel and Distributed Systems.*

first dual-core processor, the IBM POWER4 chip, was introduced in 2001 with original clock speeds in the range of 1.1 to 1.3 Ghz [33].

Later, Intel deployed its first dual-core chip, the Pentium D, in 2005 with clock speeds in the range of 3.0 – 3.2 Ghz [17]. In 2007 Intel introduced the world's first quad- core processor, Clovertown [18]. Currently, the use of multicores in computing has become the norm. Hence, there is a greater need to model how large-scale scientific applications perform and scale efficiently on these systems. Specifically, it remains to be understood what components of these emerging systems influence the achievable performance of large-scale scientific applications. Existing multicore compute systems can be configured hierarchically with multiple multicore chips within a node. These systems also utilize various levels of sharing for their memory subsystems.

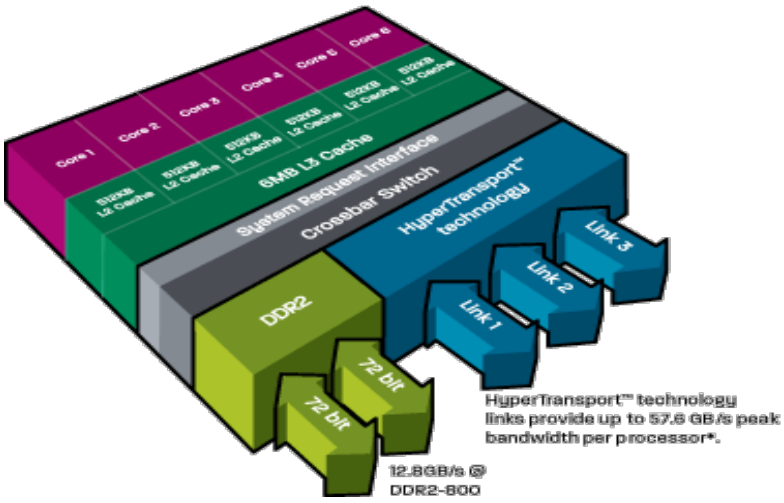

Figure 1. Hex-Core AMD Opteron [60]

The configurations of current parallel compute systems provide an arrangement in hierarchical manner for accessing the memory subsystems. For example, the hex-core AMD Opteron processor, shown in Figure 1 [60], is used in the Jaguar Cray XT5 system, available at Oak Ridge National Laboratory. Each core on the processor has both a 128 KB L1 cache and 512KB L2 cache. A 2MB L3 cache is shared amongst all four cores on the quad-core Opteron. In addition, each node of the XT5 system contains two hex-core chips providing 12 processors per node.

Table 1. Top 500 Supercomputers in the World (Top 500 List) [64]

Rank	**Site**	**Number of Cores**	**Rmax (Tflops)**	**Rpeak (Tflops)**	**Power (KW)**
1	RIKEN Advanced Institute for Computational Science (AICS), Japan	705024	10510.00	11280.38	12659.9
2	National Supercomputing Center in Tianjin, China	186,368	2566.00	4701.00	4040.00
3	DOE/SC/Oak Ridge National Laboratory, USA	224,162	1759.00	2331.00	6950.60
4	National Supercomputing Centre in Shenzhen (NSCS), China	120,640	1271.00	2984.30	2580.00
5	GSIC Center, Tokyo Inst. of Technology Japan	73,278	1192.00	2287.63	1398.61

As parallel computing systems in HPC continue to incorporate more cores onto the system, the amount of power required to run these systems continues to be a major performance bottleneck [22]. Table 1 provides an overview of the power requirements of the top 5 supercomputing systems in the world based on rankings for the Top 500 list [64]. This table illustrates the power requirements that the top systems in the world currently required in order to run large-scale scientific applications.

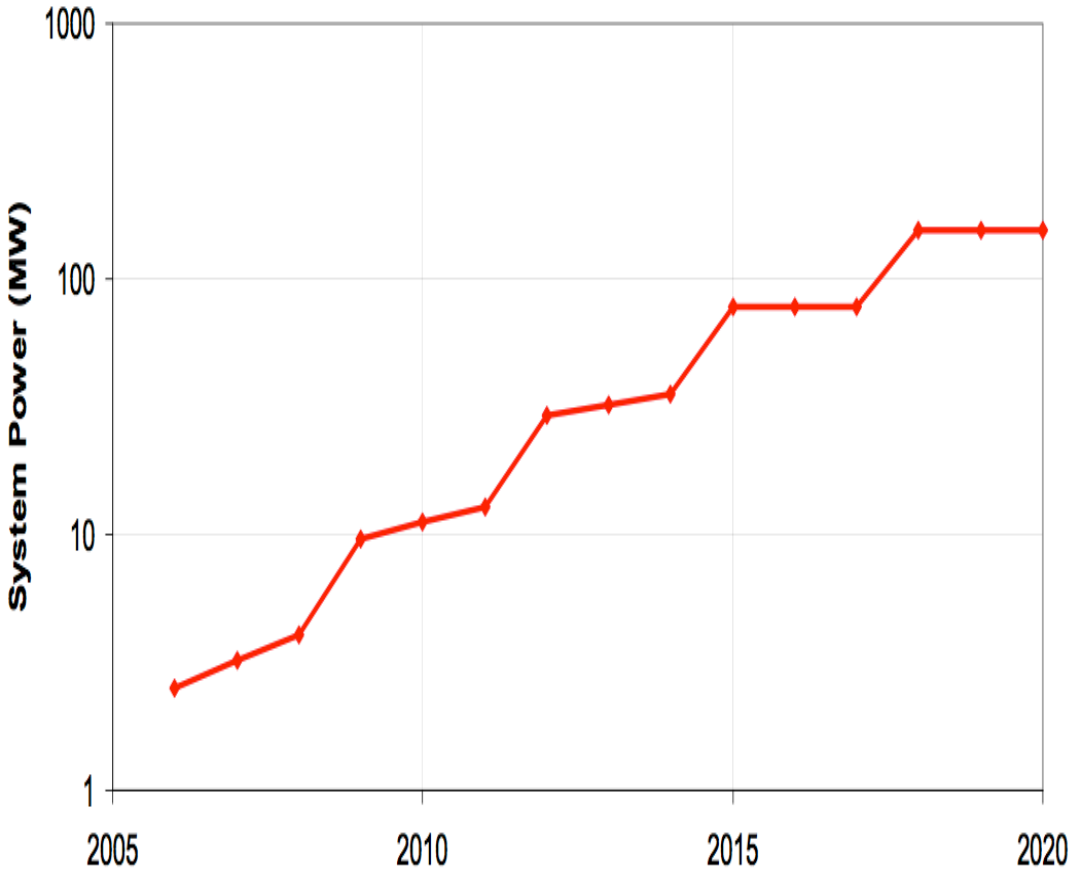

Figure 2. Predicted Power Requirements for Exascale Systems [38]

Figure 2 outlines the predicted power requirements for future systems in high performance computing [38]. Power will be the leading contributor in energy costs as systems continue to grow at the exascale HPC level [34]. Costs are already a primary concern with existing supercomputing centers. For example, the expected upgrade to Oak Ridge National Laboratory's flagship machine will cost more than $96 Million as a multiphase project. Projections indicate that for large-scale systems reducing the power consumption by 1 megawatt will result in $1 million in savings per year [22].

As high performance computing systems become more complex and hierarchical in nature it is important that scientific applications are able to effectively make use of these systems. This dissertation provides a framework for achieving improved execution of an application on a high performance computing system given a limited power budget. Researchers and application developers need appropriate methods to understand how to improve the performance and power consumption of their application. Our modeling framework is useful to HPC users in the following ways:

- E-AMOM can be used to obtain the necessary application performance characteristics to determine application bottlenecks on a given system with regards to execution time and power consumption for the system, CPU, and memory components.
- E-AMOM can be used to improve the performance of the application with regards to applying DVFS and DCT to reduce power consumption and making algorithmic changes to improve power consumption.
- E-AMOM can be used by supercomputer schedulers to provide performance

predictions (about execution time and power requirements) for scheduling methods used with systems with a fixed power budget.

The contributions of this dissertation to the current literature of evaluating performance-power tradeoffs can be summarized in the following points:

1. We present a performance-tuned principal component analysis method for identifying application characteristics that affect performance of the application.
2. We present accurate performance models of Hybrid (MPI/OpenMP) and MPI implementations of scientific applications. Our models are able to accurately predict runtime and power consumption of the system, CPU, and memory components across different number of processors, frequency settings, concurrency settings, and application inputs.
3. Our models are used to determine appropriate frequency and concurrency settings for application kernels to reduce power consumption.
4. E-AMOM is used to optimize Hybrid and MPI scientific applications to improve cache utilization through loop blocking and loop unrolling techniques.
5. Our combined optimization strategy, developed in E-AMOM, is able to reduce energy consumption of Hybrid and MPI scientific applications by as much as 18% on multicore systems for six applications.

The publications resulting from this work are the following:

- Charles Lively, Xingfu Wu, Valerie Taylor, Shirley Moore, Hung-Ching

Chang, Chun-Yi Su and Kirk Cameron, *Power-Aware Predictive Models of Hybrid (MPI/OpenMP) Scientific Applications on Multicore Systems*, International Conference on Energy-Aware High Performance Computing(EnA-HPC2011), September 2011.

- Charles Lively, Xingfu Wu, Valerie Taylor, Shirley Moore, Hung-Ching Chang, and Kirk Cameron, *Energy and Performance Characteristics of Different Parallel Implementations of Scientific Applications on Multicore Systems*, International Journal of High Performance Computing Applications (IJHPCA), Volume 25 Issue 3, August 2011, pp. 342 – 350.
- Charles Lively, Sadaf Alam, Jeffrey Vetter, and Valerie Taylor, *A Methodology for Developing High Fidelity Communications Models for Large-scale Applications on Multicore Systems*, the 20th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 2008), IEEE Computer Society Press, Oct. 29-Nov. 1, 2008, Campo Grande, Mato Grosso do Sol, Brazil.

The remainder of this section identifies the different requirements and challenges with respect to modeling and improving performance of scientific applications for reducing power consumption and provides a discussion about related work. The remainder of this dissertation is organized as follows: Section two provides an overview of E-AMOM and presents some background on the problem. Section three presents preliminary experimental results, which provide the motivation for our modeling and optimization framework. Section four presents the performance models of the Hybrid

and MPI applications and analysis. Section five discusses optimization techniques and presents optimization results of Hybrid and MPI scientific applications. The final section presents the summary and future work that will expand upon this dissertation.

1.1 Research Challenges on Multicore Systems

The importance of the detailed analysis of multicore systems and the applications that run on them is directly related to the continuous sustainability of improved performance as dictated by Moore's law [38]. If computing is to continue making the performance gains experienced during the 1990s, an explicit parallelism is needed in applications and architectures. Therefore, as the number of cores on multicore processors continue to grow there are a number of challenges that affect the performance of large-scale scientific applications are executed on multicore compute systems. These challenges include memory utilization, concurrency and locality, and power and energy utilization.

The issue of performance in scientific computing can be seen as the ability of an application to efficiently utilize a multicore compute system with respect to execution time, power and energy, and utilization of the memory subsystem. These challenges are the driving forces behind discovering new and better technologies [4]. These challenges present obstacles that must be addressed if multicore systems are able to reach their full potential for performance. The obstacles that these challenges present are explained in the following subsections.

1.1.1 Energy and Power Challenge

As the utilization of multicore processors continues to increase the power consumption of these systems becomes a problem in maintaining stability of the system. Power in a multicore compute system consists of the power utilized by the CPUs, main memory, interconnects, and storage. Power utilized by main memory includes the total energy needed to refresh main memory, the number of independent accesses per second, and the data bandwidth needed to move accessed data. Additionally, the power required to move computational data through various interconnect levels, such as on-chip, between chips, within a node, and between nodes, can be in the order of 1-3 pJ [38].

Existing compute systems utilize ten's of thousands of processing cores that requires massive amounts of power [27]. Currently, the fastest supercomputing system in the world requires more than 12MWatts of power in order to achieve 10,510 Tflops, when such systems utilize such massive amounts of power this increases operating costs, and decreases the long-term lifecycle of the compute systems. Improving the power utilization of parallel systems that utilize multicore platforms will reduce overall maintenance costs, system failures, and increase the active time that the system can be used by researchers.

In this section, we discuss two topics that are of high importance for meeting the performance expectations of scientific applications with regards to energy and power consumption: power prediction and performance-power optimization.

1.1.1.1 Power Prediction

Within the field, the ability to predict or accurately estimate the power consumption of scientific applications has posed a great challenge in multicore systems and emerging architectures. Determining the most efficient implementation to use for executing an application can be a cumbersome task that requires comparative analysis of the application's implementation for different datasets. In order, to predict and model the application one needs to understand the application characteristics that will affect performance of the application on the system.

In this dissertation, we address the power prediction challenge in parallel multicore systems modeling scientific applications using E-AMOM. We identify the application characteristics, through performance hardware counters, that affect power consumption of the application. In addition, we determine the similar and different characteristics between MPI and Hybrid implementations of an application that affect performance.

1.1.1.2 Power-Aware Optimization

Within the field, reducing the amount of power and energy consumed by a scientific application on multicore parallel systems poses a tremendous challenge. The amount of power required to run each of the fastest supercomputer systems for one year can exceed the power requirements of a city of 40,000 people [27]. In addition, it is expected that future exascale systems will be required at least 40 Gigaflops/Watt in order to maintain expected performance improvements for these systems [38].

In order to reduce power consumption novel methods must be utilized by both hardware vendors and application developers to reduce the power and energy requirements of scientific applications. Methods to reduce power consumption often employ using dynamic frequency and voltage scaling (DVFS) to reduce power and energy consumption [15][19][20][34][40][58]. However, additional methods must also be incorporated to further reduce the power consumption of an application, such as optimizing the application to better make use of the memory sub-system to reduce runtime and power consumption per workload. In this dissertation, will utilize DVFS and DCT to reduce power consumption on multicore systems and optimize the application using loop blocking and loop unrolling for further reduction in runtime and power consumption.

1.2 Modeling Infrastructure

In this work, we use MuMMI (Multiple Metrics Modeling Modeling Infrastructure) [51], which facilitates systematic measurement, modeling, and prediction of performance, power consumption and performance-power trade-offs for multicore systems. This dissertation work will be incorporated into MuMMI for the modeling and prediction components. Figure 3 provides an overview of the MuMMI framework used in this work. The MuMMI framework builds upon three existing frameworks: Prophesy [63], PowerPack [28], and PAPI [56]. We use the SystemG power-aware cluster to conduct our experiments.

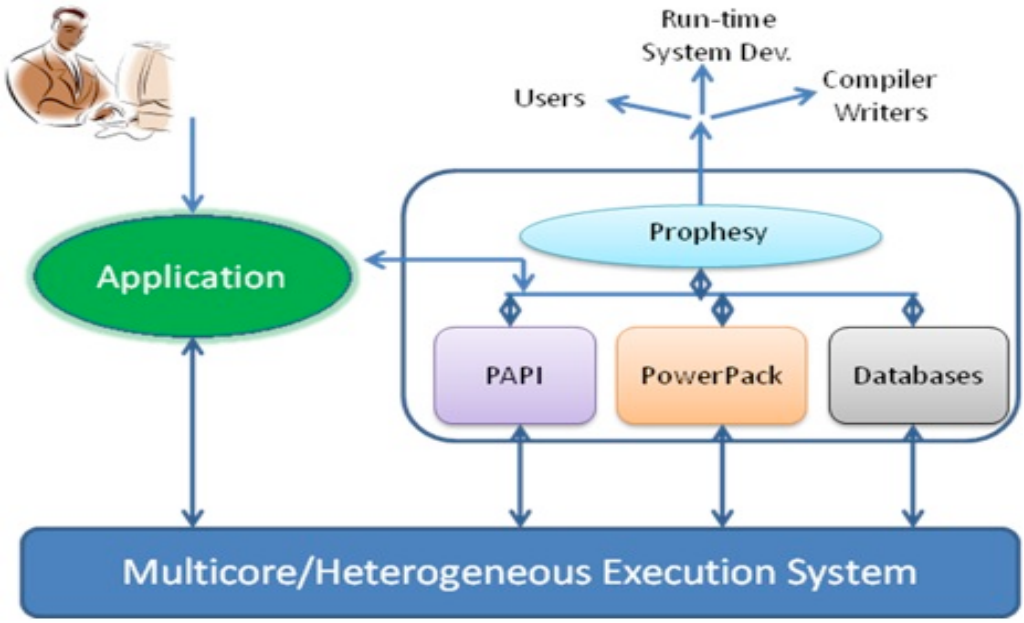

Figure 3. Multiple Metrics Modeling Infrastructure [51]

1.2.1 Prophesy

Prophesy is an infrastructure for analyzing and modeling the performance of parallel and distributed applications. The core component of Prophesy is a relational database that allows for the recording of performance data, system features and application details. The overall framework for Prophesy is illustrated in Figure 4 and consists of three major components that include: data collection, data analysis, and three central databases.

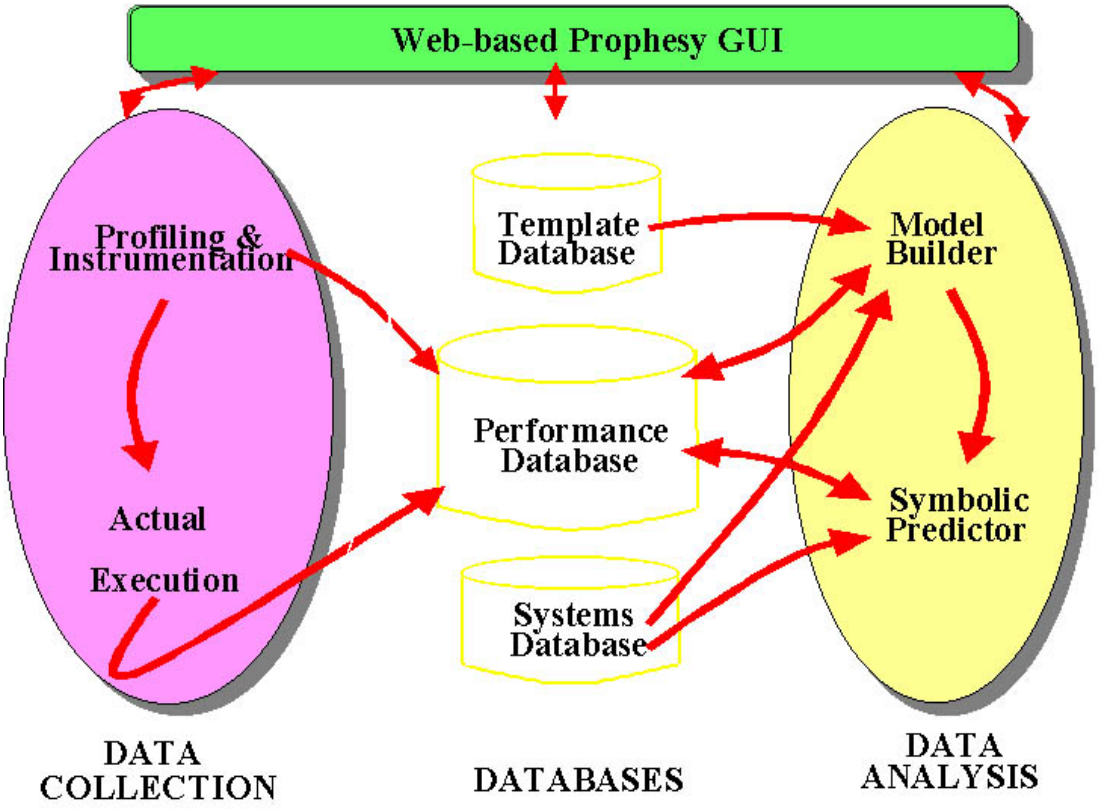

Figure 4. Prophesy Framework [63]

Prophesy allows for automatic instrumentation of codes at the level of basic blocks, procedures, or loops. In addition, a user can specify instrumentation at different granularities as well as instrument the code manually. Data collected using prophesy can be uploaded to the performance database and used to predict the performance of scientific applications under different system configurations.

1.2.2 PAPI

The Performance Application Programming Interface (PAPI), shown in Figure 5, originated at the University of Tennessee's Innovative Computing Laboratory as a

project aimed at providing a portable, standardized API to access hardware performance counters [56]. The performance counters available on PAPI can be used by application developers to gain additional insight into the performance of code sections in scientific applications. PAPI presents a portable API that can be used for accessing performance counters within an application on different systems through code instrumentation. Figure 4 provides an overview of the PAPI API and Framework.

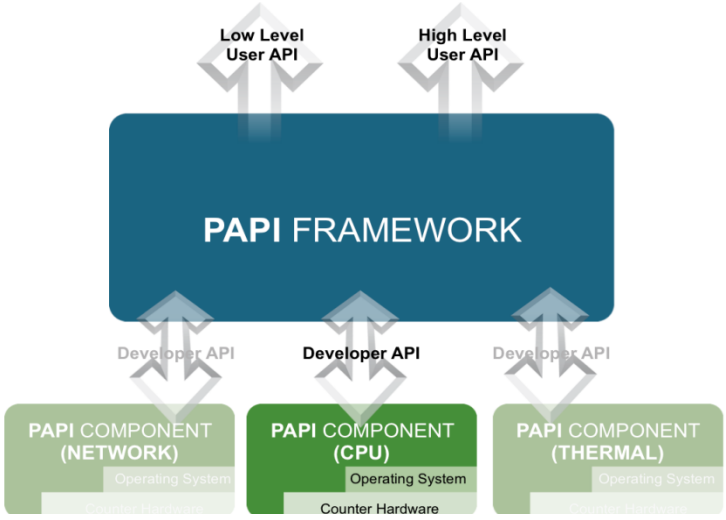

Figure 5. PAPI Framework [56]

1.2.3 PowerPack Framework

We used PowerPack 3.0 [28], shown in Figure 6, which provides power profiling information for advanced execution systems, to measure the power consumption for our applications running on the SystemG platform. The PowerPack framework shown in Figure 4 is a collection of software components, including libraries and APIs, which

enable system component-level power profiling correlated to application functions. PowerPack obtains measurements from power meters attached to the hardware of a system. The framework includes APIs and control daemons that use DVFS (dynamic voltage and frequency scaling) to enable energy reduction with very little impact on the performance of the system. As multicore systems evolve, the framework can be used to indicate the application parameters and the system components that affect the power consumption on the multicore unit. PowerPack allows the user to obtain direct measurements of the major system components' power consumption, including the CPU, memory, hard disk, and motherboard.

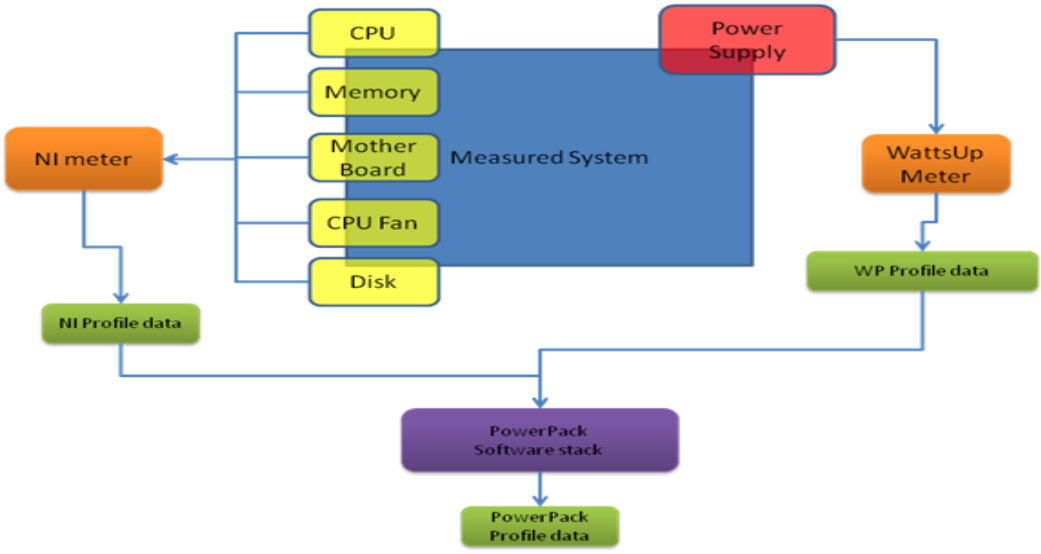

Figure 6. PowerPack Framework

In this work, power consumption is measured on one main node and then remapped to other nodes on the system. The remapping method is used because of the

limited number of power measurement instruments provided across the system. When executing an application, the PowerPack API and data acquisition measurements are used to provide for fully automated application profiling of power consumption.

1.3 Related Work

Several techniques have been used for predicting the power consumption of scientific application on parallel systems and reducing the power consumption. The most common techniques used for predicting or estimating the power consumption involve using system-level hardware counters to estimate the power consumption of the application. These techniques often use the same set of performance hardware counters for estimating the power consumption of the application. Methods for reducing power consumption have leveraged the load-imbalance of the applications to reduce frequency during communication phases within an application.

The use of performance counters to predict power consumption has been explored in previous work [9][10][17][18][19][48]. In general, this work identifies a set of common performance counters to be used across all of the applications considered. These previous methods develop a unified model using a group or class of applications to estimate power consumption. This approach measures activity on the system and correlates it to the power consumption being used by the application. The same counters and correlation coefficients are used for the class or group of applications. This approach is able to provide an accurate estimate of the power consumption of the class or group of applications, but it doesn't capture some of the characteristics unique to each

application. In contrast, E-AMOM is focused on developing models for each application and thereby understanding the unique characteristics of each application that impact runtime and power consumption. In our work, we are able to identify which counters could be seen as common across the different applications, such as PAPI_L2_TCH and PAPI_L2_TCA, in addition to identifying counters that are unique to each application. For example, for the NAS SP-MZ application we were able to determine that the L1 cache misses and L1 instruction cache misses affected the system power consumption more that L2 cache activity. Further, E-AMOM uses the performance counters to identify methods for reducing power consumption.

1.3.1 Performance Modeling

Researchers have used performance modeling to understand the performance of scientific applications on various systems. Almasi et al. use performance models to study the computational and communication kernels of a protein folding application [3]. Their study led to performance predictions for the application on an IBM Blue Gene/C architecture. Additionally, there has been work to focus on understanding the system and application parameters that are likely to affect performance[15][61][67]. In [15] Chen et al. present a performance resource framework to understand the performance characteristics of applications on chip-multiprocessors and the resources that such applications would require. In [61] A. Snavely et al. developed a single-processor model and network model to simplify the approach for performance prediction on several large-scale HPC systems. The work in [61] focused on decomposing an application's signature

to the constraints of memory usage and communication for scientific applications. The end result of this work was to achieve a performance prediction of an application on a targeted system. E-AMOM allows for performance prediction of an application and also uses loop optimizations to reduce runtime and DVFS and DCT to reduce power consumption on multicore systems. In addition, E-AMOM develops multivariate regression models of scientific applications for runtime and component power consumption, such as the total system, CPU, and memory components. E-AMOM models are able to determine which application characteristics affect the performance of the application on a multicore system. Further, the models can be used to determine appropriate optimization techniques for reducing runtime and power consumption with regards to each application.

In the work of Alam and Vetter, they developed platform independent parameterized requirement models for projecting future workloads in large-scale scientific applications [2]. The requirement models focused on understanding the floating-point computation, memory operations, and message passing communication patterns for scientific applications. Our work differs from this work in that we use multivariate regression to model application runtime and power consumption on multicore systems. This work did not consider optimizing the performance of the application, which is the reason for the development of our models using E-AMOM.

In [29] research is conducted to determine the point in an application's execution in which energy savings can be achieved with small increases in execution time for application codes. This work also analyzes the feasibility of saving energy by using

more nodes at a reduced frequency. The metrics include misses per operation (MPO) and slack to determine frequency scaling and gear usage. MPO is determined based on number of operations retired and L2 cache misses from performance counter measurements. The work explores performance in terms of single node (and single processor) as well as multiple node performance. The research identifies characteristics of the application with regards to speed up being (1) poor speedup, (2) perfect/superlinear speedup, (3) good speedup. Our work complements this work in that we are able to develop performance models that are used to determine appropriate energy savings in an application at both the application-level and kernel-level. In all six applications, E-AMOM is able to achieve energy savings without increasing the execution time of the applications.

In [35] kernel coupling values were used to predict parallel application performance using the NAS Parallel Benchmarks. This work focused on decomposing the BT, SP, and LU benchmarks into kernels based on the execution time. Our work provides a modeling methodology to determine what optimization techniques should be used to improve performance on multicore systems. Our work differs from kernel coupling in that we focus on understanding how different application factors, such as L2 cache utilization, affect the performance of the application. Kernel coupling measures the impact of performance of the adjacent kernels within a scientific application. We develop models for each application based on the total execution of the application and then use those models to predict performance at the kernel level.

The use of simulators is an important area used in evaluating the performance of scientific applications. Simulators can be used to provide application and system developers with a better understanding of key design constraints for an unavailable system or system component. Simulations are often used in better understanding design changes as they relate to performance and power consumption in high performance computing. Simulations are useful, but can be very time consuming for large-scale systems and require a larger order of magnitude in execution time for peta and exascale systems [5][17][23][39]. Our work does not make use of simulators and we therefore utilize a fully instrumented power-aware system to validate our experimental models.

1.3.2 Power Prediction

There has been extensive research focusing on understanding the trends exhibited by scientific applications in terms of performance and energy consumption.

In [50] a metric called critical power slope is introduced to explain the efficiency in executing an application at a given frequency for various systems. This work makes use of six micro benchmarks to measure various statistics in system and application performance, such as: access to register, L1 cache (read), L1 cache (write), access to memory (read), access to memory (write), and disk read. The work introduces analytical metrics for determining the energy required to complete work, W, at various frequencies. The energy efficient critical power slope of a system is determined based on minimizing the time in active state and maximizing the idle time of the system. The critical power slope metric focuses on understanding how to reduce the

energy consumption of a system that executes different workloads. E-AMOM differs in that we focus on understanding and optimizing the performance of scientific applications on a multicore system through DVFS, DCT, loop blocking and loop unrolling. Initially, we focus our work on identifying techniques that will reduce energy consumption for a single hybrid or MPI application executing on a multicore system.

In [58] a method is presented to find an energy minimization schedule that is based on performance modeling, performance prediction, and program execution. This is applied to the NAS Parallel Benchmarks (BT, CG, EP, FT, IS, LU, MG, and SP). A schedule is determined based on a combination of techniques to minimize the energy consumption of the given scientific application. The schedule for executing the application is determined based on a partial execution of each application. The energy limit is typically chosen to be 10% of peak energy consumption of the program. The average error of their work was 2.1% with a worse case scheduling error of 6.1%. The typical energy minimization schedule was 5-10% in energy consumption.

In [10] performance counters are used to provide models of power measurements of the complete system based on a method known as the “trickle-down” approach. The work provided estimation of power consumption for the system including chips, memory, I/O, and disk. In this work, the average error was less than 9% for the SPEC CPU 2000 benchmarks. This work provided a system-centered approach to modeling based on the correlation of performance counter events to applications. The counters utilized to estimate power consumption for the system included L3 Cache misses, TLB Misses, DMA Accesses, Memory Bus accesses, and I/O Interrupts. This work utilizes

uses a linear model as the first step in predicting power consumption based on the counters with the best correlation for each component. Our work differs in that we identify different counters for each application and do not use the same counters across all applications. We make algorithmic changes to our applications to improve performance through loop unrolling and loop blocking. In addition, we reduce the power consumption of the applications through DVFS and DCT.

Lim et al. present a surrogate estimation model using performance counters is presented on an Intel Core i7 system to estimate for CPU, Memory, and the total system power for OpenMP benchmarks up to 8 threads [43]. The median error was 5.32% on the system. In this work various Intel Core i7 specific counters that were representative of the system features were utilized. For example, this work used counters that represented the number of unhalted cycles in the CPU and retired instructions for building the CPU power model. To estimate the power consumption for the applications a robust regression model was built that was able to apply weights to each data point. This work used the spearman correlation to reduce 17 performance counters down to 7 counters for predicting power consumption for the system, CPU, and memory components. Our work makes use of the spearman correlation coefficients for reducing 40 counters initially, but further determines appropriate counters based on regression and principal component analysis. The combination of spearman correlation, multivariate regression, and principal component analysis is able to reduce the required number of performance counters needed for our modeling work.

In [22] power estimations using counters are presented with median errors of 5.63%. This work makes uses of performance counters to measure the effects of cache resource and thermal effects to develop a power-aware thread scheduler. The work presented by Singh classifies performance counters of the AMD Phenom processor into four different groups based on FP Units, Memory, Stalls, and Instructions Retired. Using these counter groups analytical models are derived using micro-benchmarks to develop an online thread scheduler. The piece-wise linear models developed in this work were tested on the SPEC2006, SPEC-OMP, and NAS benchmarks. Our work differs in that we use our performance models to optimize the performance of the application by using DVFS, DCT, loop blocking, and loop unrolling. This work focuses on estimating power consumption for thread scheduling. Our work focuses on reducing power consumption through application optimization. Throughout our work we utilize 15 different performance counters is modeling application performance. Our models commonly have activity that relates to the L2 cache (PAPI_L2_TCH, PAPI_L2_TCA), which is why we utilize loop blocking and loop unrolling to reduce runtime and power consumption.

1.3.3 Power Reduction Strategies

There is extensive research dealing with reducing power consumption in large-scale HPC applications. In [19] a user-level library framework is introduced that allows for the online adaptation of multithreaded application codes. This work uses the Instructions per cycle (IPC) metric and several run-time specific performance metrics to

predict application performance. A linear regression is applied to the offline training model to develop an accurate power-performance model. The performance prediction model is used to determine concurrency levels on a SMT chip at two levels, focusing on the (1) number of threads per processor and (2) the number of processors to use. The counters that this work specifically focuses on include rate of bus accesses, rate of L2 cache misses, % of cycles in which the processor's trace cache is in deliver mode, rate of branch instructions, rate of misspredicted branches, and retired instructions per cycle. The work in [19] is further extended to evaluate the effects of mapping different set of threads to cores on hierarchical multicore systems in [21]. This work applies prediction strategies for reducing energy consumption using DVFS and dynamic concurrency throttling (DVT) based on slack in communication. Our work also applies prediction strategies for reducing energy consumption; however, we determine which kernels in an application can be optimized to decrease energy consumption through DVFS, DCT, and loop optimizations**.**

A technique aimed at reducing power consumption through task placement is introduced in [44]. This work measures the affect that different MPI aggregation strategies have on application's performance and energy consumption. The NPB 3.2 benchmarks are used to test this methodology, focusing on the FT, LU, CG, and BT benchmarks. The model introduced accurately determines the effect of distributing tasks across cores of multiple nodes has on execution time and energy. The performance during computation phases is predicted using Instructions Per Cycle (IPC). The IPC for specific MPI tasks is predicted on a targeted system. The IPC is used with 12 training

benchmarks from SPEC MPI 2007 benchmarks to determine optimal MPI aggregation. The focus of this work is on reducing energy consumption through task placement on a system. Task aggregation is accomplished by using more nodes for executing an application but fewer processors per node are used. In [44] the average performance gain was 5%, but energy reduction was over 60%. Our work reduces energy consumption by reducing the runtime and decreasing power consumption through DVFS, DCT, loop blocking, and loop unrolling.

In [19] a multi-dimensional, online performance prediction framework that uses DVFS and DCT on OpenMP applications is presented to reduce power consumption. The framework used in this work makes use of statistical linear regression models for predicting application performance. The performance prediction model evaluates the effects of mapping different set of threads to cores on hierarchical multicore systems and models the effects of DVFS and DCT. A baseline prediction model is used to measure useful IPC (μIPC) with dependence functions for a target and configurations. The results provided in this work show a DVFS model median error of 3.0%, DCT model median error of 7.3%, and a unified model median error of 6.1%.

In [25] multiple energy gears are used in an application in an attempt to achieve performance and energy savings. This work applied DVFS to HPC application codes that were divided into application phases based upon the memory pressure (OPM) of the application. The significant results of this work allowed for savings of 10% energy with 5% time penalty for NAS BT, 11% energy reduction with 4% time penalty for NAS MG, and a 16% energy saving with a 1% time penalty for NAS IS. In [33] two energy-

saving techniques, DVFS and DCT, are applied to Hybrid (MPI+OpenMP) HPC application codes to improve energy consumption. This work focuses on reducing energy consumption by identifying the effects that DCT has on other MPI tasks during execution and identifying slack due to intra and inter-node interaction in hybrid HPC applications. The methodology is applied to the NPB-MZ suite and ASC Sequoia benchmarks with energy savings in the range of 4.1% to 13.8% with negligible performance loss.

Our work extends upon this work by presenting a methodology that is able to predict performance in application kernels utilizing DVFS and DCT for MPI and hybrid (MPI/OpenMP) applications. Our scheme makes use of performance models that are used for predicting the effects that DVFS and DCT strategies have on application performance by refining the regression model for each application's characteristics. Our work differs from previous approaches in that we identify alternative frequency and concurrency settings for an application's kernel to reduce power consumption. We also optimize the kernel for better performance through loop blocking and loop unrolling. The reduced power consumption and reduced execution time reduces the energy consumption of the application. Previous methods focus largely on only introducing software-based power reduction strategies. In our work, we utilize software-based power reduction strategies with algorithmic changes, such as loop blocking and loop unrolling, to improve application performance.

2. PROPOSED PERFORMANCE MODELING SCHEME

In this section we provide background information about performance issues related to multicore systems. We also give a brief outline of our performance modeling methodology, E-AMOM, which is used to predict execution time and power consumption of MPI and Hybrid scientific applications on multicore systems. The details of the modeling method are given in Section 4. Throughout the remainder of this dissertation, Hybrid will refer to an application with communication constructs based upon MPI and OpenMP. Further performance will refer to runtime and power consumption.

2.1 Energy-Aware Modeling and Optimization Methodology (E-AMOM)

In this section we present our power-aware performance modeling and optimization scheme. E-AMOM can be used for predicting the runtime and power consumption of the application in terms of System, CPU, and Memory components. E-AMOM is also used to improve the runtime and power consumption of scientific applications on multicore systems. Figure 7 presents a high level view of E-AMOM, which consists of the following steps:

1. An application is selected for evaluation on a target multicore system.

2. The performance-tuned principle component analysis method identifies appropriate performance counters that represent each performance component of the application.
3. Performance-power modeling is used to model the application kernels of the application to identify appropriate optimization strategies, which include DVFS, DCT, loop blocking, and loop unrolling.
4. The application implementation is optimized for the target multicore system.

E-AMOM makes use of a performance-tuned principle component analysis method for modeling application performance on multicore systems. A brief overview of the analysis method is given below; the details of the method used to develop the models is given in Section 4. Additionally, we improve the performance of the application to reduce both performance and power consumption on multicore systems based on the application assumptions derived from our modeling scheme. Figure 7 provides an overview of the performance modeling and optimization scheme used in this work.

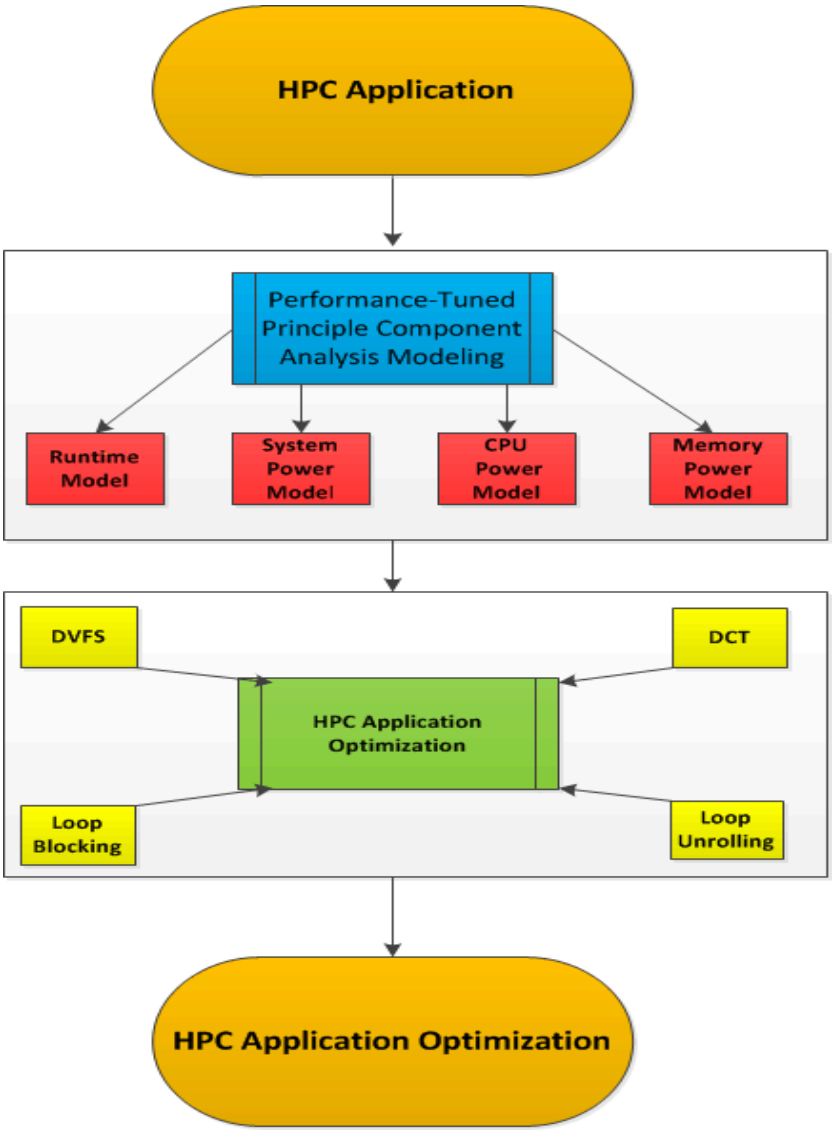

Figure 7. E-AMOM Schema

E-AMOM provides application developers and users a methodology to model and optimize the performance of scientific applications on multicore systems. Figure 8 provides an illustration of how E-AMOM is integrated into the MuMMI framework. E-AMOM is a modeling component that is integrated into Prophesy, which allows for analytical, parameterized, and kernel coupling models to be developed.

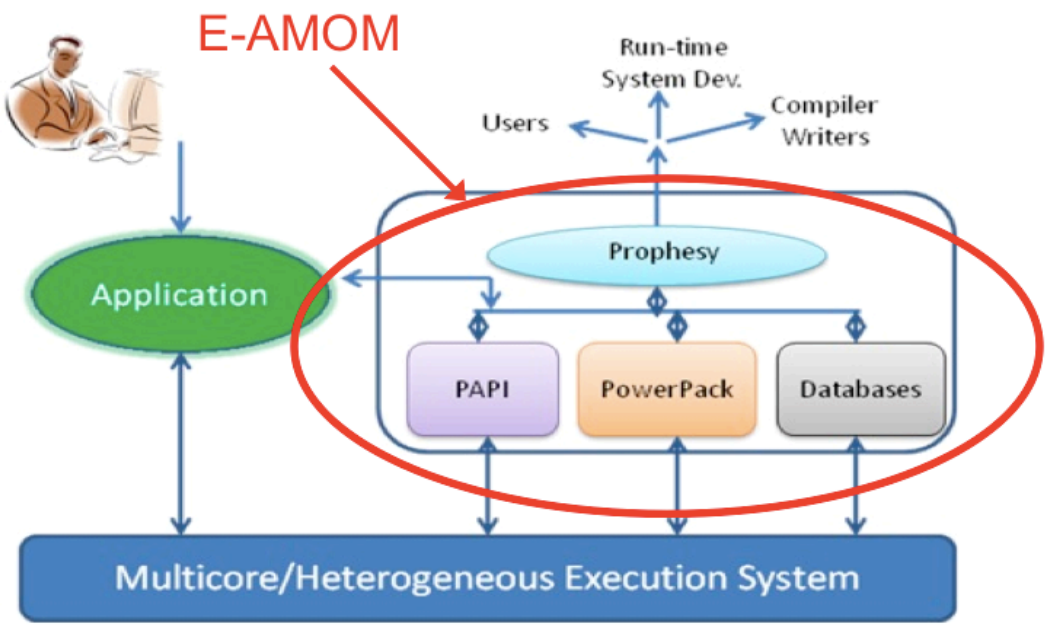

Figure 8. E-AMOM Integration into MuMMI

The models developed using E-AMOM utilize Prophesy's instrumentation framework, PowerPack for collecting power profiles, and PAPI for collection performance counter data. This allows for users of MuMMI to model the performance of scientific applications in regards to runtime and component power consumption. The models from E-AMOM can then further be used to predict runtime and power consumption of applications for different input sizes, frequency settings, concurrency settings, and number of processors.

2.2 Performance-Tuned Principle Component Analysis Method

In this section we introduce the performance-tuned principle component analysis method, which is the modeling component of E-AMOM. Using this method, we seek to

explore the application characteristics (via performance counters) that affect performance in order to gain a better understanding of how the application can be modified to improve performance with respect to runtimes and power consumption of the system, CPU, and memory. We explore the following issues in regards to modeling the runtime and power consumption of scientific applications on multicore systems:

a) What are the application characteristics that affect runtime and power consumption in scientific applications?
b) Which combination of performance counters can be used to model the application in terms of runtime, system power, CPU power, and memory power?
c) What is the accuracy of our models in estimating performance (runtime and power consumption for system, CPU, and memory components?
d) What characteristics of applications can be optimized to improve performance on multicore systems?

During each execution we capture 40 performance counter events utilizing the performance application programming interface (PAPI) [56] and the perfmon performance library. All performance counter events are normalized using the total cycles of execution to create performance event rates for each counter. Performance counter values must be normalized by a common variable (total cycles) so that underlying characteristics of the performance counter values can be prepared. Table 2 provides an overview of the 40 performance counters are analyzed for each application using a performance-tuned supervised principal component analysis method. In addition,

we make use of non-negative regression coefficients models to ensure that they were representative of realistic performance scenarios.

Table 2. Hardware Performance Counters

Hardware Counter	Description
PAPI_TOT_INS	Total instructions completed
PAPI_FP_INS	Floating point insturctions
PAPI_LD_INS	Load instructions
PAPI_SR_INS	Store instructions
PAPI_TLB_DM	TLB data misses
PAPI_TLB_IM	TLB instruction misses
PAPI_VEC_INS	Vector/SIMD instructions
PAPI_L1_TCA	L1 cache total accesses
PAPI_L1_ICA	L1 instruction cache accesses
PAPI_L1_ICM	L1 instruction cache misses
PAPI_L1_TCM	L1 total cache misses
PAPI_L1_DCM	L1 data cache misses
PAPI_L1_LDM	Level 1 load misses
PAPI_L1_STM	Level 1 store misses
PAPI_L2_LDM	Level 2 load misses
PAPI_L2_STM	Level 2 store misses

Table 2: Continued

Hardware Counter	Description
PAPI_L2_STM	Level 2 store misses
PAPI_L2_TCH	L2 total cache hits
PAPI_L2_TCA	L2 total cache accesses
PAPI_L2_ICM	L2 instruction cache misses
PAPI_CA_SHARE	Access to shared cache line
PAPI_HW_INT	Hardware interrupts
PAPI_CA_ITV	Cache line interventions
PAPI_BR_INS	Branch instructions completed
PAPI_RES_STL	System stalls on any resource
Cache_FLD_per_instruction	L1 writes/reads/hits/misses
LD_ST_stall_per_cycle	Load/stores stalls per cycle
bytes_out	Received bytes transmitted
bytes_in	Sent bytes transmitted
IPC0	Instructions Per Cycle Core 0
IPC1	Instructions Per Cycle Core 1
IPC2	Instructions Per Cycle Core 2
IPC3	Instructions Per Cycle Core 3
IPC4	Instructions Per Cycle Core 4
IPC5	Instructions Per Cycle Core 5

Table 2: Continued

Hardware Counter	Description
IPC6	Instructions Per Cycle Core 6
IPC7	Instructions Per Cycle Core 7
LLC_miss_rate0	Lower Level Cache Miss Rate Core 0
LLC_miss_rate1	Lower Level Cache Miss Rate Core 1
LLC_miss_rate2	Lower Level Cache Miss Rate Core 2
LLC_miss_rate3	Lower Level Cache Miss Rate Core 3
LLC_miss_rate4	Lower Level Cache Miss Rate Core 4
LLC_miss_rate5	Lower Level Cache Miss Rate Core 5
LLC_miss_rate6	Lower Level Cache Miss Rate Core 6
LLC_miss_rate7	Lower Level Cache Miss Rate Core 7

$$y = \beta_0 + \beta_1 * r_1 + \cdots \beta_n * r_n \qquad (1)$$

Each multivariate linear regression model is constructed for each performance component (execution time, system power, CPU power, and memory power) for each application.

2.3 Application Optimization Methods

In this section we discuss the methods that are used to improve the performance of scientific applications based upon our modeling scheme. The methods include scaling

the frequency of the application (DVFS), Dynamic Concurrency Throttling (DCT), and improve application's utilization of the memory subsystem. Each method is described below.

2.3.1 Dynamic Voltage and Frequency Scaling

Dynamic Voltage and Frequency Scaling is a technique that is used to reduce the voltage and frequency of a CPU in order reduce power consumption [30][36]. Using DVFS, we minimize the power consumption of our scientific applications by reducing the voltage and frequency of the application during performance periods. Applying DVFS is especially beneficial during period where communication slack time appears during parallel execution due to load imbalance between task communications. We use our performance-tuned principle component modeling method to determine execution time and power consumption during application periods at reduced frequencies. For simplicity, we assume that all cores that execute during the application phase will run at the same frequency.

2.3.2 Dynamic Concurrency Throttling

Dynamic Concurrency Throttling is a technique that can be used to reduce the number of threads used to execute an application [19]. Using DCT, we minimize the power consumption of our HPC applications by reducing the use of the number of cores (OpenMP threads) of the application during performance periods. Applying DCT is

especially beneficial during OpenMP performance phases that do not benefit from using the maximum number of OpenMP threads per node. We use our performance-tuned principle component modeling method to determine execution time and power consumption during application periods at reduced concurrency settings.

2.3.3 Conventional Techniques: Loop Blocking and Loop Unrolling

In high performance computing, much of the computation involved with parallel scientific applications such occurs within nested loops in each application function. Optimizations made to these application loops can leave to improved performance on multicore systems. In this section, we discuss loop blocking and loop unrolling, which can be used to improve performance of the scientific applications.

Loop blocking is a well-known loop optimization technique to aid in taking advantage of memory hierarchy; its main purpose is to eliminate as many cache misses as possible [47][60]. This technique transforms the memory domain of an application into smaller chunks, such that computations are executed on the chucks that easily fit into cache to maximize data reuse. The optimal loop block size varies with different applications on different systems.

Loop unrolling is a well-known code transformation technique that replicates the original loop body multiple times, adjusts the loop termination code and eliminates redundant branch instructions. Outer loop unrolling can increase computational intensity and minimize load/stores, while inner loop unrolling can reduce data dependency and eliminate intermediate loads and stores. We combine inner and outer loop unrolling to

improve the performance the scientific applications. For examples, we unroll the inner loops four times for four major double nested loops in GTC code so that we reconfigure the double nested loops into the single loops, then use compiler directives for further loop unrolling.

2.4 Modeling Approaches Leveraged

In this section, we discuss the initial modeling approaches considered, why these approaches were not used, and how these approaches led to the development of our final power-performance modeling approach. These approaches were based on extending several existing frameworks with the goal of providing new tools to provide for modeling methods for scientific applications. Further development led to a modeling framework, which focused upon analyzing the performance characteristics of scientific applications via performance counters that could be used for additional improvements to the application in term of performance-power tradeoffs.

2.4.1 Initial Hierarchical Modeling Approach

An initial hierarchical modeling approach was developed that leveraged existing frameworks to model the performance of large-scale scientific applications on multicore systems. This modeling approach provided for a hierarchical decomposition of applications and quantified the sharing of resources on multicore systems by focusing on

the utilization of energy and power consumption on multicore systems. Figure 9 provides an overall depiction of this initial hierarchical modeling framework.

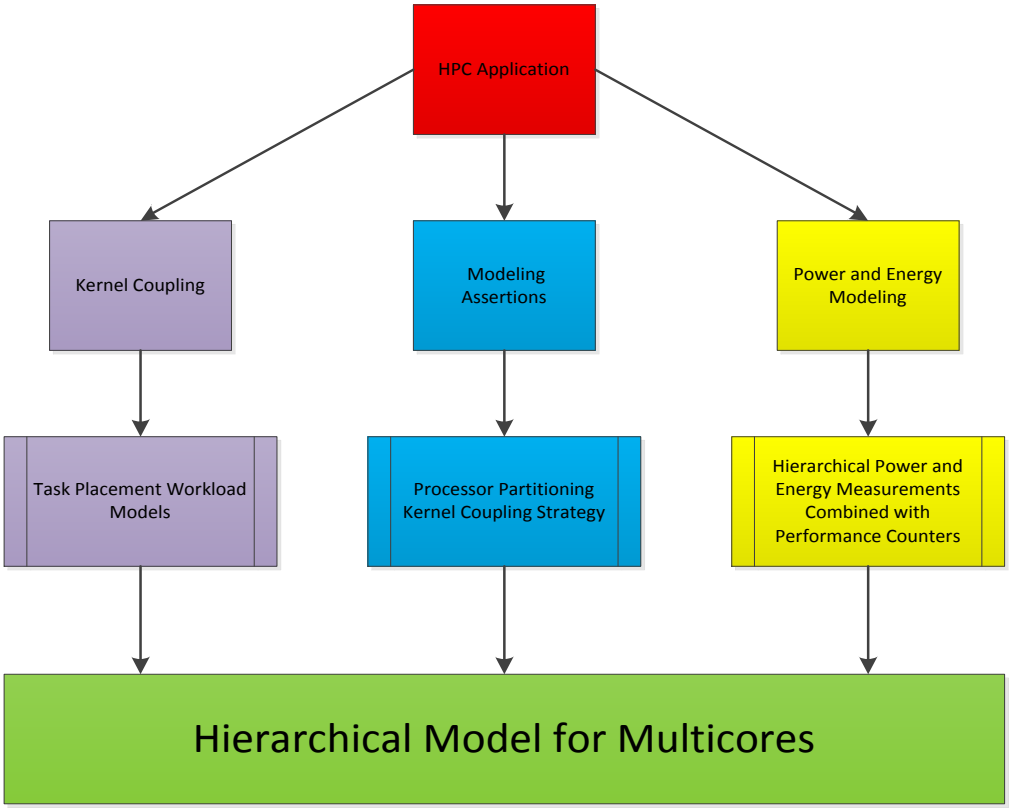

Figure 9. Initial Hierarchical Multicore Modeling Scheme

This modeling methodology allowed for detailed models of a scientific application. The following issues related to analyzing the performance of large-scale scientific applications and multicore systems were represented by the modeling methodology:

1) Application Decomposition: Determining an efficient decomposition of an application's kernels so that it effectively demonstrates the workload distribution

onto multicore systems. This was accomplished by utilizing a large-scale scientific application modeling framework, Modeling Assertions (MA).

   a. Modeling Assertions: A framework that provides for detailed information about the computation and communication workload characteristics of scientific applications.

2) Quantification of Resource Sharing: Measuring how an application's kernels make use of the memory subsystem is an important issue that can provide insight into which application kernels must be improved. The quantification of resources on multicore systems makes use of Prophesy infrastructure and the kernel coupling method.

   a. Prophesy: An infrastructure for analyzing and modeling the performance of parallel and distributed applications. The core component of Prophesy is a relational database that allows for the recording of performance data, system features and application details.

   b. Kernel coupling: The kernel coupling metric quantifies the interaction of adjacent kernels in a large-scale scientific application. The prophesy infrastructure computes kernel coupling values in our large-scale scientific applications.

3) Performance vs. Power relationship: the tradeoff between application performance and power on multicore cluster systems is examined. This relationship focuses on understanding the effects that application decomposition, data inputs, performance, and power have on each other.

a. PowerPack: The framework enables distributed systems to profile, analyze, and conserve energy in scientific applications using dynamic voltage scaling.

In the following sections we describe each component of the initial modeling framework and then discuss lessons learned and how we developed the existing framework.

2.4.1.1 Modeling Assertions

The modeling assertions (MA) framework was introduced by Alam and Vetter to provide for incremental model construction and validation **Error! Reference source not found.**. Modeling assertions combines empirical and analytical modeling techniques together to encapsulate the workload requirements of an application. There are a number of steps involved in the model creation process with MA. These step are to:

1) Determine and declare application variables that affect performance.
2) Determine and declare application operations that affect performance.
3) Refine the performance model in incremental steps.
    a. Validate performance model empirically at runtime using performance assertions.
    b. Refine model based on these error rates by adding and modifying variables and operation declarations.

c. Terminate modeling process when model is representative and when error level is acceptable.

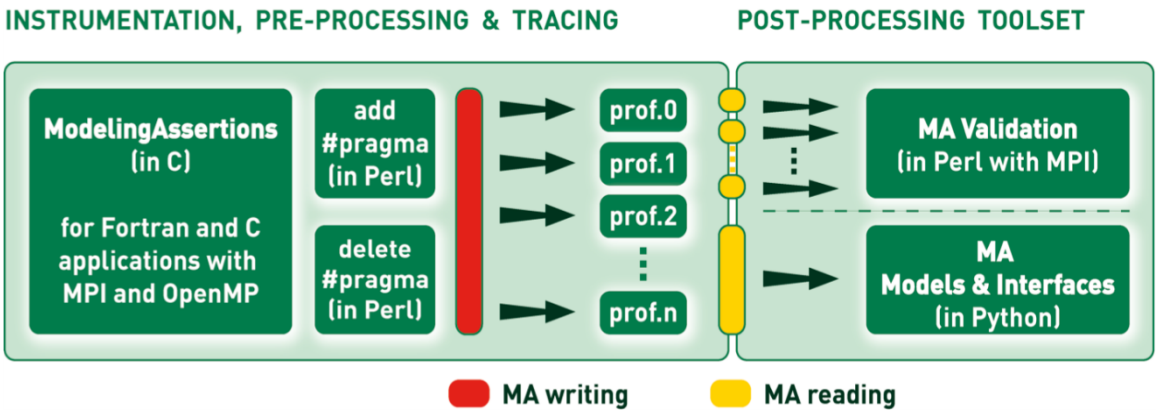

Figure 10. Modeling Assertions Framework

Figure 10 shows the scheme for the modeling assertions framework. In this example, we use a code for a matrix-matrix multiply kernel to illustrate the capabilities of the modeling assertions framework. We focus on the ability of modeling assertions to create symbolic models of an application code. Our matrix-matrix multiply example is written in fortran and uses OpenMP for communication.

First, modeling assertions creates a ma_profile of the matrix-matrix multiply code, shown in Figure 11. The ma_profile shown in Figure 11 provides an overview of the matrix-matrix multiplication code and the attributes that modeling assertions captures. First, modeling assertions defines the input parameters and their values for the application code. These parameters are then used to represent the number of loops in the code. Finally, this profile shows a declaration for the ma_flop_start routine to measure

the number of application flops for a segment of the code, in this example the number of flops occur within the “main_loop” of the code are represented.

```
MA:0:1
MA:----- Modeling Assertions enabled ---------
MA:-----------------------------------------
MA:0:ma_init:COMM_WORLD:0
MA:1: ma_subroutine_start:main
MA:1: ma_def_variable_int:L:200
MA:1: ma_def_variable_int:M:200
MA:1: ma_def_variable_int:N:200
MA:2:  ma_loop_start:main_loop:L*M*N:8,000,000
MA:3:  ma_flop_start:main_loop_fl:L*M*N*2:16,000,000
            *
            *
MA:2:  ma_subroutine_end:main
MA:1: ma_finalize:main
```

Figure 11. Matrix-Matrix Multiply MA Profile

After the modeling assertion profile is validated a control-flow model of the code can be generated, shown in Figure 12. This is a high-level, octave compatible code that can be used for a simulator and represents the symbolic variables of the code that affect performance on a parallel system. The partial control-flow shown in 9 provides key input parameters, loops, and parameter assignments for the code.

```
        Main(){
L; M; N;
a(L,M), b(M,N), c(L,N)
loop(NAME=loop-1) (COUNT=M)
    loop(NAME=loop-1-2) (COUNT=L)
    loop(NAME=loop-1-3) (COUNT=N)
 loop(NAME=loop-1) (COUNT=L)
    loop(NAME=loop-1-2) (COUNT=N)
    loop(NAME=loop-1-3) (COUNT=M)
   c(i,j) = c(i,j) + a(i,k) * b(k,j)
                          (remaining code)
```

Figure 12. Matrix-Matrix Multiply Model

Using modeling assertions, application requirements can be encapsulated with respect to computation and communication requirements. In Figure 13, we analyze the sensitivity of the kernels floating-point operations and instructions per cycle to changes in the matrix size. The variances in these application parameters for floating-point operations and instructions per cycle for different parameters can be studied.

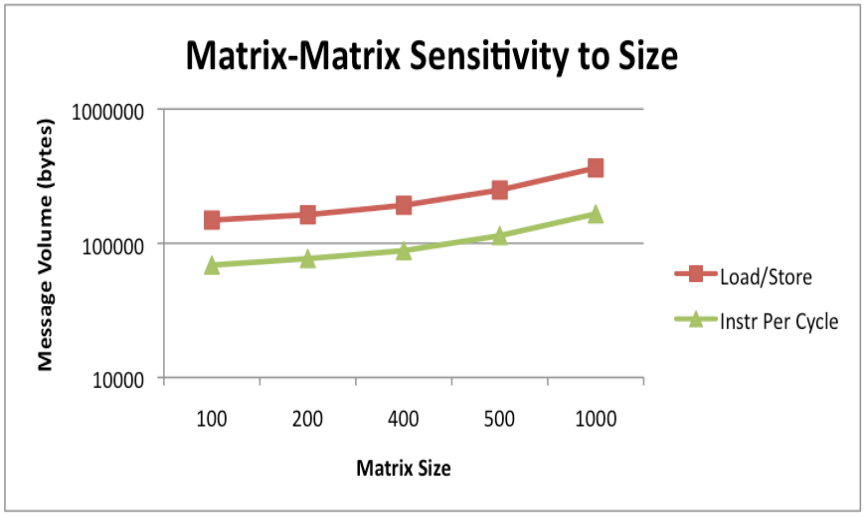

Figure 13. Matrix-Matrix Sensitivity Analysis

2.4.1.2 Kernel Coupling

In previous work, Geisler and Taylor provided the specifications for quantifying the interaction between adjacent kernels in an application [55]. A kernel is defined as a unit of computation that denotes a logical entity within the larger context of an application. In general, a kernel may be a loop, procedure, or file depending on the level of granularity or detail that is desired from the measurements. Our previous work was able to measure decompose the GYRO application into eight kernels, six of the kernels being computational kernels [47]. These kernels represent top-level subroutines that have been grouped together.

To compute the coupling parameter cij, there are three measurements that must be taken:

1) pi is the performance of kernel i in isolation,
2) pj is the performance of kernel j in isolation, and

3) pij is the performance of kernels i and j together, assuming that kernel i immediately precedes kernel j in the application.

The value c_{ij} represents the interaction between two adjacent kernels in an application. In general, for an application consisting of N kernels, only N-1 pairwise kernel interactions need to be measured.

The parameter c_{ij} can be grouped into three categories:

- c_{ij} = 1 indicates no interaction between the two kernels, yielding no change in performance.
- c_{ij} < 1 results from resource(s) being shared between the kernels, producing a performance gain.
- c_{ij} > 1 occurs when the kernels interfere with each other, resulting in a performance loss.

The equation for computing c_{ij} is presented as equation 1.

$$C_{ij} = \frac{P_{ij}}{P_i + P_j} \tag{2}$$

The coupling parameter can be generalized to apply to chains of kernels, as shown in equation 2. The parameter C_{ij} becomes C_w, assume that W represents an ordered chain of K kernels. Therefore, P_w represents the execution time of the chain W. Note that for K=2, equation 1 is equal to equation 2.

$$C_w = \frac{P_w}{\sum_{j \in W} P_j} \tag{3}$$

The coupling parameter is used in the estimation of execution time using equation 3. N_i represents the number of times that kernel i is executed. In the case of GYRO, the kernels occur in loops and N_i represents the number of times that the loop is executed. P_i represents the execution time of kernel i, and α_i is the weighted average of the coupling values that are associated with kernel i.

$$T = \sum_{i=1}^{n} \alpha_i N_i P_i \tag{4}$$

The parameter Q_i represents the set of all ordered chains of k ($2 \leq k \leq$) kernels that are involved with kernel i. The size of the Set Q_i is $\left| Q_i \right| = k$. The coefficient α_i (i = 1, 2,…, n) is represented in equation 4.

$$\alpha_i = \frac{\sum_{W \in Q_i} C_W P_w}{\sum_{W \in Q_i} P_w} \tag{5}$$

In this work we use kernel coupling to measure the utilization of the memory subsystem by the application. This work methodically uses a computation and communication decomposition to determine the application kernels from the modeling assertions framework. Previous work used performance measurements based on execution time or cache misses [55].

The kernel coupling metric can be applied to the matrix-matrix multiply kernel code in the previous section. The example code can be divided into five kernels: initialization, nested_loop1, nested_loop2, and final. The kernel coupling values obtained for kernel 1 (nested_loop-1) and kernel 2 (nested_loop-2) when executed on 4 processors on the Jaguar Cray XT4 are presented in Table 3.

Table 3. Kernel Coupling Values for M-M Kernel

Kernel	Coupling Value
(Loop-1&Loop-2)	0.99337
(Loop-2&Loop-1)	0.99474

Overall, the kernel coupling values show constructive coupling occurring in the matrix-matrix kernels. However, the kernel coupling values as presented do not provide enough insight into how the kernel is performing on the multicore system. Additional insight is needed to understand how the matrix-matrix kernel is utilizing the memory subsystem of the multicore system.

2.4.1.3 Summary of Leveraged Modeling Methods

Overall, the initial multicore modeling framework led to a more detailed understanding of how scientific applications performed on multicore systems. Initially, we utilized the modeling assertions framework to create symbolic models of our scientific applications. The symbolic models were useful for understanding predicted application requirements, such as floating-point operations and MPI communication message size requirements. However, the models that we created using modeling assertions were system-independent and therefore they did not provide the necessary insight about how the application performed on a target system. In order to reduce runtime and power consumption on a given system an understanding of how the

application performs on a given system is required. Additionally, the MA framework was not and currently does not support Hybrid applications.

Kernel coupling provides a metric for analyzing application performance and understanding how kernels of an application shared resources. However, the kernel coupling metric focused on one component of performance, such as execution time or cache misses, in quantifying the sharing of resources. To improve the performance of the application kernel using kernel coupling could require additional performance data to be obtained using performance counters, which could be costly in terms of the amount of time it takes to collect all of the data for kernel coupling. While E-AMOM does not quantify the interactions between kernels as is done with kernel coupling, E-AMOM collects detailed data for modeling using only one execution of the application.

The combined initial framework work focused on extending kernel coupling to incorporate a processor partitioning method to model trends in the kernel coupling values. We then determined and measured how an application utilizes the memory subsystem when executed on a multicore parallel system. Once kernel coupling values were determined for the respective application, the symbolic models developed from MA were used to determine the components in each kernel that were being shared.

The final framework represents an approach based on utilizing over 40 performance counters to measure various application characteristics not reflected in kernel coupling and modeling assertions. We use the performance counters to develop models for predict runtime and component power consumption of the scientific applications on multicore systems. This allows for us to determine different application

characteristics, through performance counters, that affect the runtime and power consumption of the application on a given system. The models developed using our work can then be used to determine appropriate methods for refining the application to reduce runtime and decrease power consumption.

3. PERFORMANCE-POWER TRADE-OFFS OF MPI AND HYBRID APPLICATIONS*

In this section, we discuss the performance characteristics of MPI and Hybrid scientific applications and how they affect performance-power tradeoffs on parallel multicore systems. In Section 3.1 the parallel programming paradigms used in high performance computing for scientific applications are introduced. In Section 3.2 the execution environment for these experiments is introduced. In Section 3.3 an analysis of detailed performance characteristics of three scientific applications for Hybrid and MPI implementations is provided.

3.1 Parallel Multicore Systems

Efficient use of multicores requires that the hierarchical organization of cores be exploited. One way of exploiting the hierarchical organization is to have parallel applications designed to match this organization. Currently, the widely used languages for parallel applications are MPI, OpenMP, or the combination of both languages; this combination is called hybrid. Multicore processors present significant new opportunities such as on-chip high inter-core bandwidth and low latency and present new challenges in

*Part of this section is reprinted with permission from "Energy and Performance Characteristics of Different Parallel Implementations of Scientific Applications on Multicore Systems" by Charles Lively, Xingfu Wu, Valerie Taylor, Shirley Moore, Hung-Ching Chang, and Kirk Cameron, in Int. Journal of High Performance Computing Applications, Volume 25 Issue 3, August 2011 by SAGE Publications, INC.

inter-core resource conflict and contention. In [32][43], it is argued that the full benefit of these architectures will not be harnessed until the software industry and community fully embrace parallel programming. Hybrid applications make use of shared memory programming paradigms as well as message passing paradigm, which appear to be a good fit for multicore systems whereby the nodes utilize a share memory model and distributed memory model between the nodes.

The Message Passing Interface (MPI) is one of the widely used parallel programming models in High Performance Computing [40]. MPI was initially developed as a model for achieving communication across nodes in a parallel system; however, there has been substantial work in improving the intra-node communication protocols [13][14]. Additionally, there have been improvements made to collective communications; one use of collective communication is broadcasting data across the entire system [53][54]. Currently, there are still improvements or modifications being made to the MPI standard and mechanisms in order to make it more efficient for multicores and heterogeneous architectures.

OpenMP is another widely used parallel programming model in High Performance Computing [48]. OpenMP was first introduced in 1997 as a parallel programming language for Fortran and was later expanded to C/C++. The core elements of OpenMP are based upon threads, which are used to parallelize portions of a code that can be executed on a processor in parallel. OpenMP does not deal with message passing, currently, and therefore provides a simple API for parallelizing code.

Generally, MPI is considered optimal for process-level, coarse parallelism and OpenMP is optimal for loop-level fine grain parallelism. Combining MPI and OpenMP parallelization to construct a hybrid program is not only able to achieve multiple levels of parallelism but also to reduce the communication overhead of MPI within a multicore node, by taking advantage of the shared address space and on-chip high inter-core bandwidth and low inter-core latency at the expense of introducing OpenMP overhead due to thread creation and increased memory bandwidth contention. However, which parallel programming paradigm (MPI, OpenMP, or hybrid) is the most suitable for multicore systems depends on the nature of an application, available parallel programming software, and compiler support on these multicore systems.

3.2 Experimental Environment

The experimental environment used for the experiments to explore the performance and power tradeoffs for MPI and Hybrid application is the power-aware multicore cluster, Dori, which is available in the Department of Computer Science at Virginia Tech. Dori, presented in table 4, is composed of eight compute nodes. Each node of the system consists of two dual-core AMD Opteron processors (1.8GHz) and providing 4 CPUs per node. Each CPU in Dori contains L1 and L2 caches with a size of 64KB and 1 MB, respectively, for on-chip memory access. In our experimental results we make use of the frequency settings available in Dori, which includes five settings from 1.0 GHz to 1.8 GHz as described in Table 4. The PowerPack infrastructure,

introduced in Section 1, is used for collecting the necessary performance and power data on the Dori System.

Table 4. System Configuration of Dori

Specifications of Dori System	
Number of Compute Nodes	8
CPUs Per Node	4
CPU Type	1.8Ghz Dual-Core AMD Opteron
L1 Cache	64KB
L2 Cache	1 MB
Memory Per Node	6 GB
Interconnect	1GB/sec Ethernet
Frequency Settings	1.8 , 1.6, 1.4, 1.2, 1.0 GHz

3.3 Experimental Results

In this section we present our experimental results based on three applications described in Table 5: NAS BT [35], GTC [23], and PMLB[69]. We provide the performance characteristics of the hybrid and MPI implementations. We also explore frequency scaling of each implementation of the applications. We use PowerPack [24] to measure the power consumption for our applications. The NAS BT benchmark is an application benchmark included in the NAS Parallel Benchmark suite. The

implementation that we use in this section is a hybrid implementation of the benchmark code written in Fortran and using MPI + OpenMP for communication. The Gyrokinetic Toroidal code (GTC) is a 3D particle-in-cell application developed at the Princeton Plasma Physics Laboratory to study turbulent transport in magnetic fusion. GTC is a flagship SciDAC fusion microturbulence code written in Fortran90, MPI and OpenMP. The Lattice Boltzmann method is widely used for simulating fluid dynamics. Because of the high locality of the collision and streaming operations, domain decomposition is very effective and widely used for the parallel LBM.

Table 5. Overview of HPC Applications

Application	Discipline	Problem Size	Languages
NAS BT-MZ	Computational Fluid Dynamics	Class C	Fortran, MPI/OpenMP
GTC	Magnetic Fusion	100 particles ppc	Fortran90, MPI/OpenMP
PMLB	Computational Fluid Dynamics	128x128x128	C, MPI/OpenMP

3.3.1 NAS BT Benchmark

We use a hybrid NAS parallel benchmark BT with Class B to compare the energy and performance of OpenMP and MPI BT on a single multicore node. Figures 14

and 15 provide the results from using PowerPack to collect power profiles of the CPU, memory, hard disk and motherboard components. Figure 14 indicates that there are slacks where CPUs are waiting for data exchanges among all MPI processes. This causes rapid oscillations in CPU power for the MPI BT. Figure 15 indicates that CPU power for the OpenMP BT does not vary as the MPI BT does because the OpenMP threads take advantage of intra-node communication (shared address space). However, memory power consumptions for both are similar because of the relatively small problem size.

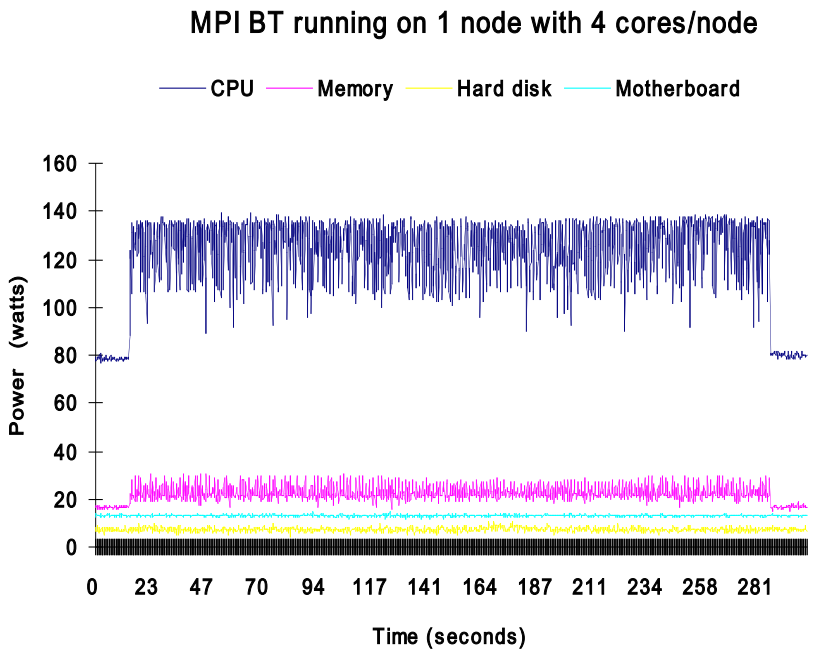

Figure 14. Power for MPI BT Executed on 1 Node

From Figures 14 and 15, we observe that, the performance (execution time) for the OpenMP BT is slightly better than that for its MPI counterpart. The CPU power

consumption for the OpenMP BT is slightly higher than that for its MPI counterpart. The execution time for OpenMP BT is 257 seconds and that for MPI BT is 269 seconds. The total energy consumption for OpenMP BT is 57779J; the energy consumption for MPI BT is 58643J.

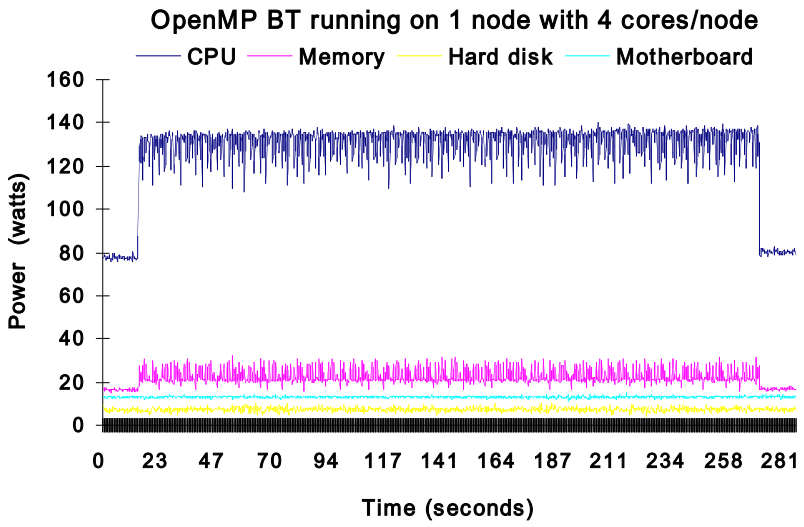

Figure 15. Power for Hybrid BT on 1 Node

Table 6 provides a comparison of the MPI and OpenMP performance of the application on 4 cores using 1 node in the system. Using the OpenMP implementation provides for an overall improvement in execution time of 4.46% and energy savings of 1.47%. The amount of energy consumed by a scientific application is represented by the following equation:

$$E = P_{average} x\, T \tag{6}$$

where $P_{average}$ is the average power consumption of the application and T is the runtime of the application. Based on our energy equation we are able to determine that there is a 4.46% performance improvement and higher power consumption for the OpenMP BT just results in the 1.47% energy saving.

Table 6. Runtime and Energy Comparison for OpenMP and MPI BT on 1 Node

On One Node	Performance	Total Energy
MPI BT	269 s	58,643 J
OpenMP BT	257 s	57,779 J
% improvement	4.46%	1.47%

We used the hybrid NAS parallel benchmark BT with Class B on Dori to evaluate its runtime and power consumptions for MPI and hybrid programming models. Using the hybrid programming achieves better performance and energy. Our results illustrate that there are slacks where CPUs are waiting for data exchanges across all MPI processes. This causes CPU power to oscillate. We observe that CPU power consumption and memory power consumption of the hybrid BT are higher than that for the MPI BT. Memory power consumption for the hybrid BT oscillates significantly compared to that for the MPI BT because of the use of shared address space by OpenMP within a node.

Table 7 provides a comparative analysis of the performance trends of the MPI and hybrid BT as it scales from 4 processors to 32 processors. The results for BT show that as the number of nodes increase, the performance improvement gained from the hybrid implementation increases slightly for both execution time and energy consumption. The results obtained on 16 processors (4x4) illustrate the largest performance difference between the MPI only and hybrid implementation.

Table 7. Energy and Runtime Comparison of MPI and Hybrid BT

#Cores	BT Type	Time(s)	Total Energy (J)	CPU Energy (J)	Memory Energy (J)
1x4	**Hybrid**	257 (-4.46%)	58643 (1.49%)	33410.11 (-5.91%)	5719.43 (-11.41%)
	MPI (Baseline)	269	57779	35508.31	6456.23
2x4	**Hybrid**	122.671 (-13.1%)	28214.1 (-13.1%)	16069 (-11.73%)	2576.29 (-20.24)
	MPI (Baseline)	141.12	32457.6	18204	3230.29
4x4	**Hybrid**	71.723 (-5.84%)	15941.091 (-4.56%)	9453.668 (-5.35%)	1580.718 (-10.48%)
	MPI (Baseline)	76.174	16702.200	9986.521	1765.706
8x4	**Hybrid**	29.9719 (-6.74%)	6728.18 (-5.60%)	3956.29 (-6.25%)	672.84 (-8.55%)
	MPI (Baseline)	31.99	7133.77	4203.53	735.77

When we scale down the CPU frequency from 1.8Ghz to 1.0Ghz, we observe that the hybrid BT has the minimum energy consumption of 14444.036 J with the CPU frequency of 1.2Ghz. We use the energy consumption as a baseline to calculate other percentages shown in Table 3. When increasing the CPU frequency from 1.2 Ghz to 1.8Ghz, we obtained runtime improvement but lost energy. So there is a trade-off between runtime and energy consumption; a decrease in runtime can result in an increase in energy.

3.3.2 Parallel Multiblock Lattice Boltzman (PMLB)

In this section, we discuss the energy performance of a large-scale scientific application, the Parallel Lattice Boltzman (PMLB) [69]. The Lattice Boltzmann method is widely used in simulating fluid dynamics. It is based on kinetic theory that entails a more fundamental level in studying and analyzing the Navier-Stokes equations. The application PMLB was implemented by researchers in the Aerospace Engineering Department at Texas A&M University using MPI for communication. The hybrid implementation of the code incorporates OpenMP to take advantage of the shared-memory architecture of multicore chips.

The PMLB code has properties that demonstrate that the MPI-only implementation provides for a better performance in terms of execution time and energy consumption for up to 16 cores. As the number of cores increases to 32 the execution time and energy consumption for the hybrid version becomes better than the MPI-only version. Specifically, on 32 cores (8x4), the energy consumption for the hybrid implementation is

over 17% better than the MPI only and the execution time for this parallel programming paradigm is 21% better.

Table 8. Energy and Runtime Comparison of MPI and Hybrid PMLB Application

#Cores	PMLB Type	Runtime(s)	Total Energy (KJ)	CPU Energy (KJ)	Memory Energy (KJ)
1x4	Hybrid	30.022 (33.92%)	6.337 (70.81%)	3.682 (65.55%)	0.818 (94.3%)
	MPI (Baseline)	22.418	3.710	2.224	0.421
2x4	Hybrid	21.045 (18.74%)	8.629 (39.42%)	5.246 (40.61%)	0.916 (37.33%)
	MPI (Baseline)	17.724	6.189	3.731	0.667
4x4	Hybrid	13.248 (5.78%)	10.534 (10.55%)	6.276 (12.17%)	1.229 (4.41%)
	MPI (Baseline)	12.524	9.529	5.595	1.177
8x4	Hybrid	11.929 (-21.32%)	17.903 (-17.26%)	10.723 (-16.13%)	2.088 (-17.34%)
	MPI (Baseline)	15.161	21.637	12.784	2.526

The results presented in Table 8 are interesting in two ways. While energy is the product of power and execution time, the percentage reduction or increase for energy was not the same as that for performance. For example, with 4 cores, the execution time for the hybrid implementation was 33% larger, but the corresponding energy was 79%

larger than MPI-only. Second, only when we have 32 cores is the hybrid method better. Further work is needed to explore if a different hybrid implementation would produce better results for 16 or fewer cores.

3.3.3 Gyrokinetic Toroidal Code (GTC)

In this section, we discuss the energy performance of the Gyrokinetic Toroidal Code (GTC). Note that the GTC is weak scaling with 100 particles per cell and 100 time steps. Table 9 provides the energy and performance comparison of the GTC application executed on one node to eight nodes of Dori with the default CPU frequency of 1.8Ghz, where KJ stands for thousand Joules, and NxM means N nodes with M cores per node. With the increase of the number of nodes from 1 to 8, the performance improvement percentage for the hybrid GTC over the MPI-only GTC increases from 37.22% on 1 node to 42.12% on 8 nodes. In addition, the hybrid also saves 37.81% of the overall system energy over the MPI GTC on one node, and 41.86% of the total system energy on 8 nodes. This also shows that using the hybrid MPI/OpenMP programming reduces MPI communication overhead and achieves better performance and save energy.

It is interesting to observe that the performance improvement percentage and energy saving percentage on a given number of nodes (from 1 to 8) are similar mainly because the energy savings are the result of the performance improvement by the hybrid GTC. It indicates that power consumption for both the hybrid GTC and the MPI GTC is similar because the application is weak scaling. This is different from the results of the BT implementation that is shown in Tables 6, where the NAS BT is strong scaling and

the performance improvement percentage for the hybrid BT is much larger than its energy saving percentage because of the higher power consumptions of the hybrid BT.

Table 9. Energy and Runtime Comparison of MPI and Hybrid GTC Application

#Cores	GTC Type	Runtime(s)	Total Energy (KJ)	CPU Energy (KJ)	Memory Energy (KJ)
1x4	Hybrid	1302.773 (-37.22%)	270.223 (-37.81%)	162.969 (-38.52%)	27.086 (-33.47%)
	MPI (baseline)	2075.376	434.524	265.071	40.714
2x4	Hybrid	1395.322 (-37.47%)	576.674 (-37.68%)	353.826 (-38.35%)	61.887 (-34.33%)
	MPI (baseline)	2231.652	925.401	574.003	94.238
4x4	Hybrid	1434.491 (-38.29%)	1182.959 (-38.40%)	711.065 (-39.31%)	118.186 (-34.64%)
	MPI (baseline)	2324.707	1920.578	1171.572	180.825
8x4	Hybrid	1463.457 (-42.12%)	2419.985 (-41.86%)	1457.945 (-42.39%)	244.013 (-37.73%)
	MPI (baseline)	2528.556	4162.998	2530.861	391.842

3.3.4 Runtime & Energy Using Frequency Scaling

To perform frequency scaling on Dori, five frequency values are utilized. The default frequency and voltage for the system is set to 1.8 Ghz and 1.4V and can be

adjusted to 1.0Ghz and 1.3V. The CPU frequency on Dori can be adjusted in increments of 200 hz from 1.8Ghz to 1.0Ghz. The power profiling data of BT and GTC executed on 4 nodes (4x4) is used to further investigate the impact on energy and runtime from applying frequency scaling to execution of the applications.

Table 10 provides the effects of applying frequency scaling to the NAS BT benchmark. When we scale down the CPU frequency from 1.8Ghz to 1.0Ghz we observe that the hybrid BT has the minimum energy consumption of 14444.036 J with the CPU frequency of 1.2Ghz. We use the runtime and energy consumption at 1.2Ghz as a baseline to calculate other percentages shown in Table 4. When increasing the CPU frequency from 1.2 Ghz to 1.8Ghz, we obtained runtime reduction but increased energy. So there is a trade-off between performance and energy consumption. Achieving a reduction in runtime may require using more energy.

Table 10. MPI and Hybrid BT on 4x4 (16 Cores) Using Frequency Scaling

CPU Speed	BT Type	Runtime(s)	Total Energy (KJ)	CPU Energy (KJ)	Memory Energy (KJ)
1.8Ghz	Hybrid	71.723 (-25.31%)	15941.091 (10.36%)	9453.668 (22.88%)	1580.718 (-26.76%)
	MPI	76.174 (-27.82%)	16702.200 (15.63%)	9986.521 (29.8%)	1765.706 (-18.19%)
1.6Ghz	Hybrid	76.139 (-21.80%)	15058.230 (4.25%)	8737.304 (13.57%)	1713.132 (-20.62%)
	MPI	81.841 (-15.94%)	15903.052 (10.1%)	9088.220 (18.13%)	1858.386 (-13.9%)
1.4Ghz	Hybrid	84.849 (-12.86%)	14732.076 (1.99%)	8186.828 (6.41%)	1852.877 (-14.15%)
	MPI	90.530 (-7.02%)	15624.080 (8.17%)	8577.551 (11.49%)	1992.754 (-7.67%)
1.2Ghz	Hybrid (BASELINE)	**97.366**	**14444.036**	**7693.547**	**2158.369**
	MPI	101.990 (4.74%)	15088.793 (4.46%)	8101.081 (5.29%)	2330.107 (7.96%)
1.0Ghz	Hybrid	111.947 (14.97%)	17041.246 (17.98%)	9325.778 (21.22%)	2480.800 (14.93%)
	MPI	117.394 (20.56%)	17774.750 (23.06%)	9630.939 (25.18%)	2606.256 (20.75%)

Table 11 shows the energy and runtime for the hybrid and MPI-only GTC at five CPU frequency gears from 1.8GHz to 1.0GHz on the Dori system. This shows the effect that adjusting the frequency of the system has on the energy and performance of the application. As shown in Table 5, for the default CPU frequency of 1.8 GHz, the performance improvement percentage for the hybrid GTC over the MPI-only GTC is 38.29% on 4 nodes (with 4 cores per node), and the hybrid also saves 38.40% of the overall system energy over the MPI GTC on 4 nodes. We use the energy and runtime for the MPI and hybrid GTC at the CPU frequency of 1.6GHz as baseline to calculate the percentages of energy and runtime at various frequencies in Table 6. As we seek to explore the saving in energy we use the lowest energy consumption obtained at 1.6 Ghz as the baseline.

For the given problem size and number of cores, it is obvious to see the total application execution times for both the MPI and hybrid GTC increase with decreasing the CPU frequency from 1.8 GHz to 1.0GHz as shown in Table 5. For instance, the execution time for the hybrid GTC executed on 4 nodes increases up to 37.87% when decreasing the CPU frequency to 1.0GHz. Decreasing CPU frequency means that a lower voltage is utilized. This results in lower power consumption for the application.

From Table 11, we observe that the total energy consumption for the hybrid GTC decreases 3.78% for the frequency of 1.6GHz, less than 1% for the frequency of 1.4GHz, increases 1.77% for the frequency of 1.2GHz, but increases 17.81% for the frequency of 1.0GHz. So there is performance-energy trade-off that needs to be seriously considered when applying frequency scaling to an application.

Table 11. GTC Power Profiling on 4x4 (16 Cores) Using Frequency Scaling

CPU Speed	GTC Type	Runtime(s)	Total Energy (KJ)	CPU Energy (KJ)	Memory Energy (KJ)
1.8Ghz	Hybrid	1434.491 (-8.62)	1182.959 (3.72%)	711.065 (-7.1%)	118.186 (-8.09%)
	MPI	2324.707 (48.1%)	1920.578 (68.50%)	1171.572 (76.42%)	180.825 (40.62%)
1.6Ghz	Hybrid (BASELINE)	1569.960	*1139.831*	664.098	128.594
	MPI	2511.532 (59.97%)	2057.516 (80.51%)	1253.041 (88.68%)	196.440 (52.76%)
1.4Ghz	Hybrid	1773.444 (12.96%)	1143.615 (0.03%)	661.161 (-0.04%)	153.450 (19.39)
	MPI	2791.607 (77.81%)	1778.682 (56%)	1040.457 (56.67%)	230.353 (79.13%)
1.2Ghz	Hybrid	2094.598 (33.40%)	1162.393 (1.97%)	628.386 (-5.37%)	176.897 (37.56%)
	MPI	3126.446 (99.1%)	1724.057 (51.26%)	940.227 (41.57%)	254.275 (97.73%)
1.0Ghz	Hybrid	2445.155 (37.87%)	1393.650 (22.26%)	769.366 (15.85%)	204.96 (4.34%)
	MPI	3553.982 (127.37%)	2015.483 (76.82%)	1112.277 (67.49%)	285.326 (121%)

Table 12 illustrates the effect that frequency scaling has on the performance of GTC at a functional granularity. The runtime for GTC at the default frequency of 1.8GHz are used as baselines to calculate the performance percentages for reduced frequencies for hybrid and MPI implementations. Therefore, the hybrid percentages are shown compared to the baseline of 1.8 GHz for the hybrid implementation. Also, the

MPI implementation is compared to the baseline of the MPI application at 1.8 Ghz. We observe that the hybrid GTC outperforms its MPI counterpart because of big performance improvements for five functions shift, charge, poisson, smooth and field of the GTC and poor L2 cache behavior for the MPI implementation which increases the amount of off-chip communications and degrades the performance. This is consistent across different CPU frequencies. This further shows that using the hybrid MPI/OpenMP programming can not only reduce MPI communication overhead but also achieve better performance and save energy. The function-level information for CPU frequency scaling can help us aid in finding the best combination of CPU frequency adjustments for the entire GTC to save more energy when applying frequency scaling to the entire application.

Table 12. Function Comparison of GTC using Frequency Scaling

CPU Speed	GTC Type	Time(s)	Pusher	Shift	Charge	Poisson	Smooth
1.8Ghz	Hybrid (Baseline)	1434.49	854.5	36.75	498.1	16.94	10.25
	MPI	2324.70 (62.1%)	823.7 (-3.63%)	268.4 (630%)	627.7 (26%)	159.6 (842%)	284.9 (2680%)
1.6Ghz	Hybrid	1569.96 (-9.4%)	940.1 (-10.0%)	39.64 (-7.83%)	542.3 (-8.87%)	20.81 (-22.84%)	8.019 (-21.76)
	MPI	2511.53 (75.1%)	850.7 (-0.44%)	348.2 ((847%)	692.5 (39%)	160.8 (849%)	292.7 (2756%)

Table 12: continued

1.4Ghz	**Hybrid**	1773.44 (-23.6%)	1067.00 (-24.9%)	38.48 (-4.8%)	617.2 (-23.91%)	21.01 (-24.03%)	8.532 (-16.76%)
	MPI	2791.60 (95%)	1053.00 (23.33%)	307.40 (736%)	772.6 (55.1%)	157.8 (832%)	323.7 (3058%)
1.2Ghz	**Hybrid**	2094.598 (-46.0%)	1255.00 (-46.9%)	46.43 (-26.34%)	734.2 (-47.4%)	24.21 (-42.91%)	9.884 (-3.7%)
	MPI	3126.446 (117%)	1218.00 (42.5%)	324.3 (782%	897.9 (80.2%)	162.80 (861%)	335.00 (3168%)
1.0Ghz	**Hybrid**	2445.16 (-70.5%)	1473.00 (-72.4%)	48.18 (-31.10%)	855.9 (-71.83%)	28.60 (-68.83%)	11.21 (-9.36%)
	MPI	3553.982 (148%)	1458.00 (70.62%)	339.0 (822%)	1056.00 (112%)	161.3 (852%)	345.9 (3275%)

3.4 Summary

In this section we investigated energy and performance characteristics of different parallel implementations of scientific applications on multicore systems, and explored interactions between power consumption and application performance. We used the power profiling tool PowerPack to collect power profiling data for four scientific applications: a hybrid NAS parallel BT benchmark, a hybrid Lattice Boltzmann application PMLB and a hybrid Gyrokinetic Toroidal Code for our comparative analysis of energy and performance on multicore clusters. Our experimental results show that there are various ways to save energy and improve performance of parallel application codes.

First, we found, with respect to the MPI-only versus the hybrid implementation for a scientific application, the best implementation can be dependent upon the application executed on 16 or fewer cores. For the case of 32 cores, the results were consistent that hybrid resulted in less execution time and energy on the Dori platform. For example, the hybrid PMLB achieved 21% performance improvement and 17% reduction in energy consumption compared to the MPI only implementation. With the CPU Frequency Scaling, the best case for energy saving was not the best case for execution time. For example, the hybrid GTC executed at the CPU frequency 1.6 Ghz provided the lowest energy consumption but the execution time increased by 8.62%.

The hybrid implementations are based on the existing MPI applications and were implemented to exploit the shared-memory architectures of multicore systems. The results from these experiments show that the energy consumption and runtime of scientific applications can vary at different frequencies. Therefore, these experiments provide a strong foundation for the development of performance models that can be used to analyze the runtime and power consumption of hybrid and MPI applications. Furthermore, accurate power-aware models can be used to better determine appropriate ways to optimize or refine scientific applications to reduce runtime and reduce power consumption.

4. POWER-AWARE PERFORMANCE MODELS OF SCIENTIFIC APPLICATIONS

In this section, we present the E-AMOM methodology for developing power-aware predictive models of Hybrid and MPI scientific applications. This section provides the detailed predictive models for six scientific applications. The E-AMOM experimental models identify the characteristics in the applications that affect each performance component (runtime, system power, CPU power, and memory power).

4.1 Performance-Tuned Principal Component Analysis Methodology

The performance-tuned principal component analysis method included in E-AMOM was developed in order to determine a subset of predictors (via performance counters) that could be used in modeling the performance of scientific applications. The performance-tuned principal component analysis method combines statistically sound concepts into a schema to determine performance counters that can be associated with an application's outcome, such as execution time and component power consumption of the system, CPU, and memory. The remainder of this section will describe the components of this method and how it is applied to scientific applications.

4.1.1 Modeling via Performance Counters

The most difficult part in predicting application performance via performance counters is determining the appropriate counters to use for modeling each component.

We develop the performance-tuned principal component analysis method for modeling the performance components of Hybrid and MPI scientific applications. Using this method, we identify the application characteristics (via performance counters) that affect performance of the application. The performance characteristics that are modeled include runtime, system power consumption, CPU power consumption, and memory power consumption.

The following algorithm was used to identify the performance counter events needed to build the predictive models for each application:

1. Compute the Spearman's rank correlation for each performance counter event rate for each performance component (runtime, system power, CPU power, and memory power).

Equation (7) defines how the spearman correlation coefficient is computed for identifying the rank between each performance counter and each performance component. The relationship between two variables x and y is determined based on this formula. The x_i and y_i variables represent the ranks of the performance counter and performance component (time, system power, CPU power, memory power) that are being correlated. The variables $\bar{x}$ and $\bar{y}$ represent the average of the samples for each variable. The spearman rank correlation will provide a value between -1 and 1 for ρ. If

ρ is -1 then the variables are not correlated in any way, if ρ is 1 then the values are highly correlated and as one increases the other value increases as well.

$$\rho = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2 \sum_i (y_i - \bar{y})^2}} \quad (7)$$

The spearman's rank correlation coefficient is used because it provides a correlation value that is not easily influenced by outliers in the dataset. The pearson correlation coefficient uses the actual values for the correlation and therefore it is more influenced by outliers in the dataset. Figure 16 illustrates the effects that outliers have on different correlation values. This figure shows corresponding values collected with X being the performance counters and Y being runtime. As shown in the figure, the correlation coefficient for the spearman rank is larger than the pearson correlation because pearson correlation is affected by the outlier values.

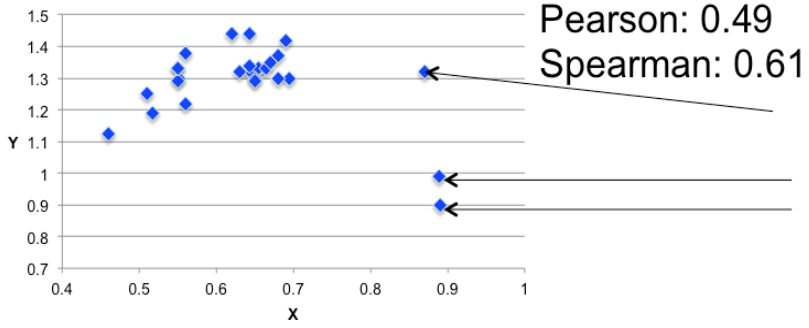

Figure 16. Spearman and Pearson Correlation Comparison

2. Establish a threshold, β_{ai}, to be used to eliminate any counters with Spearman rank correlation values below the threshold.

β_{ai} is used to determine an appropriate threshold for eliminating performance counters with low correlation to the performance event that is to be modeled. For example, if $\beta_{ai} = 0.59$ then all performance counters with correlation coefficients below this value are eliminated from the performance counter group. The value for β_{ai} is established based on a cluster analysis of the data. When more than half of the values from the spearman rank correlation analysis are found above a point, the remaining values are eliminated at the threshold β_{ai}. The values in red in Table 13 illustrate values that have been reduced based on the threshold β_{ai}. The β_{ai} threshold is used to reduce the number of performance counters that must be used in developing our regression model. A reduced number of performance counters improves the time that it takes in developing the regression model and also simplifies the model for future use.

Table 13. Reduced Performance Counters and Correlation Values

Performance Counter	**Correlation Value**
PAPI_TOT_INS	0.9187018
PAPI_FP_OPS	0.9105984
PAPI_L1_TCA	0.9017512
PAPI_L1_DCM	0.8718455
PAPI_L2_TCH	0.8123510
PAPI_L2_TCA	0.8021892
Cache_FLD	0.7511682

Table 13: continued

Performance Counter	Correlation Value
PAPI_TLB_DM	0.6218268
PAPI_L1_ICA	0.5487321
Bytes_out	0.5187535
PAPI_L1_ICA	0.4876423
PAPI_L1_ICM	0.4449848
PAPI_L2_ICM	0.4017515
PAPI_CA_SHARE	0.3718456
PAPI_HW_INT	0.3813516
PAPI_CA_ITV	0.3421896

3. Compute a multivariate linear regression model based upon the remaining performance counter event rates.

Equation (8) outlines the multivariate linear regression model that is used for creating an initial application model based upon the remaining application performance counters. The multivariate linear regression model is based upon the performance

counters that have not been eliminated thus far in the performance-tuned principal component analysis process. Table 14 provides the derived regression coefficients that could be used in modeling the runtime of a sample application code.

$$y = \beta_0 + \beta_1 * r_1 + \cdots \beta_n * r_n \quad (8)$$

Table 14. Reduced Performance Counters and Regression Coefficients

Performance Counter	Regression Coefficients
PAPI_TOT_INS	1.984986
PAPI_FP_OPS	1.498156
PAPI_L1_DCM	0.9017512
PAPI_L1_TCA	0.465165
PAPI_L2_TCA	0.0989485
PAPI_L2_TCH	0.0324981
Cache_FLD	0.026154
PAPI_TLB_DM	0.0000268
PAPI_L1_ICA	0.0000021
Bytes_out	0.000009

4. Establish threshold, β_{bi}, and eliminate performance counters and ensure that the regression coefficients are not greater than β_{bi} in terms of magnitude.

The value β_{bi} serves as the second elimination threshold and serves a similar purpose as β_{ai} to eliminate performance counters that do not contribute substantially to modeling in the initial multivariate linear regression model. The determination of β_{bi} is accomplished by analyzing the coefficients of the multivariate linear regression model. A visual analysis is used to determine an appropriate threshold that should be used to eliminate the values that are negligible. An appropriate value for β_{bi} is important because if the value is not correctly chosen, or is too high, then values needed in the modeling will be eliminated. Table 15 provides an illustration of applying the β_{bi} threshold to eliminate counters that are below the elimination value.

Table 15. Reduced Performance Counters and Regression Coefficients-Step 5

Counter	Regression Coefficient
PAPI_TOT_INS	1.984986
PAPI_FP_OPS	1.498156
PAPI_L1_DCM	0.9017512
PAPI_L1_TCA	0.465165
PAPI_L2_TCA	0.0989485
PAPI_L2_TCH	0.0324981
Cache_FLD	0.026154

5. Compute the principal components of the reduced performance counter event rates.

Principal component analysis is a statistical method that is used to provide a linear transformation of data that can be used to reduce the dimensionality of data. The principal components of data, (Y_i), is given by a linear combination of variables X_i, X_2, …..X_p. For example, the first principal components would be represented by equations (9)

$$Y_1 = a_{11}X_1 + a_{12}X_2 + \cdots \ldots . . + a_{1p}X_p \quad (9)$$

The values for a_{i1}, a_{i2}, ……a_{ip} are calculated with the constraint that the sum of their squares must equal to 1.

$$a_{11}^2 + a_{12}^2 + \cdots \ldots . . + a_{1p}^2 = 1 \quad (10)$$

The second principal component would be calculated in the same manner as the first principal component following the condition that it must be uncorrelated (perpendicular) to the first principal component.

$$Y_2 = a_{21}X_1 + a_{22}X_2 + \cdots \ldots . . + a_{2p}X_p \quad (11)$$

The number of principal components calculated is dependent upon the number of variables included in the original data. In our work, the number of principal components would be equal to the number of performance counters reduced for modeling the performance component of the application. Figure 17 provides an illustration of determining the points through the data that would compose Principal Component 1 and Principal Component 2. PCA is used to identify the final performance counters to use

because PCA is a statistically proven technique for identifying. We use these values as guidelines for determining which performance counters to use in our final model.

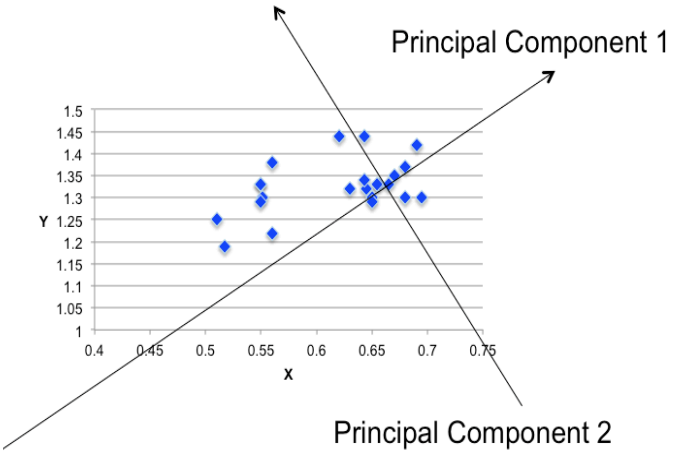

Figure 17. PCA Analysis Example

6. Use the performance counter event rates with the highest principal component coefficient vectors to build a multivariate linear regression model to predict the respective performance metric.

The first two principal components represent the largest amount of variability, or information, in our data. Therefore, the first two principal components are used for further reducing the number of variables for creating our multivariate linear regression model. The variables with the highest coefficients, or weights, contribute the most to variability of the data and therefore serve as the most accurate predictors for modeling. Table 16 represents the final multivariate regression model that is developed using the performance-tuned principal component analysis method in E-AMOM. Our final model

reduced the number of performance counters to 4 to represent the execution time of the application from the original 10 counters in the initial regression (step 3).

Table 16. Final Multivariate Regression Model

Counter	Regression Coefficient
PAPI_TOT_INS	2.59653
PAPI_L1_TCA	1.19494
PAPI_L2_TCA	0.92378
PAPI_L2_TCH	1.138384

We explore the following issues in regards to modeling the runtime and power consumption of MPI and Hybrid scientific applications on multicore systems:

a) What is the accuracy of our models in estimating performance (runtime and power consumption for system, CPU, and memory components?
b) Which combination of performance counters can be used to model the Hybrid application in terms of runtime, system power, CPU power, and memory power?
c) What are the application characteristics that affect runtime and power consumption in Hybrid scientific applications?
d) What characteristics of Hybrid applications can be refined to improve performance on multicore systems?

Our application-models explore the following dimensions in terms of modeling and predicting application performance:

i. Difference application data sizes
ii. Number of processors
iii. Different frequency settings
iv. Different concurrency settings

4.1.1.1 Modeling Summary

Overall, the modeling component of E-AMOM includes six detailed steps that are used to construct our performance models. Our performance-tuned principal component analysis method is applied to each application for each performance component. In step 1, the correlation coefficients are used to determine which performance counters are highly correlated with the performance component being modeled. A high correlation is strong indication that the value will be accurate in modeling the performance component. In step 2, a threshold is used to reduce the number of counters that would be used in the final regression step. Step 3 and Step 4 provide additional measures that are used to reduce the number of performance counters that would be included in our model. Steps 5 applies principal component analysis to our reduced model set of counters to determine which of the remaining counters capture the strongest overall structure of our data. The final step builds the model that is used for the performance component of the application.

4.1.2 Training Set Criteria

We develop predictive performance models for the Hybrid and MPI applications in this work. Our predictive models are developed using various configuration points for each application and each implementation to constitute a training set for predicting performance in terms of runtime and power consumption of the system, CPU, and memory components. A 6 points of the configuration space are used for predicting intra-node performance and 6 points are used for predicting inter-node performance.

Scaling processor performance for each implementation is represented as (M x N), where M is the number of nodes used, and N is the number of processors per node. In the case of MPI-only, M is the number of nodes and N is MPI processes per node; each MPI process is mapped to a processor. In the case of our hybrid application M is the number of nodes and N is the number of OpenMP threads utilized per node; each OpenMP thread is mapped to a processor. For a Hybrid application, a configuration of 11x8 means that 11 MPI processes with 8 OpenMP threads per node were used for a total of 128 processors. The training set for inter-node performance uses six data points consisting of configurations for points at 1x8, 3x8, 5x8, 7x8, and 9x8, 10x8. The 12 training points were used to predict performance for up to 16 larger configuration points to determine how accurate our model was at predictions for larger processor sizes.

4.2 HPC Applications

In this section, we present the scientific applications that are used throughout this section. These applications include the three NAS Multi-Zone Parallel Benchmarks [35]

and three large-scale scientific applications. We model the runtime, system, CPU, and memory power consumption of Hybrid and MPI implementations of scientific applications on multicore systems. Table 17 provides an overview of the applications that are used in this modeling work.

The NPB-MZ suite contains three benchmarks (BT-MZ, SP-MZ, and LU-MZ), which are used to simulate the performance characteristics of large-scale scientific applications. Each benchmark in NPB-MZ consists of using a main loop to exchange values during MPI communication and an OpenMP phase within the loop. Each benchmark contains a number of performance inputs, which enable the performance of the application to be tested for different workload sizes.

Table 17. Overview of HPC Applications

Application	Discipline	Problem Size	Languages
NAS BT-MZ	Computational Fluid Dynamics	Class B Class C Class D	Fortran, MPI/OpenMP
NAS SP-MZ	Computational Fluid Dynamics	Class B Class C Class D	Fortran, MPI/OpenMP

Table 17: continued

Application	Discipline	Problem Size	Languages
NAS LU-MZ	Computational Fluid Dynamics	Class B Class C	Fortran, MPI/OpenMP
GTC	Magnetic Fusion	50 particles ppc 75 particles ppc 100 particles ppc	Fortran90, MPI/OpenMP
PMLB	Computational Fluid Dynamics	64x64x64 128x128x128 256x256x256	C, MPI/OpenMP
Parallel EQdyna	Earthquake Simulation	200m	Fortran90, MPI/OpenMP

The Block Tri-diagonal algorithm (BT-MZ) contains (16x16) x-zones x y-zones and has uneven mesh tilings. BT-MZ represents realistic performance case for exploring the discretization meshes in parallel computing. The Scalar Penta-diagonal algorithm (SP-MZ) contains (16x16) x-zones x y-zones and is representative of a balanced workload in the suite. The Lower-Upper symmetric Gauss-Seidel algorithm (LU-MZ) contains (4x4) x-zones x y-zones and the coarse-grain parallelism of LU-MZ is limited to 16. Therefore, at most 16 MPI processes can be used in executing LU-MZ. The

problem sizes for all NPB-MZ benchmarks are strong scaling using class C, utilizing 800MB of memory.

The GTC application was previously introduced in Section 3. There are 7 major functions: load, field, smooth, poisson, charge, shift and pusher in the code. Charge, pusher and shift dominate most of the runtime. Note that GTC is executed in weak scaling to keep a constant workload per processor as the number of processors increase using 100 particles per cell and 100 time steps.

The PMLB application was previously discussed in Section 3. In this section we will provide additional details about the application. In the parallel implementation, the entire computational domain is divided into $n_1 \times n_2 \times n_3$ blocks, where n_1, n_2, n_3 are the number of segments in the x-, y-, and z-dimensions. Each block is assigned to a processor. The grid sizes in each block can be different. The PMLB application code is written in C with MPI, and is divided into six kernels:

- Initialization: reads input files and sets up the initial parameter values.
- Collision: computes the effect of the collisions, which occur during the particle movement.
- Communication: communicates the needed data among neighboring blocks.
- Streaming: moves particles in motion to new locations along their respective velocities.
- Physical: calculates macroscopic variables such as fluid density, which are used in the collision and streaming steps.
- Finalization: cleans up the program and outputs the results.

In the PMLB application, the Collision, Communication, Streaming, and Physical kernels are executed in a loop for 200 iterations. The Initialization and Finalization kernels are executed once for the entire application run.

The Parallel EQdyna application is used to illustrate an element-based partitioning scheme for explicit finite element methods. The application efficiently uses hybrid MPI/OpenMP to parallelize the sequential, explicit finite element earthquake rupture simulation code Parallel EQdyna. This enables the application to achieve multiple levels of parallelism and also reduce the communication overhead of MPI within a multicore node. The Hybrid EQdyna application is based on what the OpenMP implementation of EQdyna.

4.3 Experimental Results

We present the experimental results for our power-aware predictive models of hybrid and MPI scientific applications. We use our method to predict the runtime, system power, CPU power, and memory power of scientific applications.

4.3.1 BT-MZ

4.3.1.1 Hybrid

We use our performance-tuned principal component analysis method to develop accurate models for the Hybrid implementation of the NAS BT-MZ benchmark application. The components that we utilize to model our application include runtime,

system power, CPU power, and memory power. Table 18 shows the regression coefficients that are needed to accurately model each component. For a detailed explanation of what each performance counter means, please refer to Table 2 in Section 2. The frequency component is included in our regression table because it is used in predicting the performance of each application for different frequencies. With regards to runtime, PAPI_L2_TCM has the largest regression coefficient that is used in modeling the BT-MZ Hybrid application. Modeling of the system power consumption results in the PAPI_L2_TCA having the largest regression for the application.

Table 18. Regression Coefficients for BT-MZ Hybrid

Time		System Power		CPU Power		Memory Power	
Frequency	0.00476	**Frequency**	0.00125	**Frequency**	0.00112	**Frequency**	0.00012
PAPI_TOT_INS	0.105050	**PAPI_L2_TCH**	0.01584	**PAPI_L1_TCM**	0.3551	**PAPI_L1_TCA**	0.0310818
PAPI_L2_TCM	0.178700	**PAPI_L2_TCA**	0.07793	**PAPI_L2_TCH**	0.03187	**PAPI_L2_TCM**	0.45754
PAPI_L2_TCA	0.097108	**PAPI_RES_STL**	0.05803	**PAPI_RES_STL**	0.128897	**PAPI_L2_TCH**	0.0018475
						PAPI_BR_INS	0.0001894

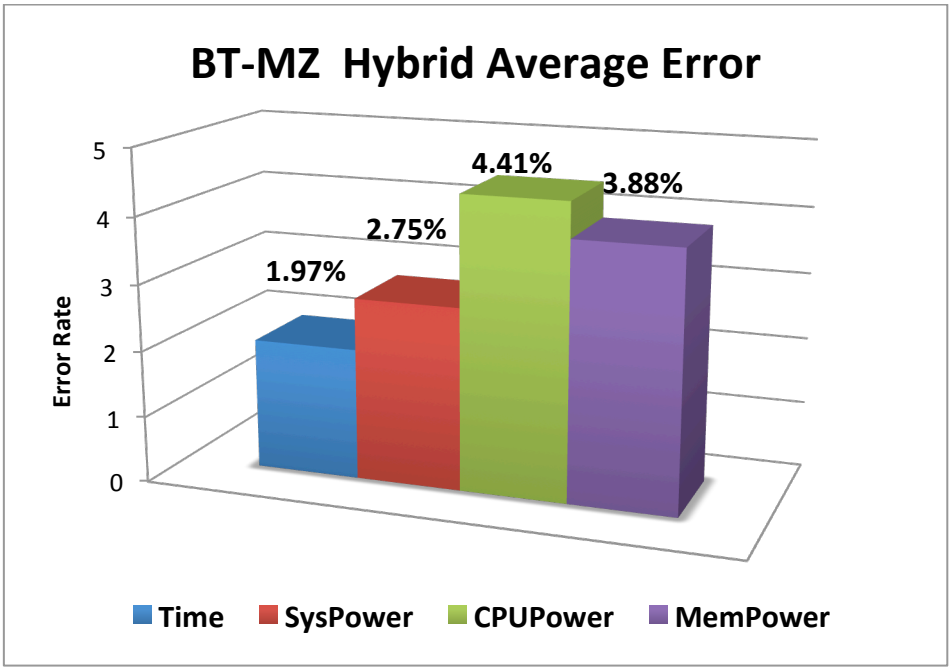

Figure 18. Average Error of BT-MZ Hybrid

With regards to modeling of the CPU power consumption, the PAPI_L1_TCM counter has the largest regression coefficient in comparison to the other counters and components. The PAPI_RES_STL counter has the 2nd largest regression coefficient in modeling the CPU power consumption. The memory power consumption is modeled using PAPI_L1_TCA, PAPI_L2_TCM, PAPI_L2_TCH, and PAPI_BR_INS. PAPI_L2_TCM has the largest regression coefficient for modeling memory power.

Figure 18 shows the average error for predicting the performance and power consumption of the BT-MZ Hybrid application. The performance components modeled in the BT-MZ Hybrid application had an average prediction error of less than 5%. Specifically, the smallest error was found for predicting the runtime for the application

across 40 prediction points, less than 2%. The highest error rate occurred in modeling the CPU power consumption (4.41%).

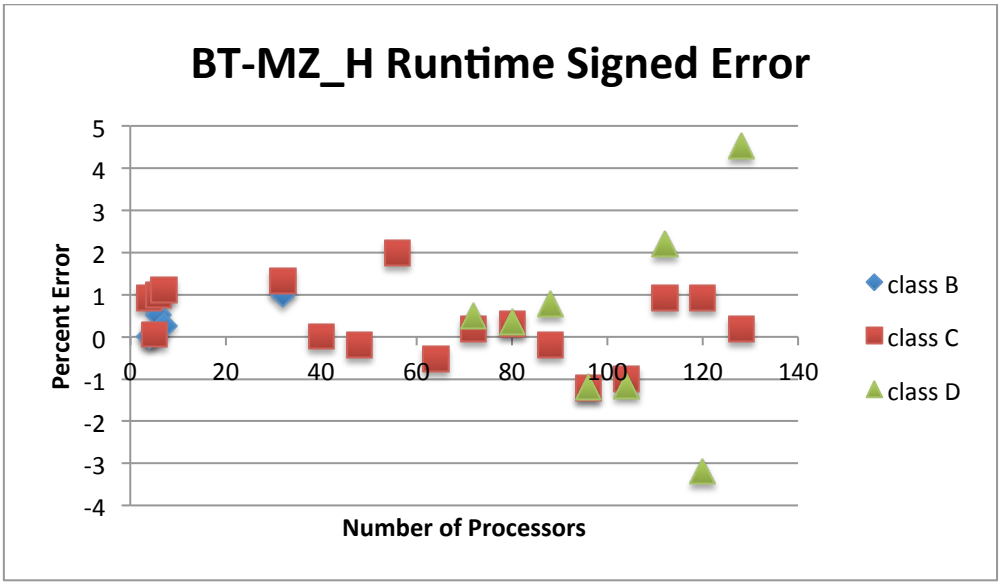

Figure 19. Scatterplot of BT-MZ Hybrid for Runtime

In Figure 19 we provide an overview of the distribution of the percent error for the runtime of BT-MZ Hybrid application. The values show that predicting the runtime the percent error is largely positive for approximately 80% of the points. As the number of processors increases, the error increases for class D only.

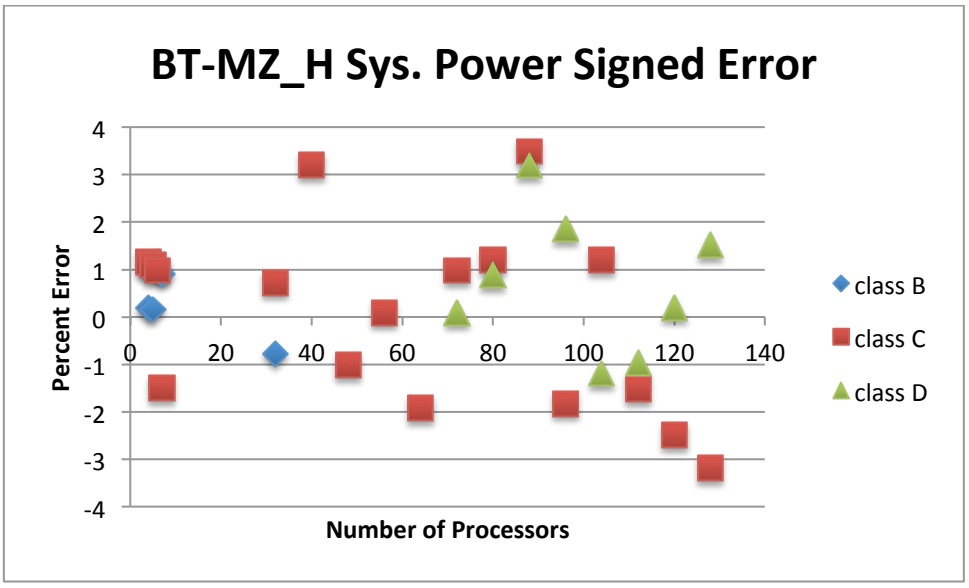

Figure 20. Scatterplot of BT-MZ Hybrid for System Power Consumption

In Figure 20 we provide an overview of the distribution of the percentage error for the system power of the BT-MZ Hybrid application. The figure illustrates that a large number of the points are over-predicted, resulting in positive percent errors, for class B and class D. When the number of processors is greater than 80, the model underestimates the power consumption. This could be caused from the modeling not being able to take into account the scaling of the workload for the class C benchmark at the larger processor configuration.

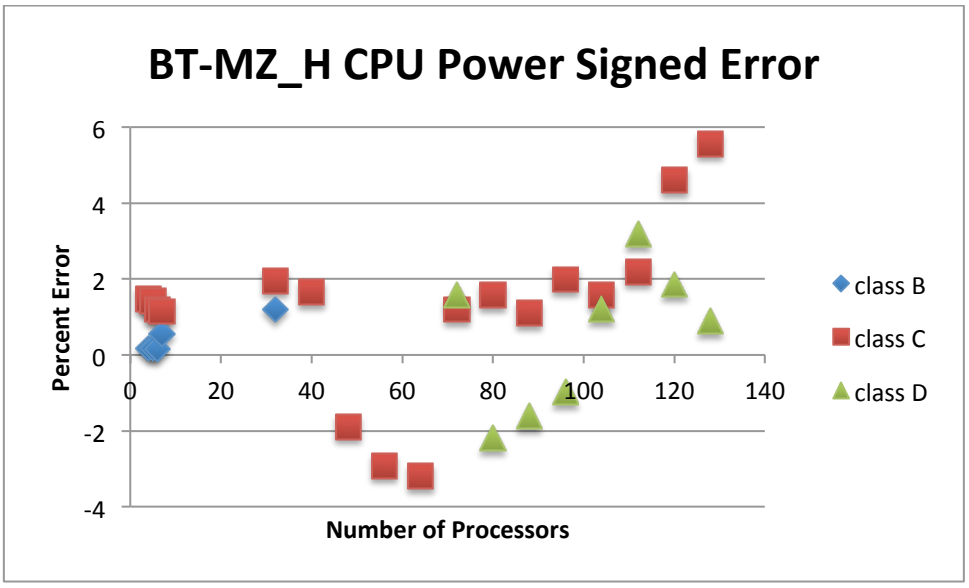

Figure 21. Scatterplot of BT-MZ Hybrid for CPU Power Consumption

In Figure 21 we provide an overview of the distribution of each prediction point for the CPU power consumption of the BT-MZ Hybrid application. The figure illustrates that a large number of the points are over-predicted or have positive percent errors across all problem sizes. For the case of class C and class D there is a period between 48 and 96 processes in which our model under-estimates the CPU power consumption for the application. In the case of class B, it should be noted that the model provides for all positive estimations of the CPU power consumption.

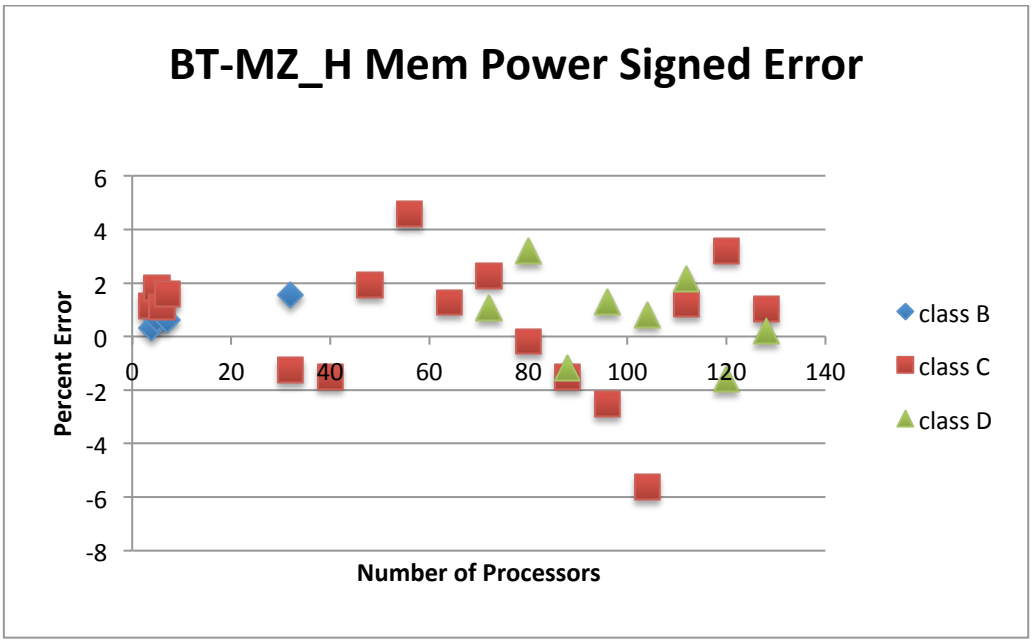

Figure 22. Scatterplot of BT-MZ Hybrid for Memory Power Consumption

In Figure 22 we provide an overview of the distribution of each prediction point for the memory power consumption of BT-MZ Hybrid application. The figure illustrates that a large number of the points are over-predicted or positive for class B and class D. For larger number of processes, greater than 80 processes, the system power prediction underestimates the actual value. This could be caused from the model not being able to take into account the scaling of the workload for the class C benchmark.

4.3.1.2 MPI

In this section, we use our performance-tuned principal component analysis method to develop accurate models for the MPI implementation of the NAS BT-MZ benchmark application. Table 19 shows the regression coefficients that are used to

model each component. With regards to time, PAPI_L2_TCA has the largest regression coefficient that is used in modeling the BT-MZ Hybrid application. System power shows that PAPI_TOT_INS has the largest regression coefficient in model.

Table 19. Regression Coefficients for BT-MZ MPI

Time		System Power		CPU Power		Memory Power	
Frequency	0.001358	**Frequency**	0.00329	**Frequency**	0.021876	**Frequency**	0.00918984
PAPI_TOT_INS	0.0398942	**PAPI_TOT_INS**	0.01694	**PAPI_TOT_INS**	0.041895	**PAPI_L1_TCA**	0.047188359
PAPI_L2_TCH	0.01584	**PAPI_L2_TCH**	0.00157	**PAPI_L1_TCM**	0.003182	**PAPI_L2_TCH**	0.002761895
PAPI_L2_TCA	0.0431898	**PAPI_L2_TCA**	0.00132	**PAPI_L2_TCH**	0.171658	**PAPI_L2_TCA**	0.0817945
PAPI_RES_STL	0.0098415	**PAPI_RES_STL**	0.00182	**PAPI_L2_TCA**	0.918457		

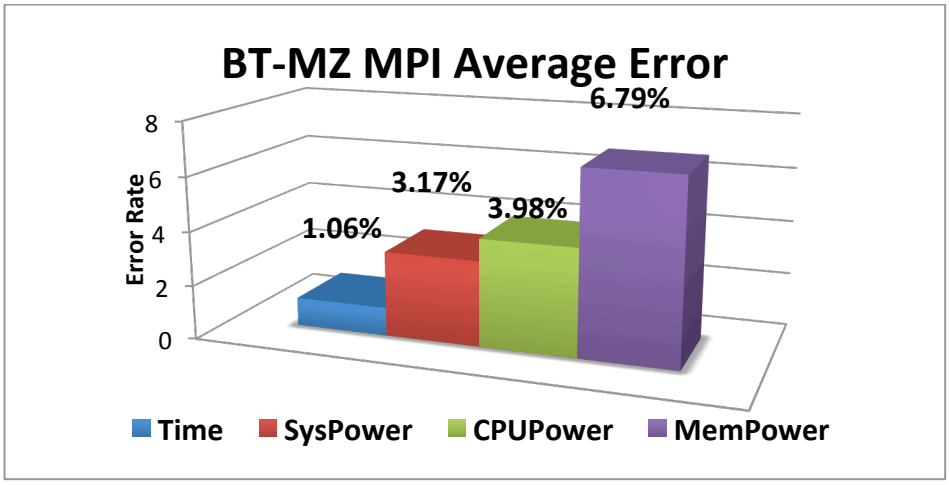

Figure 23. Average Error of BT-MZ MPI

With regards to modeling of the CPU power consumption, the PAPI_L2_TCA counter has the largest regression coefficient in comparison to the other counters and

components. The PAPI_TOT_INS counter has the 2nd largest regression coefficient in modeling the CPU power consumption. The memory power consumption is modeled using PAPI_L1_TCA, PAPI_L2_TCH, and PAPI_L2_TCA. PAPI_L2_TCA has the largest regression coefficient for modeling memory power.

Figure 23 shows the average error resulting from the modeling of the BT-MZ MPI application. The performance components modeled for the BT-MZ MPI application had an average prediction error of less than 3.8%. The smallest error is for predicting the runtime for the application across 40 prediction points, less than 1.1%. The highest error rate occurred in modeling the memory power consumption (6.79%).

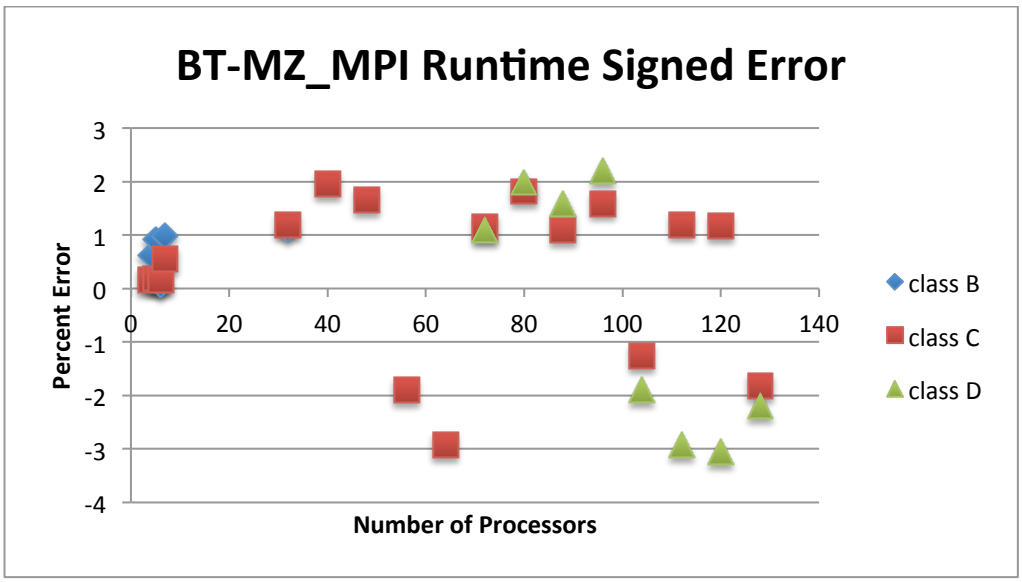

Figure 24. Scatterplot of BT-MZ MPI for Runtime

Figure 24 and Figure 25 present a detailed overview of the predictions for the BT-MZ MPI application when predicting application runtime and system power

consumption. Overall, our modeling methodology provided predictions in the range of +2% to -3% for percent error across three input datasets. The smaller processor configurations show percent errors of less than 1% across class B and class C. There are negative percent errors for class C and class D as the number of processors increase. These under-predictions are likely caused by the training set for this application not providing the best fit for predicting at those specific processor configurations.

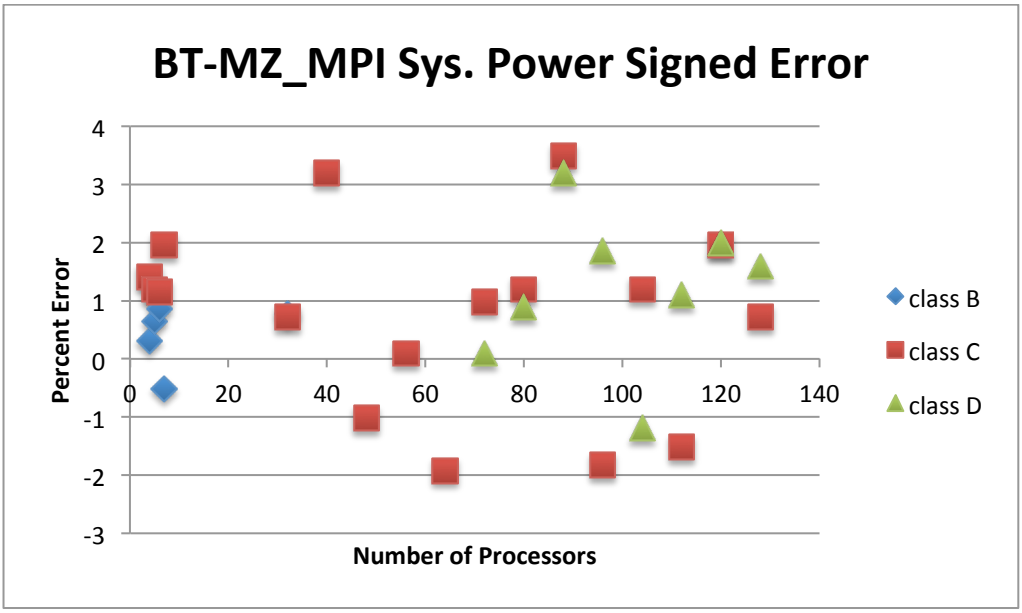

Figure 25. Scatterplot of BT-MZ MPI for System Power Consumption

The majority of the predictions that were made for system power consumption resulted in positive percent error rates or over-predictions. The performance counters used in predicting these values could cause variations in predictions if the counter values fluctuate or do exhibit consistent trends as the number of processors is increased.

However, the modeling methodology provides consistent predictions in the range of +2% to 0% for most values for the BT-MZ MPI application.

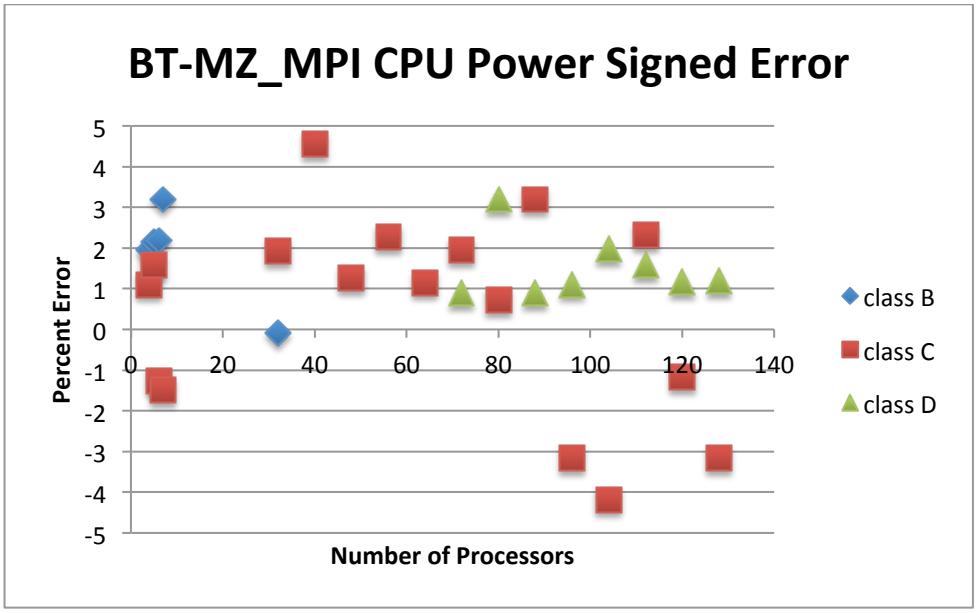

Figure 26. Scatterplot of BT-MZ MPI for CPU Power Consumption

In Figure 26 and Figure 27 we provide an overview of the distribution of each prediction point for the CPU and memory power consumption of the BT-MZ MPI application. The figures illustrate that a large number of the points are over-predicted or positive across all problem sizes. For both CPU and memory power consumption prediction it is shown that all of the predictive values for class B and class D remain positive. In predicting the CPU power consumption the values for class C result in negative values for the larger processor sizes.

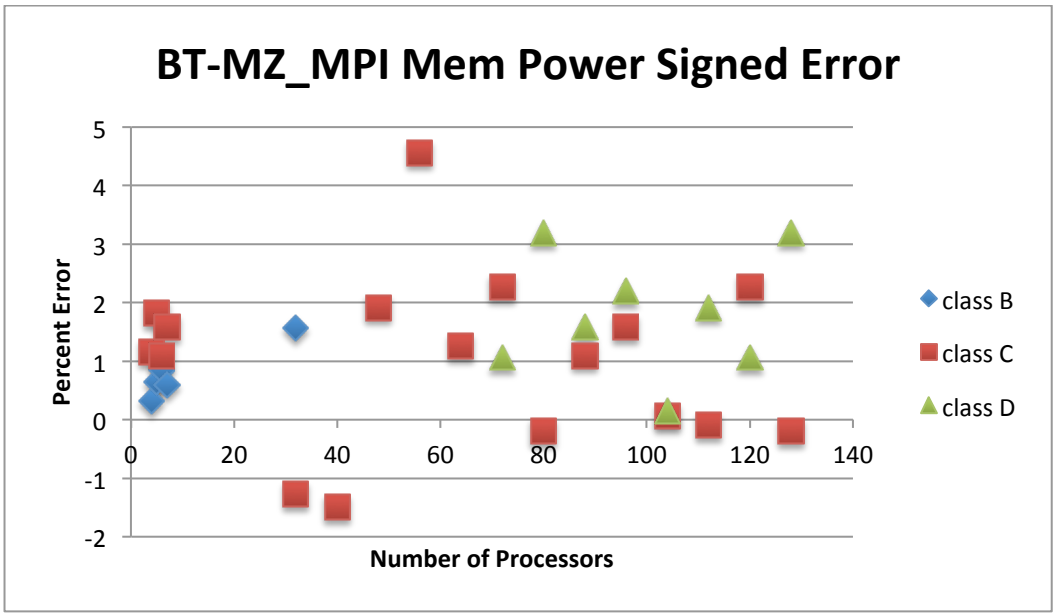

Figure 27. Scatterplot of BT-MZ MPI for Memory Power Consumption

4.3.2 SP-MZ

4.3.2.1 Hybrid

In this section, we use our performance-tuned principal component analysis method to develop accurate models for the hybrid implementation of the NAS SP-MZ application benchmark. The components that we utilize to model our application include runtime, system power, CPU power, and memory power. Table 20 shows the regression coefficients that are needed to accurately model each component. With regards to time, PAPI_L2_TCH has the largest regression coefficient that is used in modeling the SP-MZ hybrid application. System power shows that PAPI_L1_TCM and PAPI_L2_TCH have the largest regression coefficient in modeling the system power consumption.

Table 20. Regression Coefficients for SP-MZ Hybrid

Time		System Power		CPU Power		Memory Power	
Frequency	0.0015175	Frequency	0.001322	Frequency	0.001333	Frequency	0.0264156
PAPI_TOT_INS	0.0105484	PAPI_L1_ICA	0.2287	LD_ST_stall	0.01897	Cache_FLD	0.315519
PAPI_L1_TCA	0.002452	PAPI_L2_TCH	0.2378	PAPI_L1_TCM	0.175466	LD_ST_stall	0.0054159
PAPI_L2_TCH	0.63284	PAPI_L1_TCM	0.99784	PAPI_L2_TCH	0.401895	PAPI_L2_TCH	0.0948931
PAPI_L1_TCM	0.12548					PAPI_L2_TCA	0.07819535

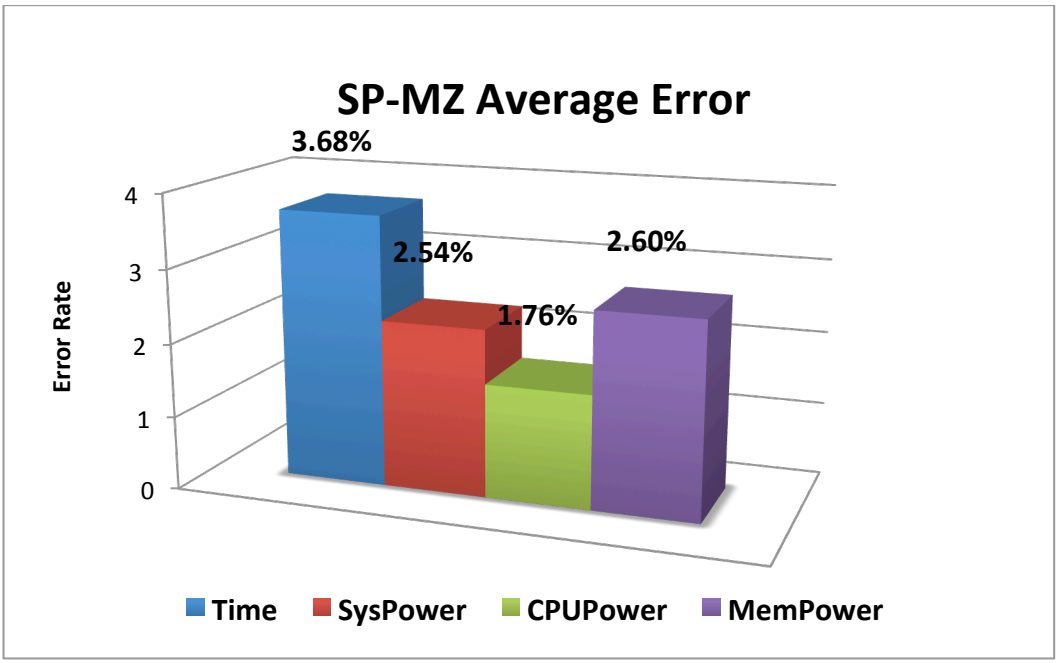

Figure 28. Average Error of SP-MZ Hybrid

With regards to modeling of the CPU power consumption, the PAPI_L2_TCH counter has the largest regression coefficient in comparison to the other counters and components. The PAPI_L1_TCM counter has the 2nd largest regression coefficient in modeling the CPU power consumption. The memory power consumption is modeled

using four performance counters including: Cache_FLD_per_instruction, LD_ST_stall, PAPI_L2_TCH, and PAPI_L2_TCA. The Cache_FLD_per_instruction has the largest regression coefficient for modeling memory power.

Figure 28 shows the average error resulting from the modeling of the SP-MZ hybrid application. The performance components modeled for the SP-MZ hybrid application had an average prediction error of less than 3%. Specifically, the smallest error was found for predicting the CPU power consumption for the application across 40 prediction points, less than 2%. On the other hand, the highest error rate occurred in modeling the runtime (3.78%).

In Figure 29 we provide an overview of the distribution of each prediction point for the runtime of SP-MZ Hybrid application. The performance counters used for predicting runtime are able to provide for consistent predictions across the various application inputs. The values show that the prediction for the runtime is largely positive for across all datasets.

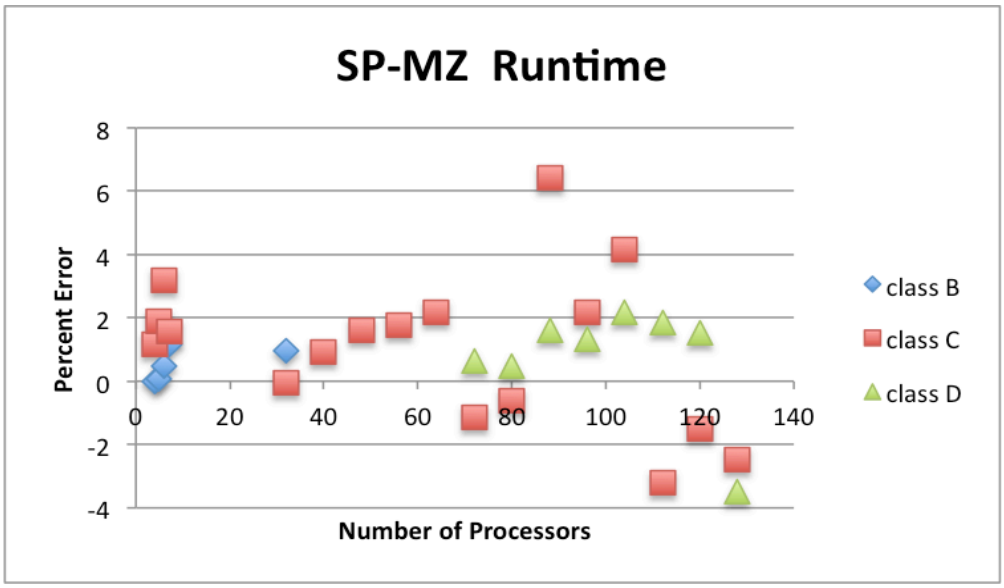

Figure 29. Scatterplot of SP-MZ Hybrid for Runtime

For predicting the runtime for class B all prediction points are positive and remain less than 2% as the number of processors are increased. Class C shows slight fluctuations in predicting the runtime, specifically, as the number of processors increase pass 104 our runtime model under-predicts. The predictions for the Class D benchmark show that all points except for 1 are positive which shows that the runtime model predicts well within the range of 0 to 2%.

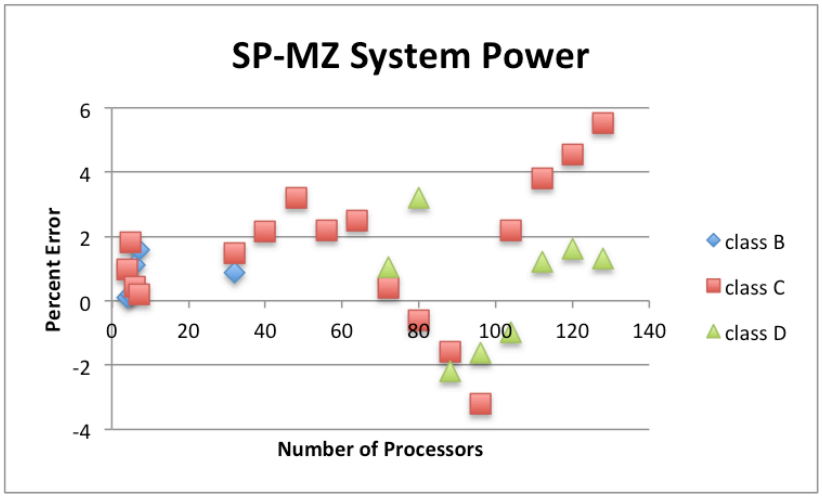

Figure 30. Scatterplot of SP-MZ Hybrid for System Power Consumption

Figure 30 outlines the prediction trends of the system power consumption for the SP-MZ Hybrid application. The prediction values illustrated show the overall trends in prediction across class B, class C, and class D. For predicting the system power consumption for class B all prediction points are positive and remain less than 2% as the number of processes are increased. Class C shows several clusters of trends as the number of processors increase. Figure 30 shows that for less than 72 processes the percent error for most predictions is positive. There is a linear decrease in the prediction accuracy from 72 to 96 processes, however, from 104 processes to 128 processes our model results in positive percent error for the modeling predictions.

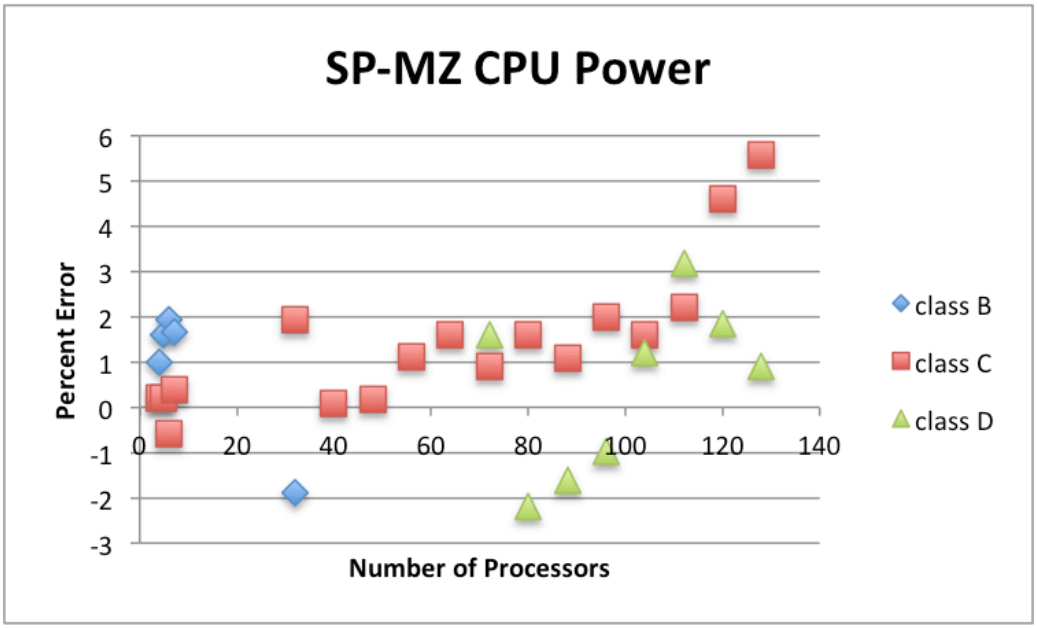

Figure 31. Scatterplot of SP-MZ Hybrid for CPU Power Consumption

In Figure 31 we show the values associated with our model of SP-MZ Hybrid for predicting CPU Power consumption. Largely, for class B the values are positive except for one negative value at 32 processes. The overall trend in prediction for class C results in positive prediction values for most processor points.

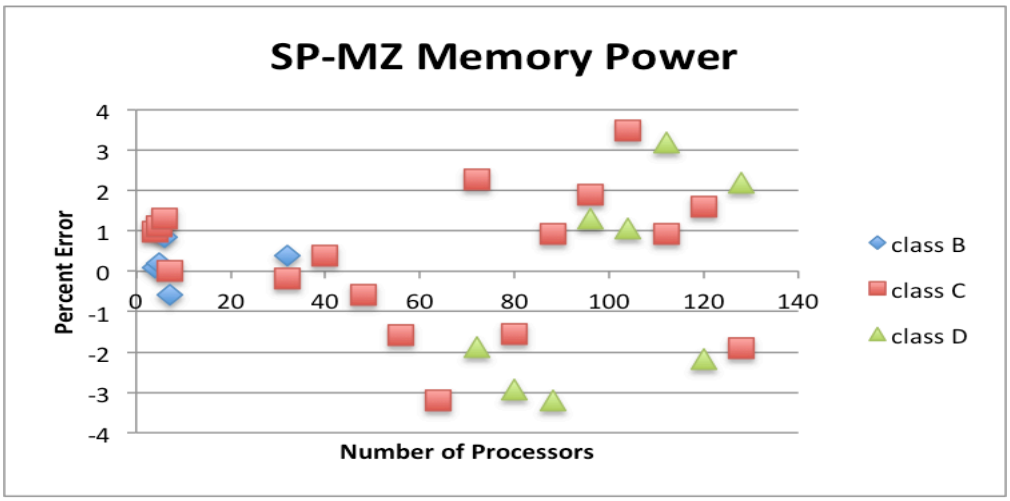

Figure 32. Scatterplot of SP-MZ Hybrid for Memory Power

In Figure 32 we show the values associated with our model of SP-MZ Hybrid for predicting memory power consumption. The memory power consumption shows different patterns that highlight the clustering of prediction values at different processor sizes. The usage of PAPI_L2_TCH and PAPI_L2_TCA has an effect on the ability of our model to accurately predict as the number of processors increase. The best fit using these two performance counters can result in over prediction or under-prediction of values across class C and class D. The smaller memory requirements by class B makes it less likely for large variances to occur when predicting memory power consumption.

4.3.2.2 MPI

In this section we present the models for the MPI implementation of the NAS SP-MZ application. The components that we utilize to model our application include runtime, system power, CPU power, and memory power. Table 21 shows the regression

coefficients that are needed to accurately model each component. With regards to runtime, PAPI_L1_TCM has the largest regression coefficient that is used in modeling the SP-MZ MPI application. System power shows that PAPI_TOT_INS has the largest regression coefficient in modeling the system power consumption of the application.

Table 21. Regression Coefficients for SP-MZ MPI

Time		System Power		CPU Power		Memory Power	
Frequency	0.0013182	**Frequency**	0.21895	**Frequency**	0.001253	**Frequency**	0.0264156
PAPI_TOT_INS	0.0475319	**PAPI_TOT_INS**	0.50888	**Cache_FLD_**	0.01897	**Cache_FLD_**	0.315519
PAPI_L1_TCA	0.0380158	**PAPI_L2_TCH**	0.41659	**PAPI_L1_TCA**	0.175466	**LD_ST_stall**	0.0054159
PAPI_L2_TCH	0.0761895	**PAPI_L1_TCM**	0.08198	**PAPI_L2_TCH**	0.401895	**PAPI_L2_TCH**	0.0948931
PAPI_L1_TCM	0.0818185					**PAPI_L2_TCA**	0.07819535

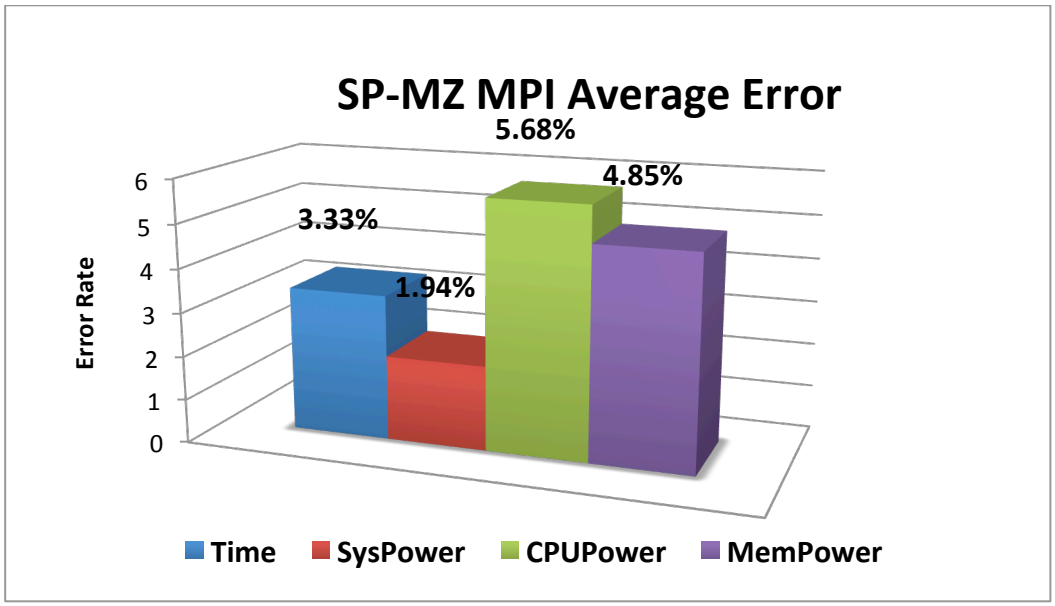

Figure 33. Average Error of SP-MZ MPI

With regards to modeling of the CPU power consumption, the PAPI_L2_TCH counter has the largest regression coefficient in comparison to the other counters and components. The memory power consumption is modeled using Cache_FLD_per_instruction, LS_ST_stall_per_instruction, PAPI_L2_TCH, and PAPI_L2_TCA. Cache_FLD_per_instruction has the largest regression coefficient.

Figure 33 shows the average error resulting from the modeling of the SP-MZ MPI application. The performance components modeled for the SP-MZ MPI application had an average prediction error of less than 3.6%. Specifically, the smallest error was found for predicting the runtime for the application, which was 1.94%. On the other hand, the highest error rate occurred in modeling the CPU power consumption (5.68%).

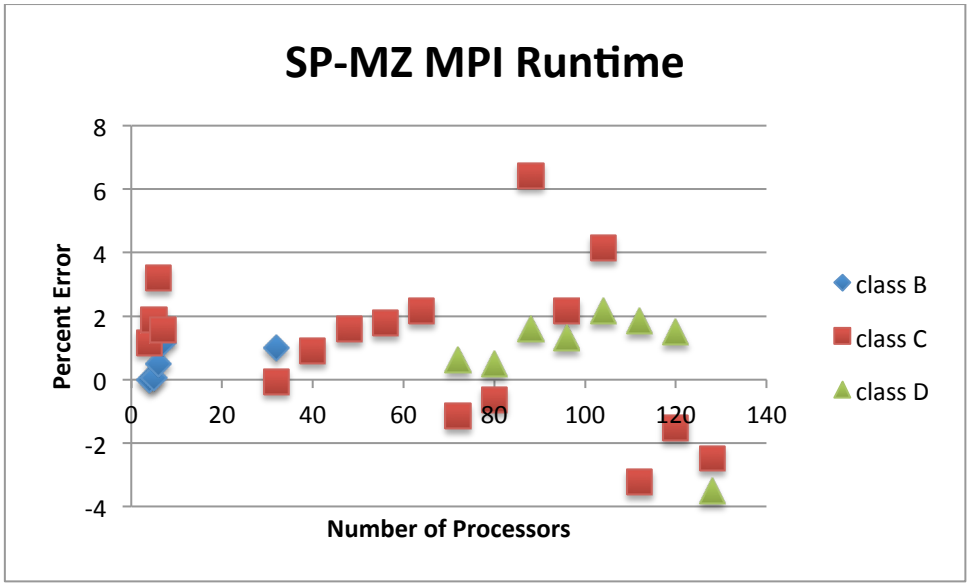

Figure 34. Scatterplot of SP-MZ MPI for Runtime

In Figure 34 we provide an overview of the distribution of each prediction point for the runtime of SP-MZ MPI application. For predicting the runtime for class B all prediction points are positive and remain less than 2%. Class C has the largest number of points used for prediction and these values remain largely clustered between +4% and 0%. The predictions for the Class D benchmark show that all points except for 1 are positive which shows that the runtime model predicts well within the range of 0 to 2%.

Figure 35 shows the distribution of prediction points for system power consumption. There are slight variations in the prediction values as the number of processors increases past 96. For class C the values increase for the larger processors sizes. For class D the prediction values result in negative values but then increase to positive predictive values.

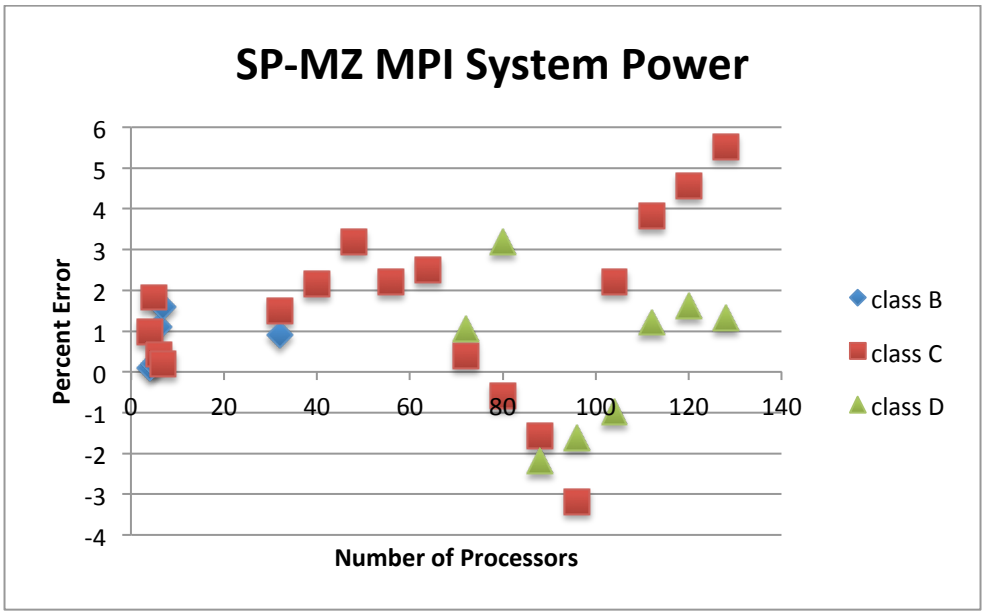

Figure 35. Scatterplot of SP-MZ MPI for System Power Consumption

In Figure 36 we show the values associated with our model of SP-MZ MPI for predicting CPU power consumption. The modeling of the CPU power consumption for SP-MZ MPI results in positive-valued predictions for most values in class B and class C. Class D has a cluster of negative predictive values from 80 to 96 processors which could indicate a performance trend that occurs for the workload executed on 80 to 96 processors.

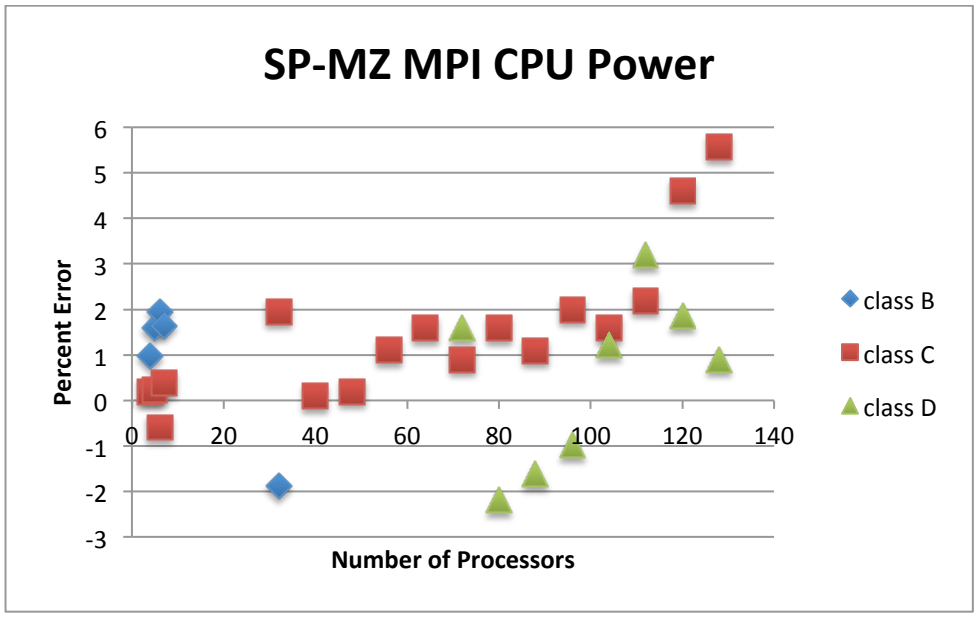

Figure 36. Scatterplot of SP-MZ MPI for CPU Power Consumption

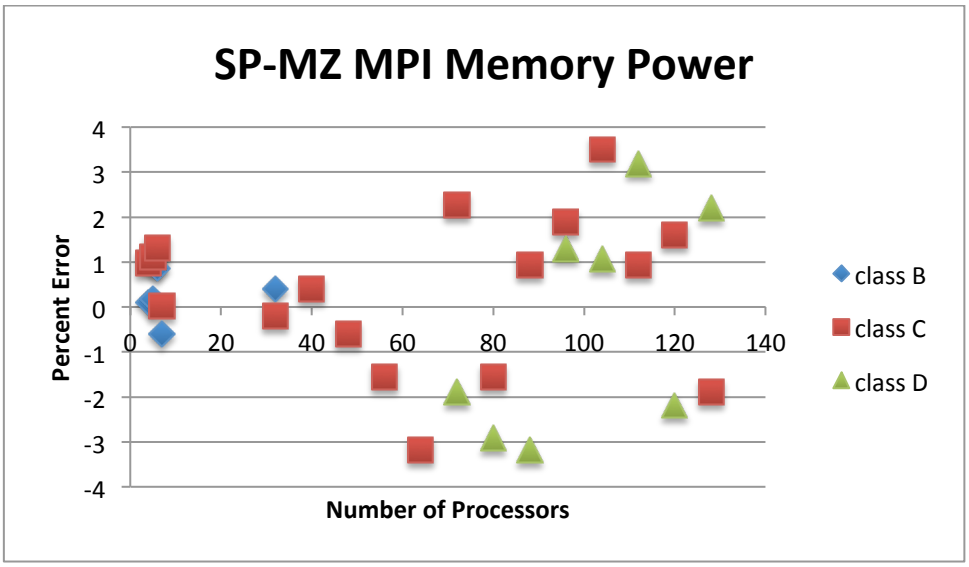

Figure 37. Scatterplot of SP-MZ MPI for Memory Power Consumption

In Figure 37 we show the values associated with our model for predicting memory power consumption of the SP-MZ MPI application. The memory power consumption shows different patterns that highlight the clustering of prediction values at different processor sizes. Prediction of the memory power consumption exhibits larger variability than other components for SP-MZ MPI. This larger variability could be an indication that different models could provide for more consistent predictions for prediction power consumption of the memory component.

4.3.3 LU-MZ

4.3.3.1 Hybrid

In this section, we use our performance-tuned principal component analysis method to develop accurate models for the Hybrid implementation of the NAS LU-MZ application benchmark. The components that we utilize to model our application include runtime, system power, CPU power, and memory power. Table 22 shows the regression coefficients that are needed to accurately model each component. With regards to time, PAPI_TLB_DM and PAPI_L2_TCH have the largest regression coefficients that are used in modeling the runtime for the LU-MZ Hybrid application. System power shows that PAPI_L2_TCA and PAPI_L2_TCH have the largest regression coefficients in modeling the system power consumption for the hybrid implementation of the application.

Table 22. Regression Coefficients for LU-MZ Hybrid

Time		System Power		CPU Power		Memory Power	
Frequency	0.001322	**Frequency**	0.00109	**Frequency**	0.00112	**Frequency**	0.00178
Cache_FLD	0.005197	**PAPI_L2_TCH**	0.11585	**LD_ST_stall**	0.06548	**PAPI_L1_TCA**	0.01877
PAPI_TOT_INS	0.000518	**PAPI_L2_TCA**	0.12254	**PAPI_L2_TCH**	0.1894	**PAPI_L2_TCH**	0.08431
PAPI_TLB_DM	3.9085	**PAPI_RES_STL**	0.0211	**PAPI_L2_TCA**	0.7149	**PAPI_RES_STL**	0.07451
PAPI_L2_TCH	1.11565						

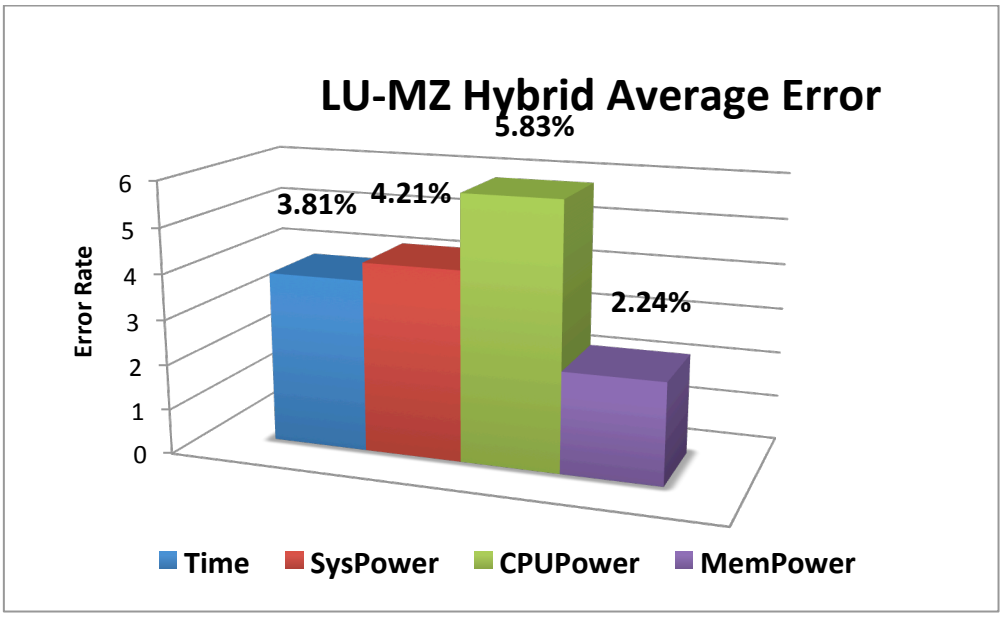

Figure 38. Average Error of LU-MZ Hybrid

With regards to modeling of the CPU power consumption, the PAPI_L2_TCA counter has the largest regression coefficient in comparison to the other counters and components. The PAPI_L2_TCA counter has the 2nd largest regression coefficient in modeling the CPU power consumption. For modeling the memory power consumption of the LU-MZ hybrid application, the PAPI_L2_TCH and PAPI_RES_STL counters have the largest regression coefficients.

Figure 38 shows the average error resulting from the modeling of the LU-MZ Hybrid application. The performance components modeled for the LU-MZ Hybrid application have an average prediction error of 4.02%. The smallest error was found for predicting the memory power consumption for the application. On the other hand, the highest error rate occurred in modeling the CPU power consumption (5.83%).

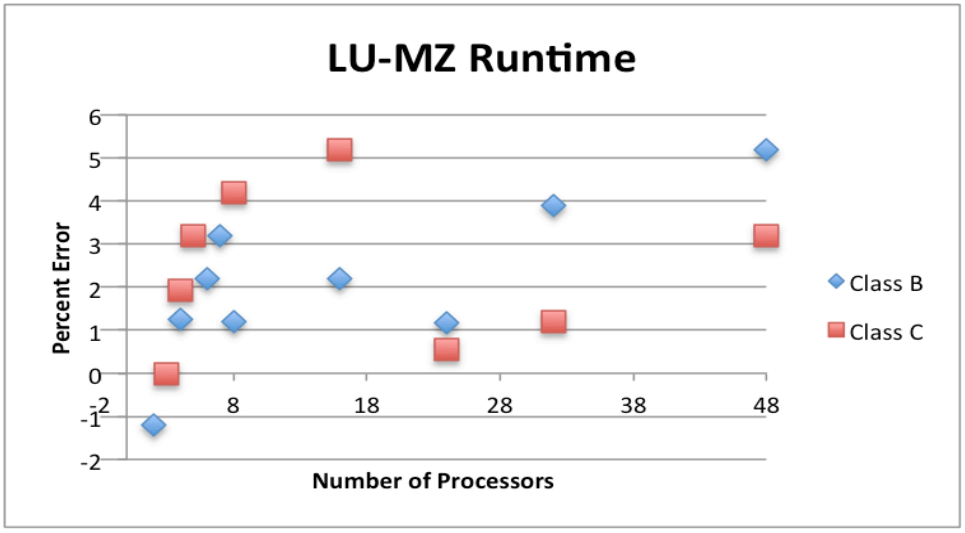

Figure 39. Scatterplot of LU-MZ Hybrid for Runtime

In Figure 39 we provide an overview of the distribution of each prediction point for the runtime of LU-MZ Hybrid application. The performance counters used for predicting runtime are able to provide for consistently positive values for predicting the runtime across the two application inputs. The values show that the prediction for the runtime is largely positive for the application datasets.

For predicting the runtime for class B all prediction points except one are positive and remain less than 5% as the number of processes are increased. In addition, class C shows all positive points for predicting the runtime, specifically, as the number of processes increases passes. The scalability of the LU-MZ benchmark is limited so the maximum number of processors that could be used to obtain results was limited to 48 processors.

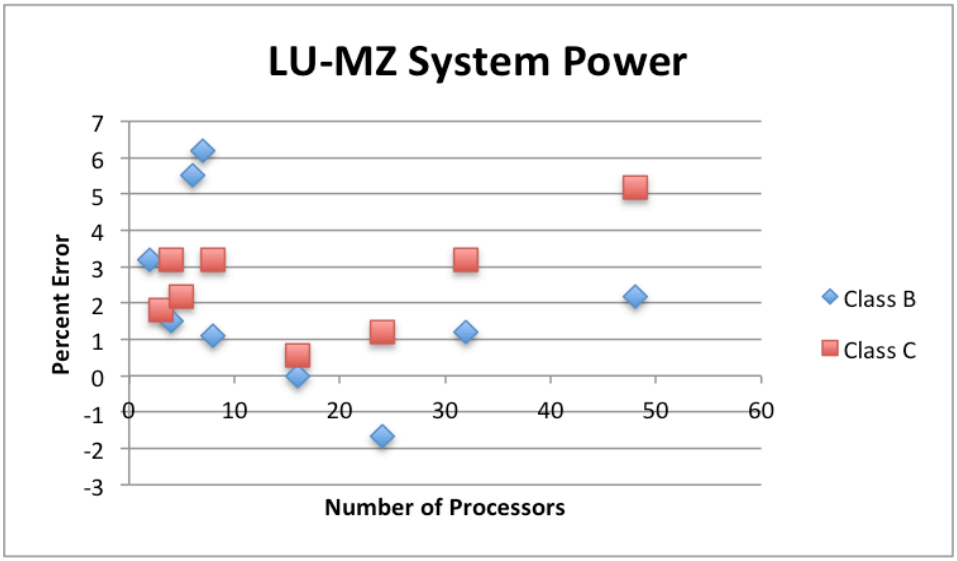

Figure 40. Scatterplot of LU-MZ Hybrid for System Power

Figure 40 outlines the prediction values of the system power consumption for the LU-MZ hybrid application. The prediction values show the general trend towards positive values for class B and class C. For predicting the system power for class B all prediction points are positive except for one point. The error is slightly large for predicting class B system power at certain points for less than 8 processes. This occurs because the general scalability of the application is limited as the number of threads increase within one node or less than 8 processors. The large error in prediction also occurs with class C, but decreases and then shows slight increases as the number of processors increase.

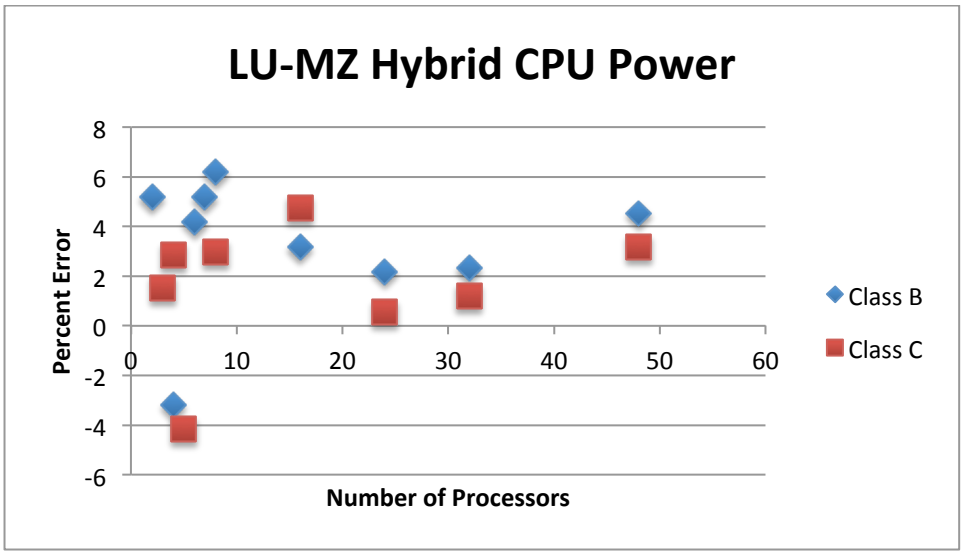

Figure 41. Scatterplot of LU-MZ Hybrid for CPU Power

In Figure 41 we show the values associated with our model of LU-MZ Hybrid for predicting CPU Power consumption. The CPU Power consumption shows some interesting trends for the prediction of the CPU power consumption of LU-MZ hybrid. Largely, for class B and class C the values are positive except for one negative value at 4 processes for each class. The values for prediction CPU power consumption are largely consistent based upon the values exhibited from the Cache_FLD_per_instruction and PAPI_L2_TCA counters for this application.

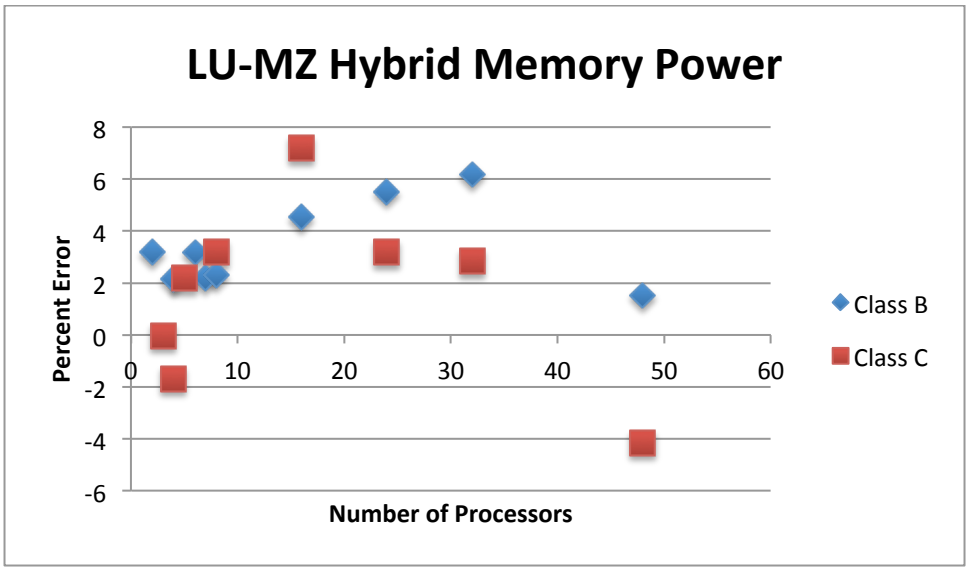

Figure 42. Scatterplot of LU-MZ Hybrid for Memory Power Consumption

Figure 42 outlines the prediction trends of the memory power for the LU-MZ Hybrid application. For predicting the memory power consumption for class B all prediction points are positive and remain positive as the number of processes is increased. Class C shows different trends in predicting memory power consumption as the number of processors increase. The larger memory required by the class C application input results in having prediction trends that are not linear as the number of processes increase.

4.3.3.2 MPI

In this section, we use our performance-tuned principal component analysis method to develop accurate models for the MPI implementation of the NAS LU-MZ application benchmark. The components that we utilize to model our application include

runtime, system power, CPU power, and memory power. Table 23 shows the regression coefficients that are needed to accurately model each component. With regards to time, PAPI_TLB_DM and PAPI_L2_TCH have the largest regression coefficients that are used in modeling the runtime for the LU-MZ MPI application. System power shows that PAPI_L2_TCA and PAPI_L2_TCH have the largest regression coefficients in modeling the system power consumption for the MPI implementation of the application.

Table 23. Regression Coefficients for LU-MZ MPI

Time		System Power		CPU Power		Memory Power	
Frequency	0.101195	**Frequency**	0.03128	**Frequency**	0.17488	**Frequency**	0.003184
Cache_FLD_	0.0459785	**PAPI_TOT_INS**	0.01233	**PAPI_TOT_INS**	0.068175	**PAPI_L2_TCA**	0.218984
PAPI_TOT_INS	0.3718951	**PAPI_L1_DCA**	0.01984	**Cache_FLD_**	0.031820	**PAPI_L2_TCH**	0.541351
PAPI_L1_ICA	1.5165885	**PAPI_L2_TCA**	0.33187	**PAPI_L2_TCA**	0.018749	**Cache_FLD_**	0.411589
PAPI_L2_TCA	1.23155						

With regards to modeling of the CPU power consumption, the PAPI_TOT_INS counter has the largest regression coefficient in comparison to the other counters and components. The Cache_FLD_per_instruction counter has the 2nd largest regression coefficient in modeling the CPU power consumption. For modeling the memory power consumption of the LU-MZ MPI application, the PAPI_L2_TCH and Cache_FLD_per_instruction counters have the largest regression coefficients.

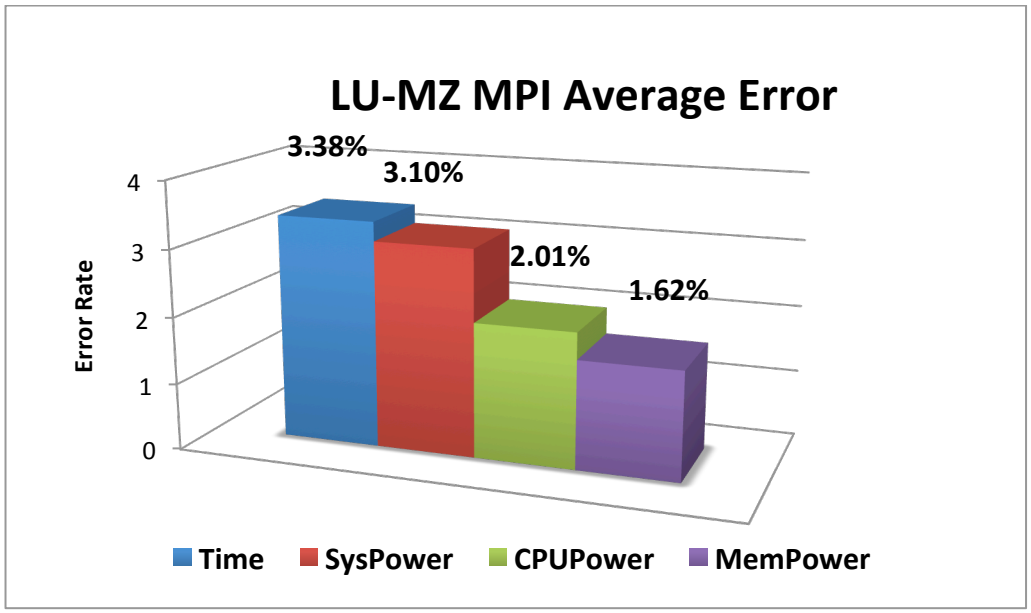

Figure 43. Average Error of LU-MZ MPI

Figure 43 shows the average error resulting from the modeling of the LU-MZ MPI application. The performance components modeled for the LU-MZ MPI application had an average prediction error of less than 4%. Specifically, the smallest error was found for predicting the memory power consumption (1.62%) for the application. The highest error rate occurred in modeling runtime (3.38%).

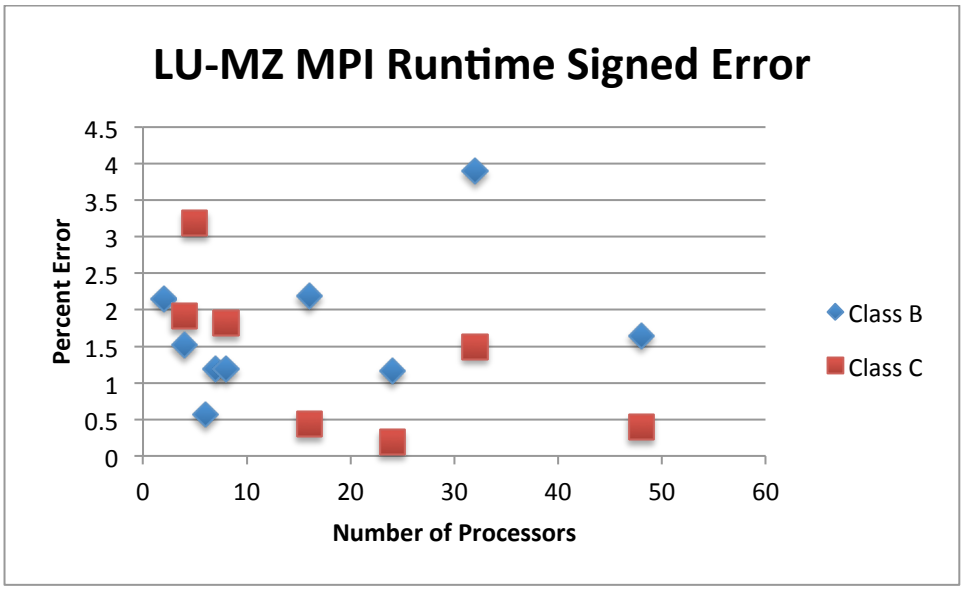

Figure 44. Scatterplot of LU-MZ MPI for Runtime

In Figure 44 we provide an overview of the distribution of each prediction point for the runtime of LU-MZ MPI application. The values show that the prediction for the runtime is positive across both datasets and for all prediction points. Figure 40 provides an overview of the distribution for each prediction point for the system power consumption. Class B shows that for less than 8 processors that there are slightly large error values that could be a result of the lack of scalability for the LU-MZ application.

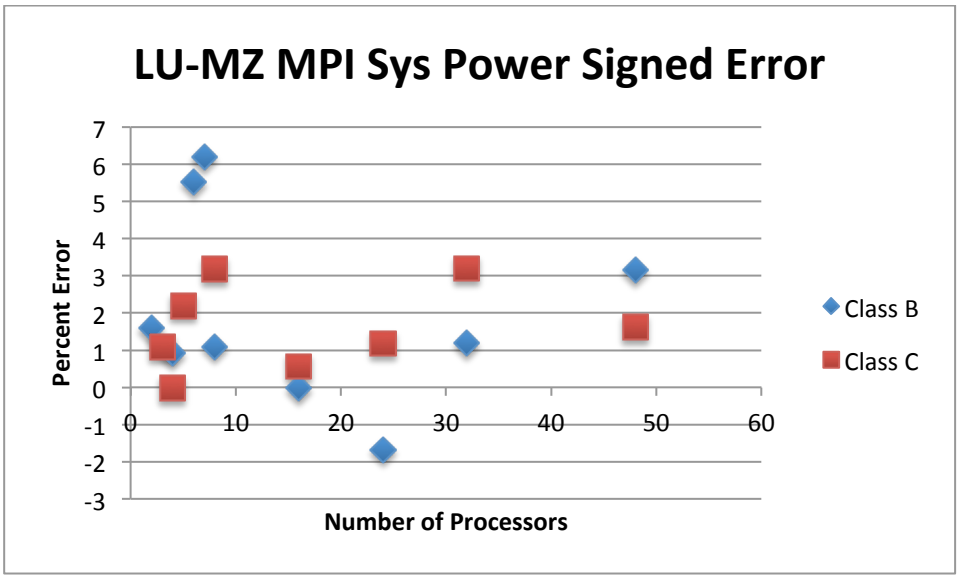

Figure 45. Scatterplot of LU-MZ MPI for System Power Consumption

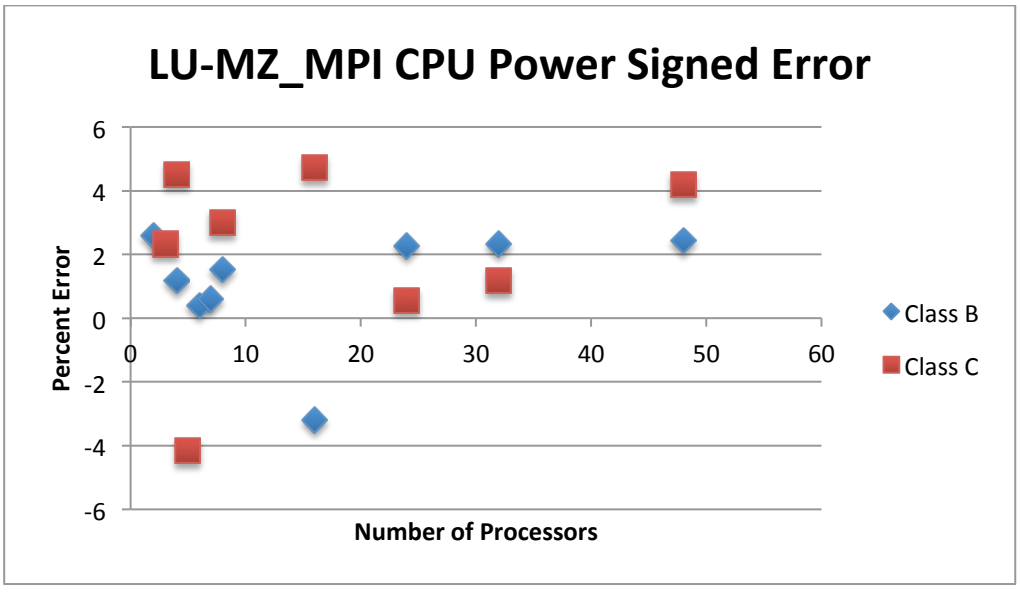

Figure 46. Scatterplot of LU-MZ MPI for CPU Power Consumption

In Figure 46 we show the values associated with our model of LU-MZ MPI for predicting CPU Power consumption. Trends for the prediction of the CPU power

consumption of LU-MZ MPI show a general pattern that prediction for the class B application input remains in the range of +3% to +1%.. Largely, for class B and class C the values are positive except for one negative for each class.

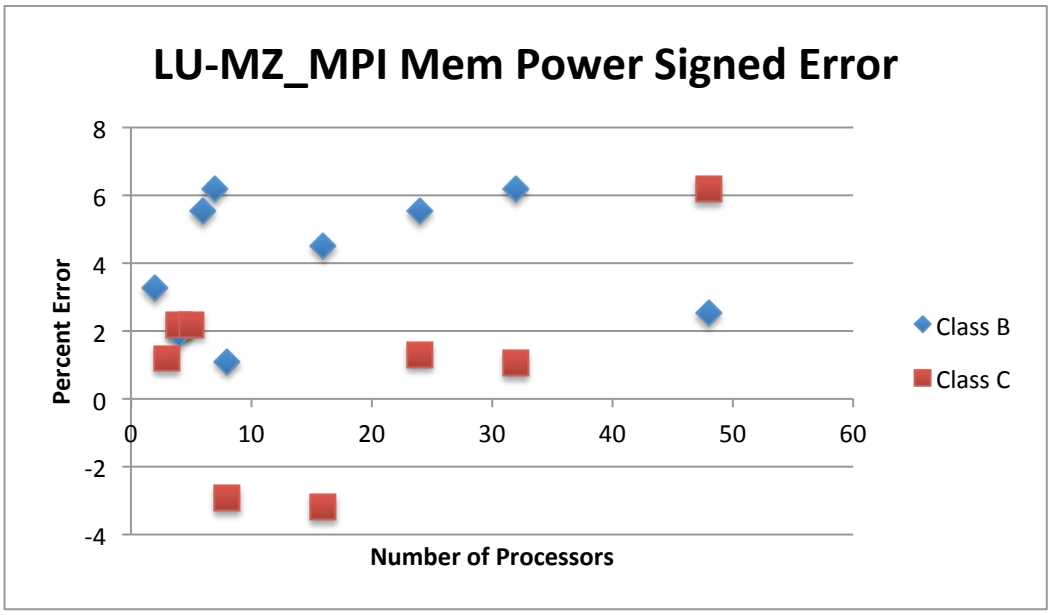

Figure 47. Scatterplot of LU-MZ MPI for Memory Power Consumption

Figure 47 outlines the prediction trends of the memory power for the LU-MZ MPI application. For predicting the memory power consumption for class B all prediction points are positive and remain positive as the number of processes increase. The prediction errors for class B remain in the range of +6% to +2%.

4.3.4 GTC

4.3.4.1 Hybrid

In this section, we use our performance-tuned principal component analysis method to develop accurate models for the Hybrid implementation of the large-scale GTC application. The components that we utilize to model our application include runtime, system power, CPU power, and memory power. Table 24 shows the regression coefficients that are needed to accurately model each component.

With regards to runtime, PAPI_TOT_INS and PAPI_L2_TCA have the largest regression coefficients that are used in modeling the runtime for the GTC Hybrid application. System power shows that PAPI_L2_TCH and PAPI_L1_TCA have the largest regression coefficients in modeling the system power consumption for the hybrid implementation of the application.

Table 24. Regression Coefficients for GTC Hybrid

Time		System Power		CPU Power		Memory Power	
Frequency	0.012382	**Frequency**	0.19549	**Frequency**	1.1658	**Frequency**	1.16585
PAPI_TOT_INS	0.054015	**PAPI_RES_STL**	0.41849	**PAPI_RES_STL**	0.34649	**PAPI_TOT_IN**	0.06716
PAPI_L2_TCH	0.008475	**PAPI_L2_TCH**	1.54152	**PAPI_TOT_IN**	0.41899	**PAPI_L2_TCH**	1.4942
PAPI_L2_TCA	0.031587	**PAPI_L1_TCA**	1.32657	**PAPI_L1_TCA**	1.32169	**PAPI_L2_ICM**	0.78199
PAPI_BR_INS	0.03157			**PAPI_L2_TCH**	1.16584		

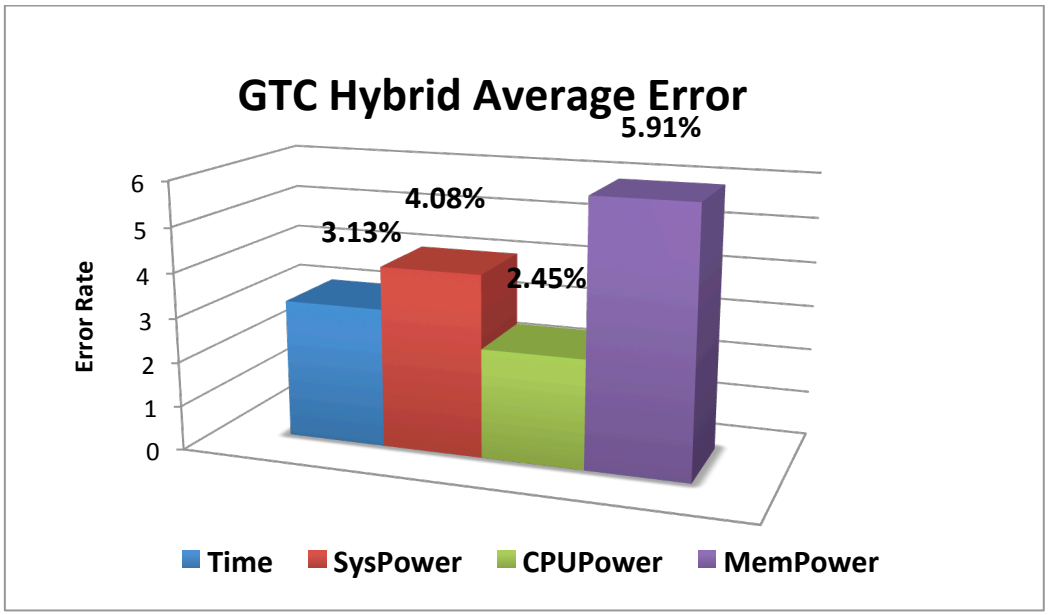

Figure 48. Average Error of GTC Hybrid

With regards to modeling of the CPU power consumption, the PAPI_L1_TCA and PAPI_L2_TCH counters have the largest regression coefficients. For modeling the memory power consumption of the GTC Hybrid application, the PAPI_L2_TCH and PAPI_L2_ICM counters have the largest regression coefficients. Additionally, from Table 24 it can be seen that the PAPI_L2_TCH counter is used across all components.

Figure 48 shows the average error resulting from the modeling of the GTC Hybrid application. The performance components modeled for the GTC Hybrid application had an average prediction error of 3.89%. Specifically, the smallest error was found for predicting the CPU power consumption (2.45%) for the application. On the other hand, the highest error rate occurred in modeling the memory power consumption (5.91%).

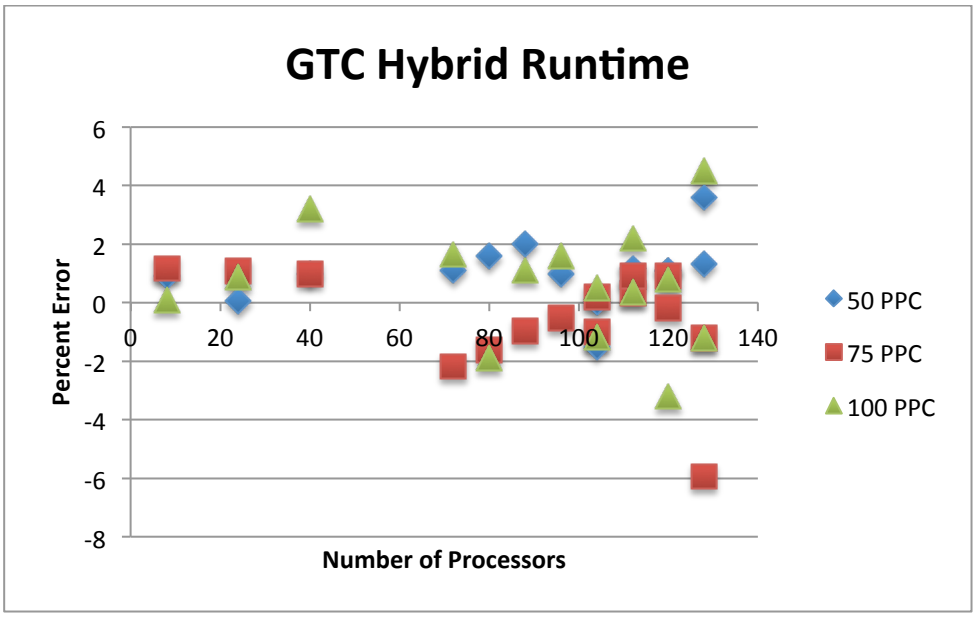

Figure 49. Scatterplot of GTC Hybrid for Runtime

In Figure 49 we provide an overview of the distribution of each prediction point for the runtime of GTC Hybrid application. The performance counters used for predicting runtime are able to provide for consistent values for predicting the runtime across three application inputs, which include 50 particles per cell, 75 particles per cell, and 100 particles per cell. Prediction across all application inputs for GTC remains in the range of +2.0% to -2.0% across all of the processor predictions. The most notable trend occurs for 75 PPC as the prediction of this set increases as processors are increased. This likely is a result of the large communication requirements for GTC, which is not taken into consideration using the performance-tuned modeling methodology.

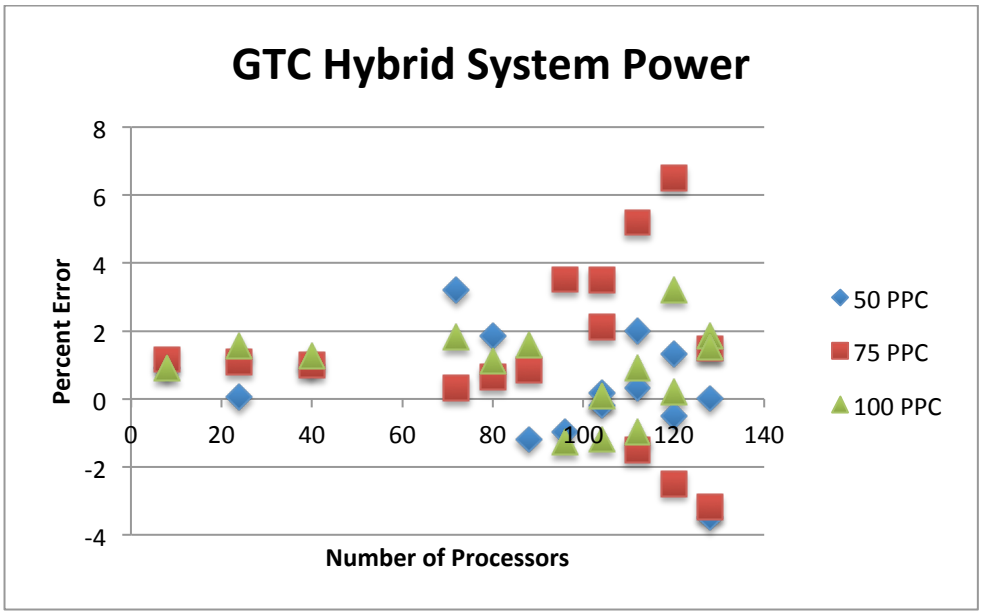

Figure 50. Scatterplot of GTC Hybrid for System Power

Figure 50 outlines the prediction values of the system power consumption for the GTC hybrid application. The prediction values show the spread of prediction values for system power consumption up to 128 processes for the application. For predicting the system power consumption the prediction values are generally positive for less than 96 processes. For predictions larger than 96 processes there are variations in the values being predicted being as the values fluctuate between positive or negative. However, the general trend for the 50 PPC dataset shows that the values predicted stay within the range of +2.0% to -2.0% from 96 processes to 128 processes.

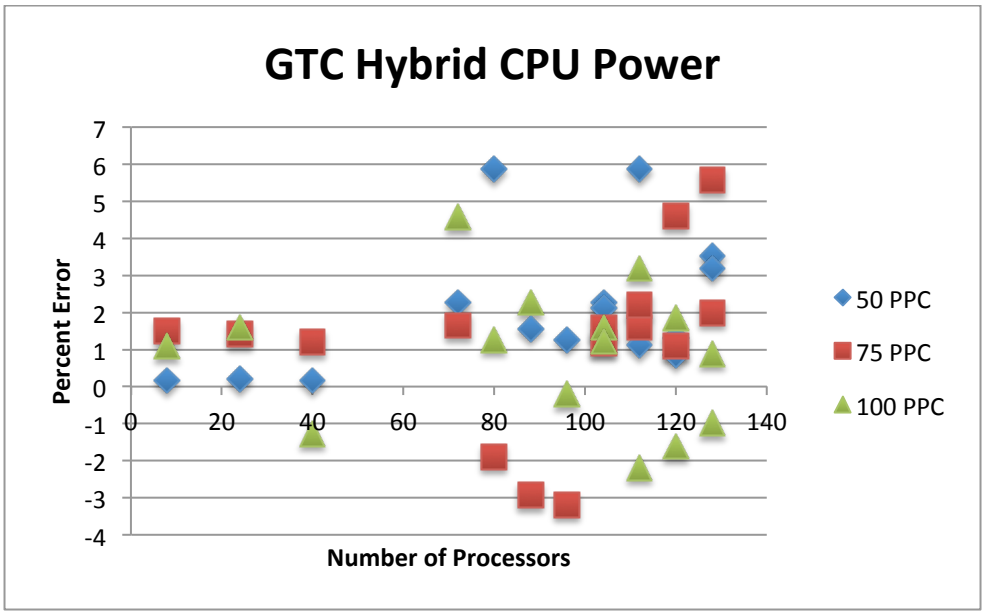

Figure 51. Scatterplot of GTC Hybrid for CPU Power

Figure 51 and Figure 52 show the values associated with our model of GTC hybrid application for predicting CPU and memory power consumption. The values associated with the prediction for these two performance components show variations in the trends and there are several points that are under-predicted past 72 processors for both models. Overall, in modeling both CPU and memory power consumption the values that are predicted are positive as the number of processors increase.

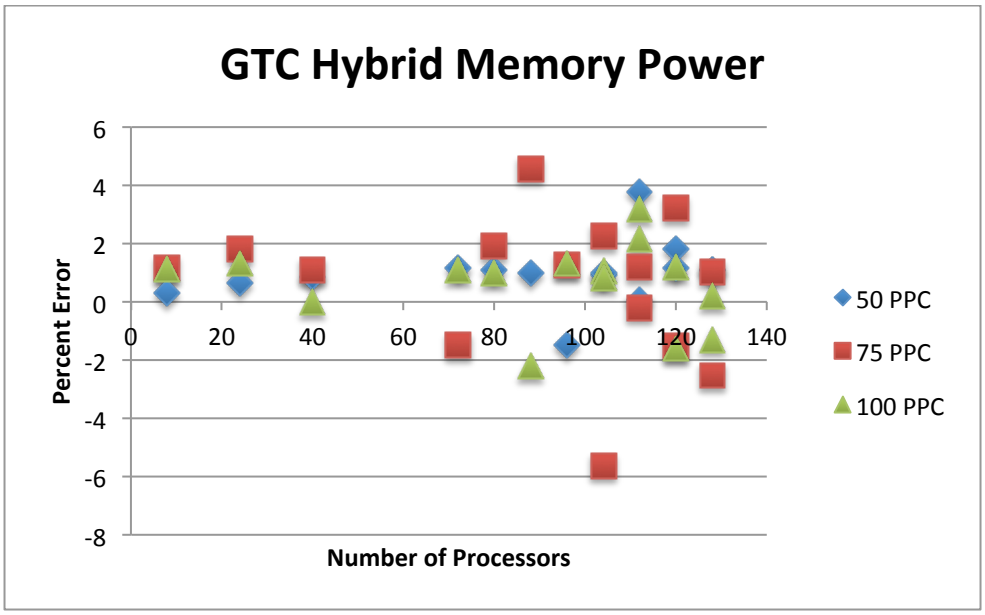

Figure 52. Scatterplot of GTC Hybrid for Memory Power Consumption

4.3.4.2 MPI

In this section, we use our performance-tuned principal component analysis method to develop accurate models for the MPI implementation of the large-scale GTC application. The components that we utilize to model our application include runtime, system power, CPU power, and memory power. Table 25 shows the regression coefficients that are needed to accurately model each component.

With regards to runtime, PAPI_TOT_INS and PAPI_L2_TCM have the largest regression coefficients that are used in modeling the runtime for the GTC MPI application. System power shows that PAPI_TOT_INS and PAPI_L2_TCA have the largest regression coefficients in modeling the system power consumption for the hybrid implementation of the application. It should be noted that the regression coefficient for

the performance counters for the system power has very close regression coefficients, which shows a close weighting for providing an accurate fit for system power prediction.

Table 25. Regression Coefficients for GTC MPI

Time		System Power		CPU Power		Memory Power	
Frequency	0.0018418	**Frequency**	0.01424	**Frequency**	0.161554	**Frequency**	0.016452
PAPI_TOT_INS	0.0269816	**PAPI_TOT_INS**	1.93185	**PAPI_TOT_INS**	0.412689	**PAPI_L1_TCA**	0.0579185
PAPI_L1_TCA	0.0018485	**PAPI_L2_TCH**	1.20056	**PAPI_L2_TCH**	0.398485	**PAPI_L2_TCH**	0.045198
PAPI_L2_TCH	0.0038482	**PAPI_L1_TCA**	1.10245	**LD_ST_Stall**	0.984182	**LD_ST_Stall**	0.00895
PAPI_L2_TCM	0.0097816						

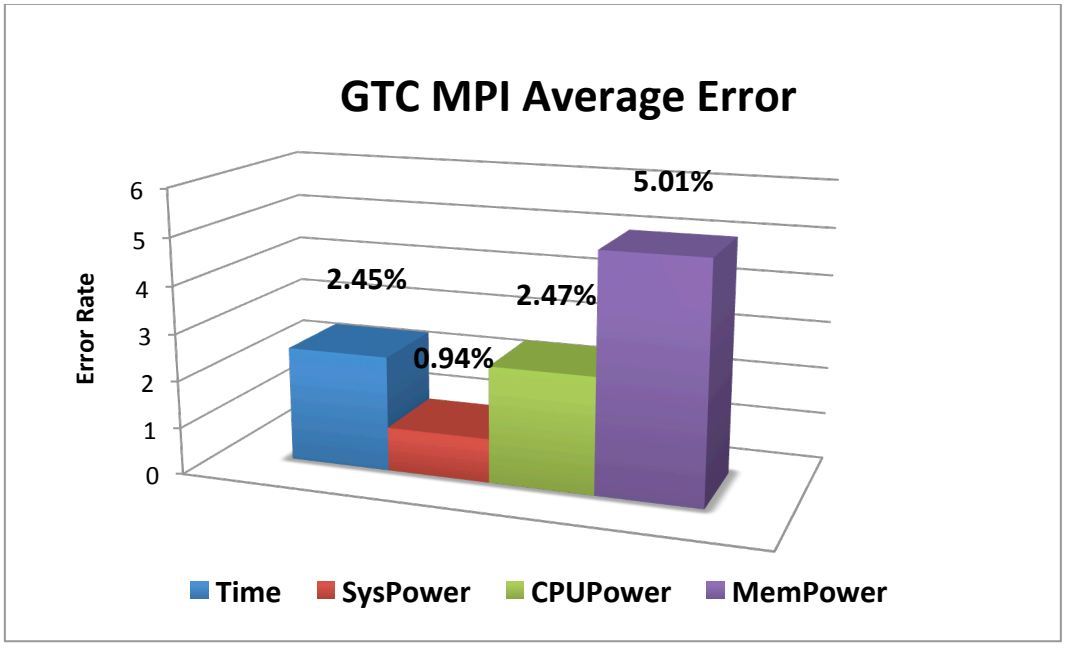

Figure 53. Average Error of GTC MPI

With regards to modeling of the CPU power consumption, the LD_ST_stall_per_cycle and PAPI_TOT_INS counters have the largest regression coefficients. The PAPI_L2_TCH counter also contributes to the modeling of the CPU power consumption as its regression coefficient is less than .04 smaller than the PAPI_TOT_INS regression coefficient. For modeling the memory power consumption of the GTC MPI application, the PAPI_L1_TCA and PAPI_L2_TCH counters have the largest regression coefficients.

Figure 53 shows the average error resulting from the modeling of the GTC MPI application. The performance components modeled for the GTC MPI application had an average prediction error of 2.72%. Specifically, the smallest error was found for predicting the CPU power consumption (0.94%) for the application. On the other hand, the highest error rate occurred in modeling the memory power consumption (5.91%).

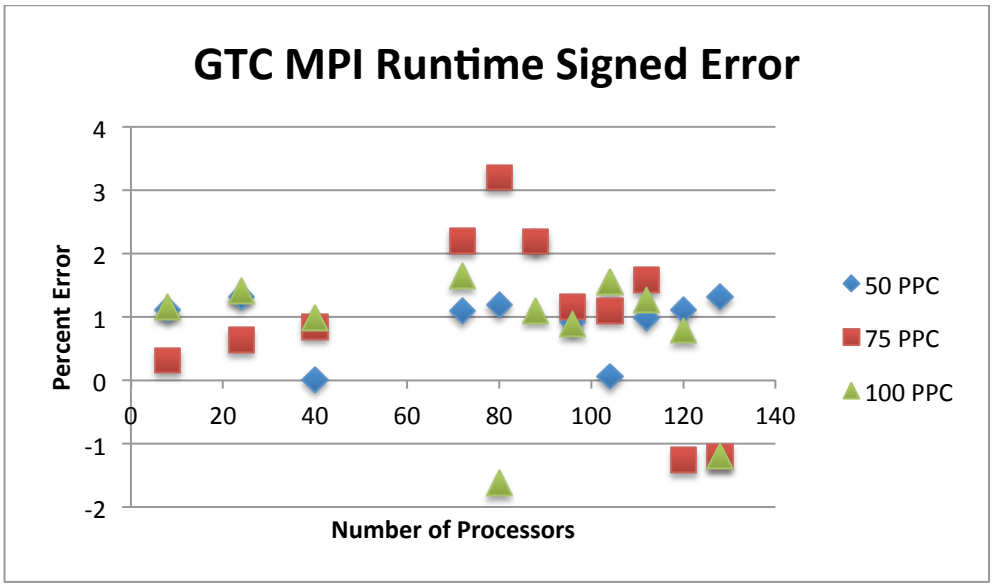

Figure 54. Scatterplot of GTC MPI for Runtime

In Figure 54 and Figure 55 we provide an overview of the distribution of each prediction point for the runtime and system power consumption of the GTC MPI application. The performance counters used for predicting runtime are able to provide for consistent values for predicting the runtime across three application inputs, which include 50 particles per cell, 75 particles per cell, and 100 particles per cell. The majority of the predictions across all application inputs for GTC remain in the range of +2.0% to 0%.

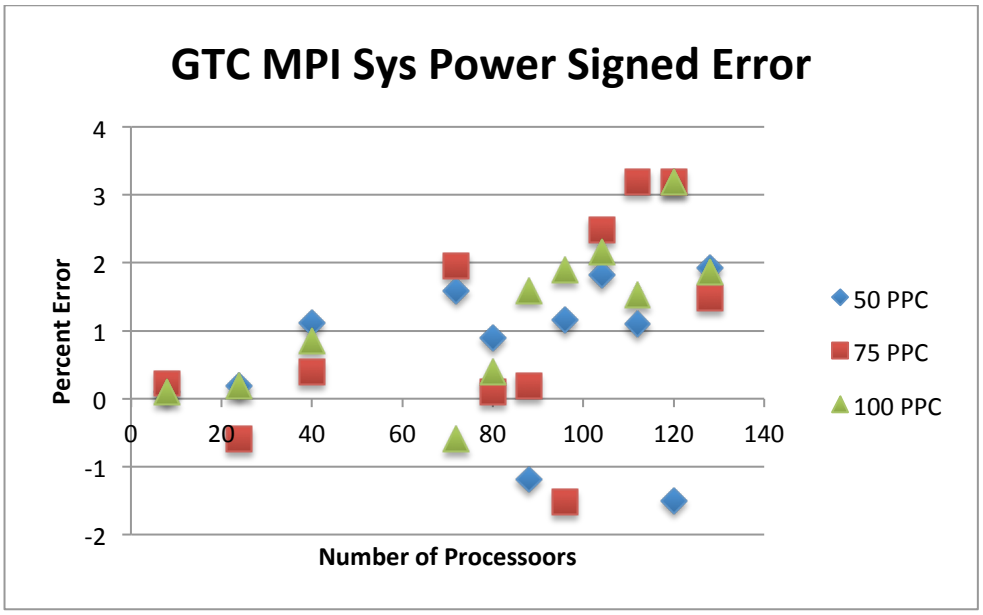

Figure 55. Scatterplot of GTC MPI for System Power Consumption

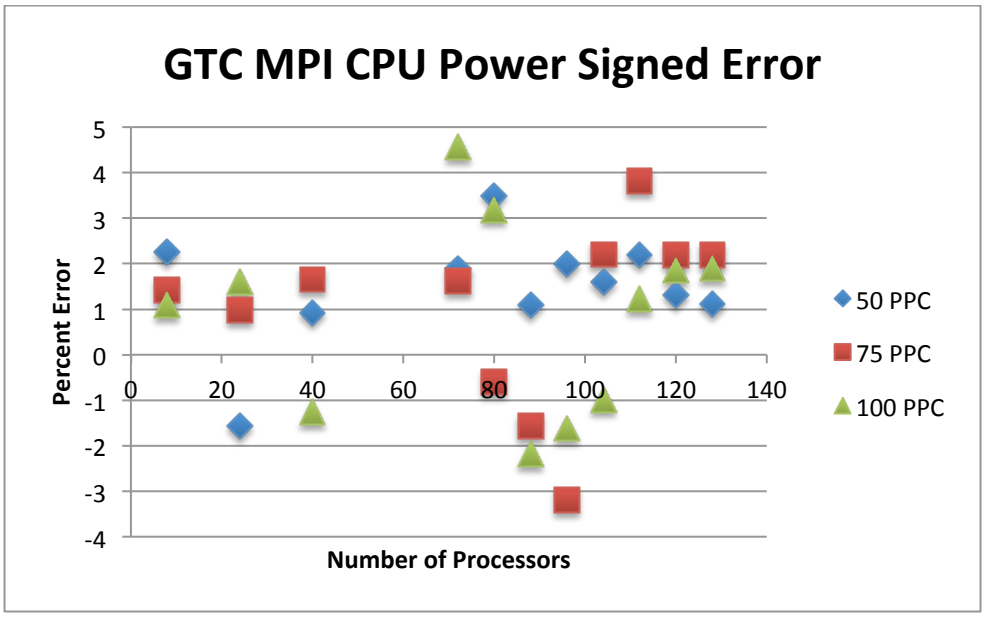

Figure 56. Scatterplot of GTC MPI for CPU Power Consumption

Figure 56 and Figure 57 show the values associated with our model of GTC MPI application for predicting CPU and memory power consumption. With regards to CPU power consumption, the values associated with the prediction for these two performance components show variations that occur for 75PPC and 100PPC. Overall, the predictions for 75PPC remain consistent in the range of +2% to +1% as the number of processors increase. The under predictions that occur from 80 to 96 processors could largely be accountable to an inefficient decomposition when using those number of processors. Most Overall, in modeling both CPU and memory power consumption the values that are predicted are positive as the number of processors increase.

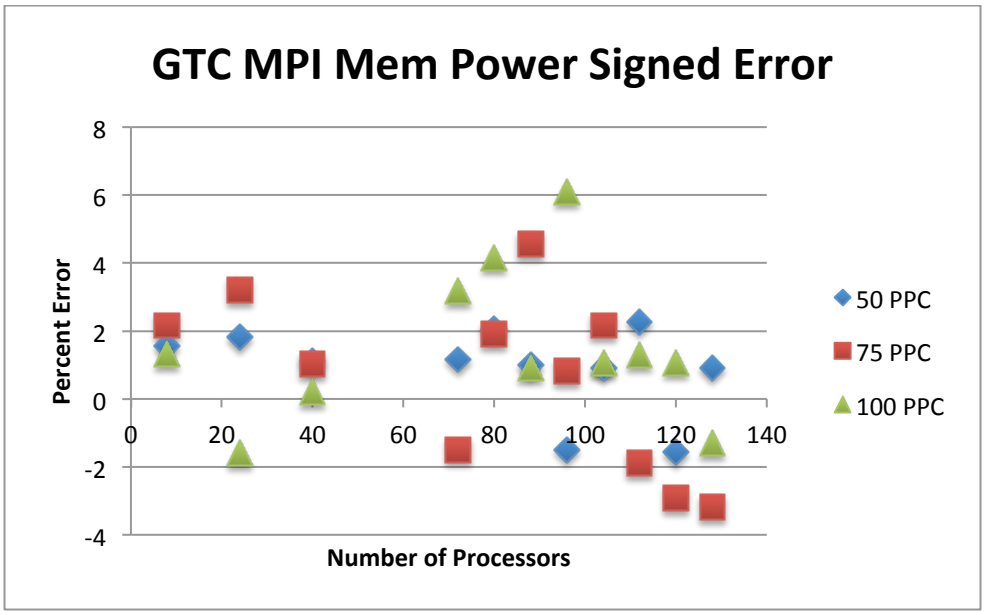

Figure 57. Scatterplot of GTC MPI for Memory Power Consumption

4.3.5 PMLB

4.3.5.1 Hybrid

In this section, we use our performance-tuned principal component analysis method to develop accurate models for the Hybrid implementation of the large-scale PMLB application. Table 26 shows the regression coefficients that are needed to accurately model each component. With regards to runtime, PAPI_L2_TCA and PAPI_RES_STL have the largest regression coefficients that are used in modeling the runtime for the PMLB hybrid application. System power shows that PAPI_L2_TCA and PAPI_L2_TCH have the largest regression coefficients in modeling the system power consumption for the hybrid implementation of the application.

Table 26. Regression Coefficients for PMLB Hybrid

Time		System Power		CPU Power		Memory Power	
Frequency	0.001655	**Frequency**	0.33257	**Frequency**	1.40655	**Frequency**	1.165849
PAPI_TOT_INS	0.0006702	**PAPI_TOT_INS**	0.00519	**PAPI_TOT_INS**	0.00519	**PAPI_TOT_INS**	0.000352
PAPI_L1_TCA	0.0001899	**PAPI_L2_TCH**	0.51562	**PAPI_L1_TCA**	0.678485	**PAPI_L2_TCH**	0.55189
PAPI_L2_TCA	0.05918	**PAPI_L2_TCA**	1.15642	**PAPI_L2_TCA**	0.481518	**PAPI_RES_STL**	0.11909
PAPI_RES_STL	0.0273006						

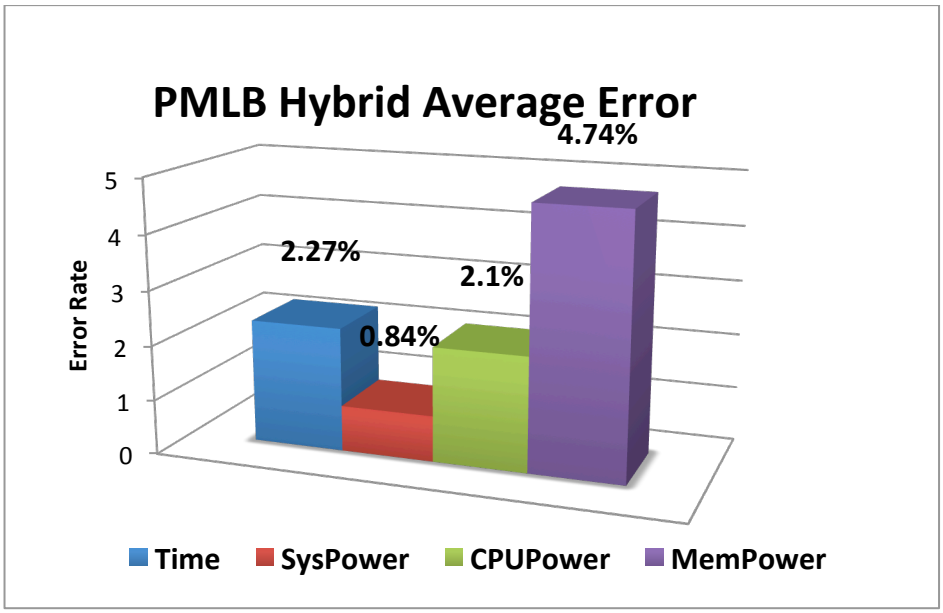

Figure 58. Average Error of PMLB Hybrid

With regards to modeling of the CPU power consumption, the PAPI_L1_TCA and PAPI_L2_TCA counters have the largest regression coefficients. For modeling the memory power consumption of the PMLB hybrid application, the PAPI_L2_TCH and PAPI_RES_STL counters have the largest regression coefficients of the hardware

counters. It is interesting to note that the PAPI_TOT_INS counter is used in the modeling of all of the performance components for the PMLB hybrid application.

Figure 58 shows the average error resulting from the modeling of the PMLB Hybrid application. The performance components modeled for the PMLB Hybrid application had an average prediction error of 2.5%. Specifically, the smallest error was found for predicting the system power consumption (0.84%) for the application. On the other hand, the highest error rate occurred in modeling the memory power consumption (4.74%).

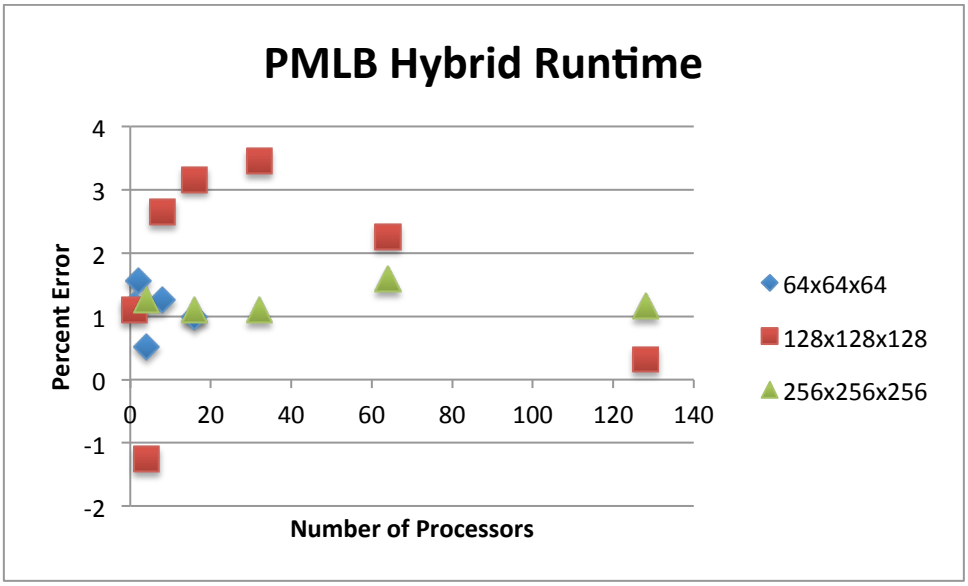

Figure 59. Scatterplot of PMLB Hybrid for Runtime

In Figure 59 we provide an overview of the distribution of each prediction point for the runtime of PMLB Hybrid application. The performance counters used for predicting runtime are able to provide for consistent values for predicting the runtime

across three application inputs, which include the grid sizes of 64x64x64, 128x128x128, and 256x256x256. Predictions across all application inputs for the PMLB hybrid application are largely positive.

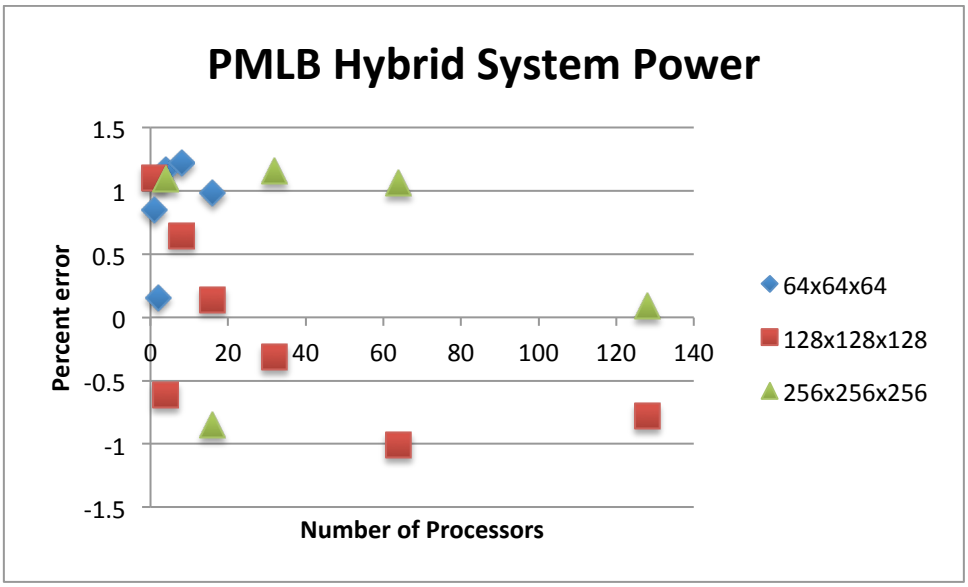

Figure 60. Scatterplot of PMLB Hybrid for System Time

Figure 60 shows the predicted values for the system power consumption for the PMLB hybrid application. The values predicted across the three input data sizes remain in the range of +2.0% to -2.0% error. As the number of processors increases, the predicted values are more negative in nature for our larger data sizes. This might be an indication that our model, which makes use of the PAPI_L1_TCA, is not providing the strongest estimate for the larger data sizes that might not make sure of the L1 cache.

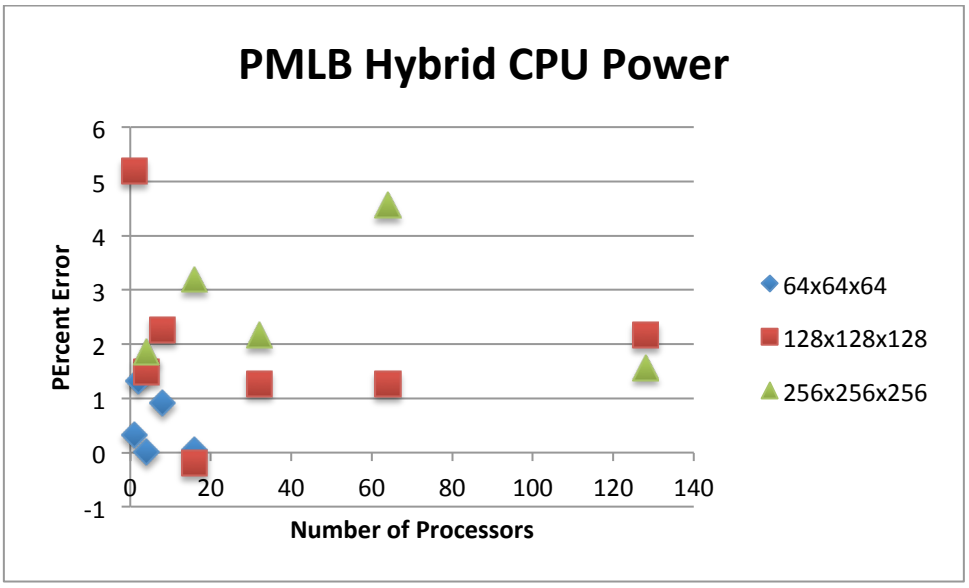

Figure 61. Scatterplot of PMLB Hybrid for CPU Power

Figure 61 and Figure 62 show the values associated with our model of PMLB hybrid application for predicting CPU and memory power consumption. The values associated with the prediction for these two performance components show that our models consistently predict power consumption for CPU and memory components in the range of +2.0% to -2.0%. There are slight increases in the error rate for both CPU and memory power consumption for processor sizes smaller than 8 processors. These similar trends are largely caused by the use of both PAPI_TOT_INS and PAPI_L2_TCH for both CPU and memory power consumption components.

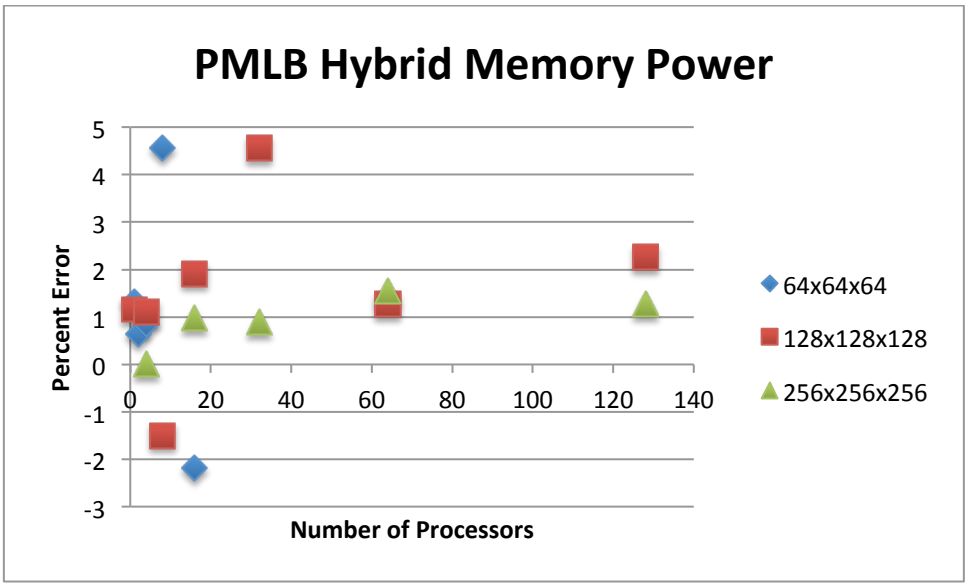

Figure 62. Scatterplot of PMLB Hybrid for Memory Power

4.3.5.2 MPI

In this section, we use our performance-tuned principal component analysis method to develop accurate models for the MPI implementation of the large-scale PMLB application. Table 27 shows the regression coefficients that are needed to accurately model each component. With regards to runtime, PAPI_L2_TCA and PAPI_TOT_INS have the largest regression coefficients that are used in modeling the runtime for the PMLB MPI application. System power shows that PAPI_RES_STL and PAPI_L1_TCA have the largest regression coefficients in modeling the system power consumption for the hybrid implementation of the application.

Table 27. Regression Coefficients for PMLB MPI

Time		System Power		CPU Power		Memory Power	
Frequency	0.004519	**Frequency**	1.30562	**Frequency**	0.915316	**Frequency**	0.064518
PAPI_TOT_INS	0.0231894	**PAPI_TOT_INS**	0.21849	**PAPI_TOT_INS**	0.618498	**PAPI_TOT_INS**	0.415152
PAPI_L1_TCA	0.0032198	**PAPI_L1_TCA**	1.76156	**PAPI_L2_TCH**	0.478156	**PAPI_L2_TCH**	0.19583
PAPI_L1_ICA	0.0041519	**PAPI_L2_TCA**	1.18982	**LD_ST_Stall**	1.315619	**PAPI_RES_STL**	0.04052
PAPI_L2_TCA	0.0518918	**PAPI_RES_STL**	1.91816				

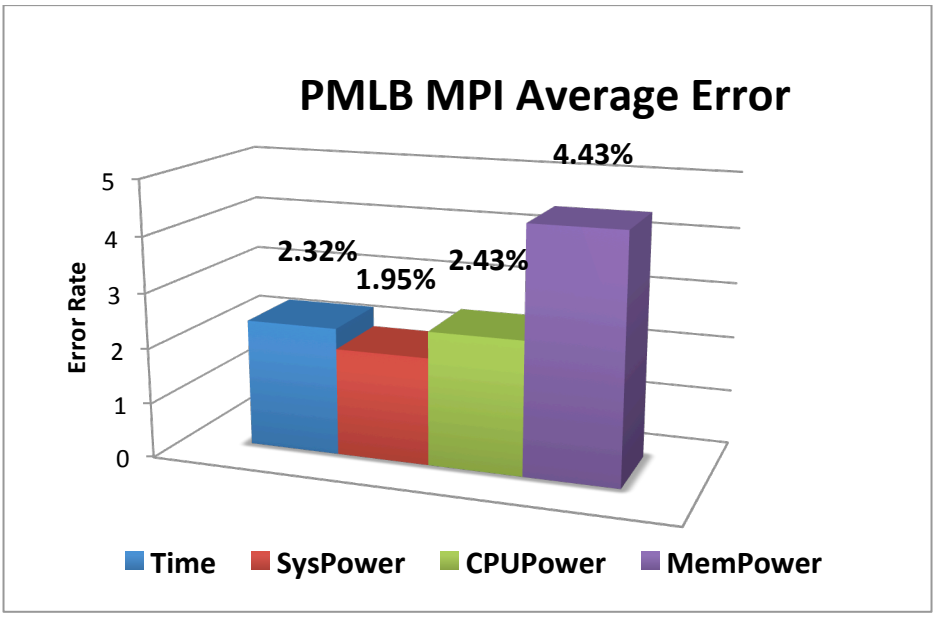

Figure 63. Average Error of PMLB MPI

With regards to modeling of the CPU power consumption, the LD_ST_stall_per_cycle and PAPI_TOT_INS counters have the largest regression coefficients. For modeling the memory power consumption of the PMLB MPIapplication, the PAPI_TOT_INS and PAPI_L2_TCH counters have the largest

regression coefficients of the hardware counters. It is interesting to note that the PAPI_TOT_INS counter is used in the modeling of all of the performance components for the PMLB MPI application.

Figure 63 shows the average error resulting from the modeling of the PMLB MPI application. The performance components modeled for the PMLB MPI application had an average prediction error of 2.78%. Specifically, the smallest error was found for predicting the system power consumption (1.95%) for the application. On the other hand, the highest error rate occurred in modeling the memory power consumption (4.43%).

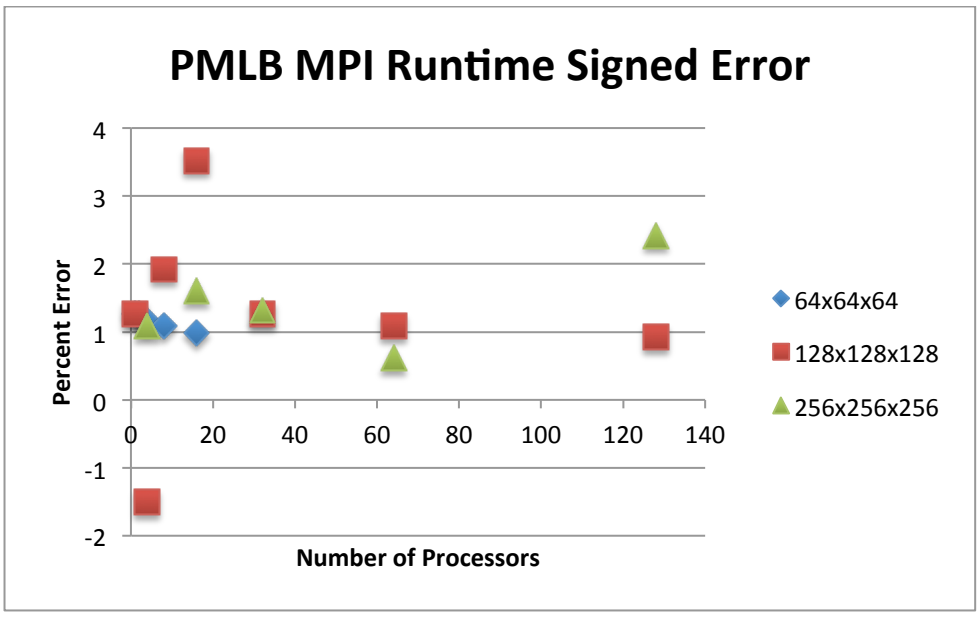

Figure 64. Scatterplot of PMLB MPI for Runtime

In Figure 64 we provide an overview of the distribution of each prediction point for the runtime of PMLB MPI application. The performance counters used for predicting runtime are able to provide for consistent values for predicting the runtime across three application inputs. Predictions across all application inputs for the PMLB MPI application are largely positive except for one prediction point in the case of 128x128x128.

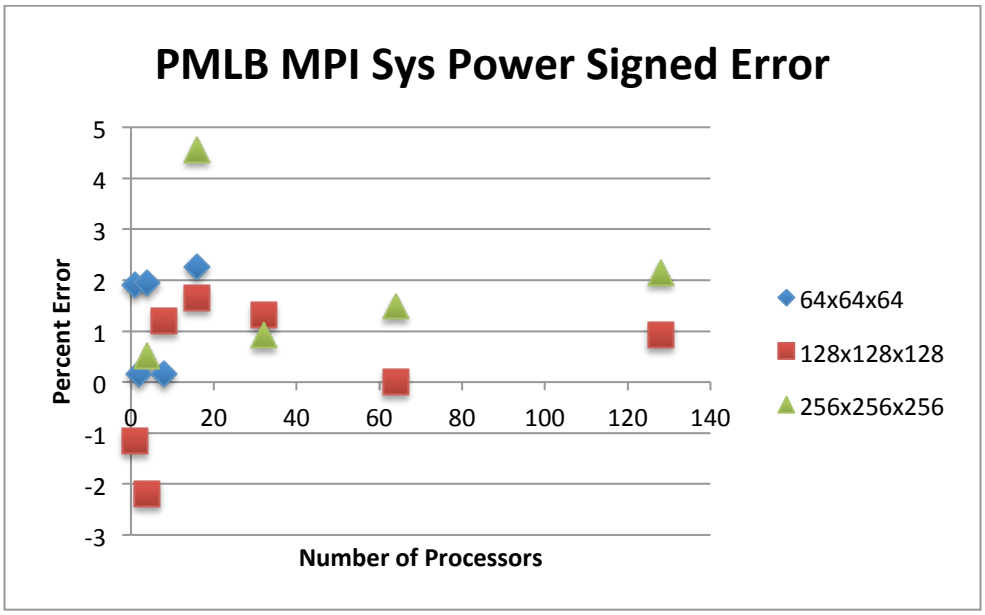

Figure 65. Scatterplot of PMLB MPI for System Power Consumption

Figure 65 shows the predicted values for the system power consumption for the PMLB MPI application. The percent error for the predictions across the three input data sizes remain in the range of +2.0% to 0% error, except for two outliers. For the input size of 128x128x128, there are only two points that are under-predicted. These negative

predict error values could be a result of not including a larger number of prediction points for smaller processors sizes in our application training set.

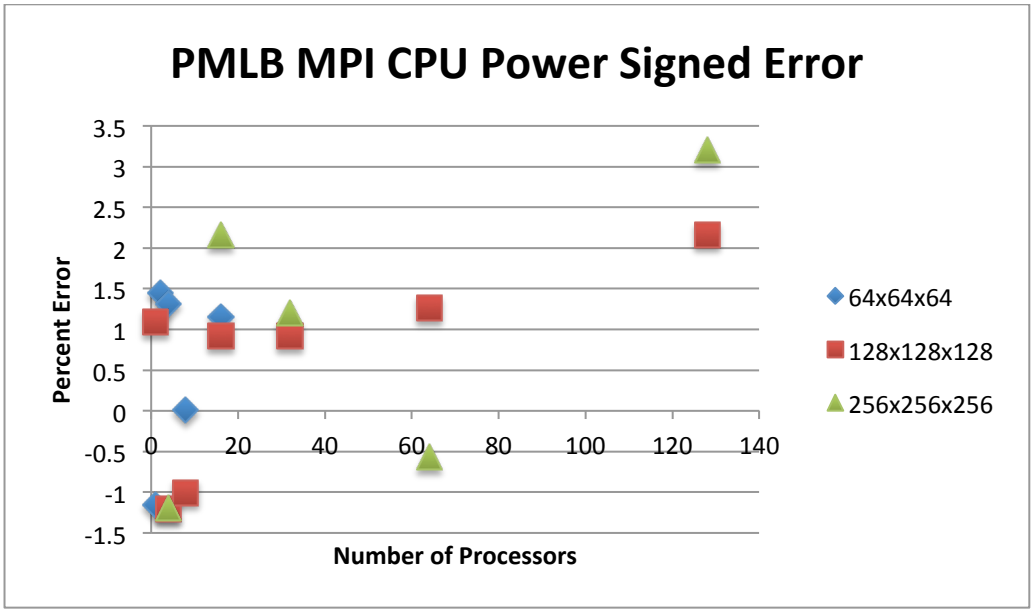

Figure 66. Scatterplot of PMLB MPI for CPU Power Consumption

Figure 66 and Figure 67 provide the predicted values for the CPU and Memory power consumption for the PMLB MPI application. The figures for both CPU and Memory power consumption show that for the smaller number of processors our modeling methodology under-predicts the power consumption of the application. This under-prediction is likely a result of the training set not including enough points to more accurately model the power consumption at these smaller processor configurations.

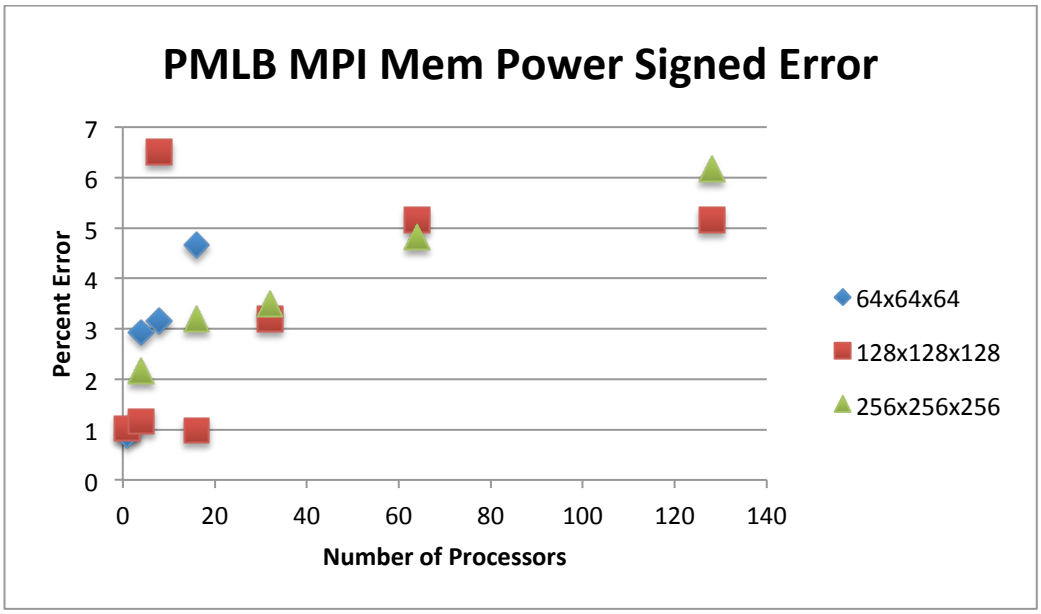

Figure 67. Scatterplot of PMLB MPI for Memory Power Consumption

4.3.6 Parallel EqDyna

4.3.6.1 Hybrid

In this section, we use our performance-tuned principal component analysis method to develop accurate models for the hybrid implementation of the large-scale EqDyna application. Table 28 shows the regression coefficients that are needed to accurately model each component. The PAPI_TOT_INS and PAPI_L2_TCH counters have the largest regression coefficients that are used in modeling the runtime for the EqDyna hybrid application. System power shows that PAPI_TOT_INS and PAPI_L1_TCA have the largest regression coefficients in modeling the system power consumption for the hybrid implementation of the application.

Table 28. Regression Coefficients for Parallel EqDyna Hybrid

Time		System Power		CPU Power		Memory Power	
Frequency	0.001953	**Frequency**	1.09547	**Frequency**	1.190092	**Frequency**	0.041202
PAPI_TOT_INS	0.038194	**PAPI_TOT_INS**	0.08749	**PAPI_TOT_INS**	0.158920	**PAPI_TOT_INS**	0.189290
PAPI_L1_TCA	0.015018	**PAPI_L1_TCA**	0.07938	**PAPI_L1_TCM**	0.078036	**PAPI_L2_TCA**	1.01223
PAPI_L2_TCH	0.021165	**PAPI_L2_TCA**	0.42389	**PAPI_L2_TCA**	0.910928	**PAPI_L2_TCH**	0.91284
PAPI_L2_TCA	0.041248	**PAPI_L2_TCH**	0.59384			**PAPI_RES_STL**	0.45298

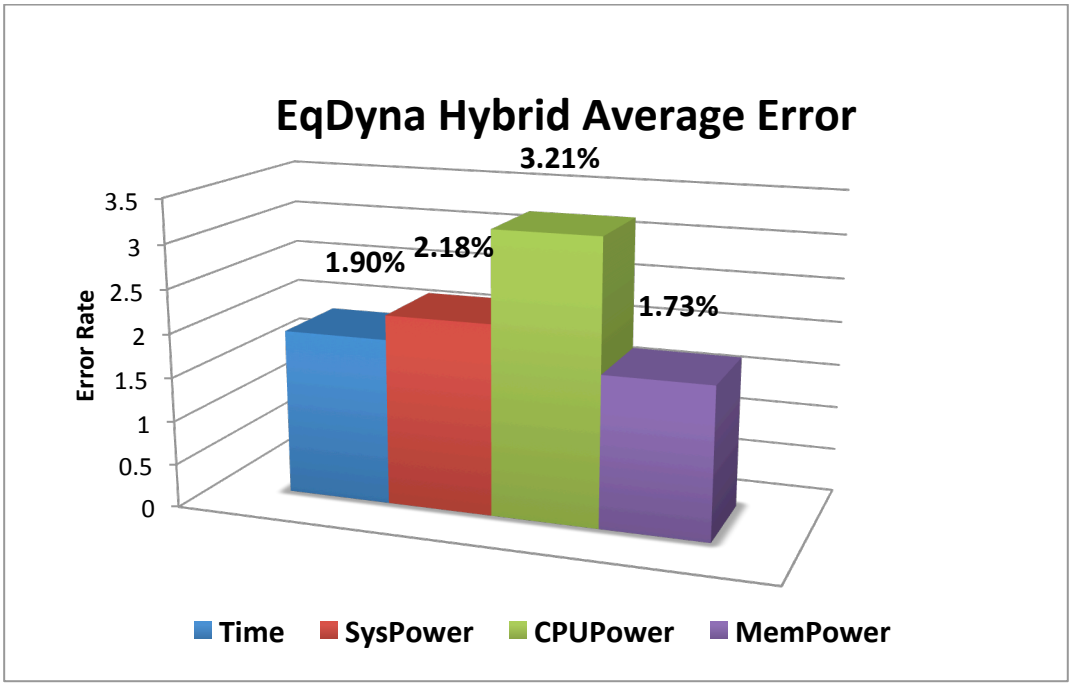

Figure 68. Average Error of Parallel EqDyna Hybrid

With regards to modeling of the CPU power consumption, the PAPI_L2_TCA and PAPI_TOT_INS counters have the largest regression coefficients. For modeling the memory power consumption of the EqDyna hybrid application, the PAPI_L2_TCA and PAPI_L2_TCH counters have the largest regression coefficients of the hardware

counters. For the EqDyna hybrid application, it is interesting to note that the PAPI_TOT_INS and PAPI_TOT_TCA counters are used in the modeling of all of the performance components for the EqDyna hybrid application.

Figure 68 shows the average error resulting from the modeling of the EqDyna hybrid application. The performance components modeled for the EqDyna hybrid application had an average prediction error of 2.26%. Specifically, the smallest error was found for predicting the memory power consumption (1.73%) for the application. The highest error rate occurred in modeling the CPU power consumption (3.21%).

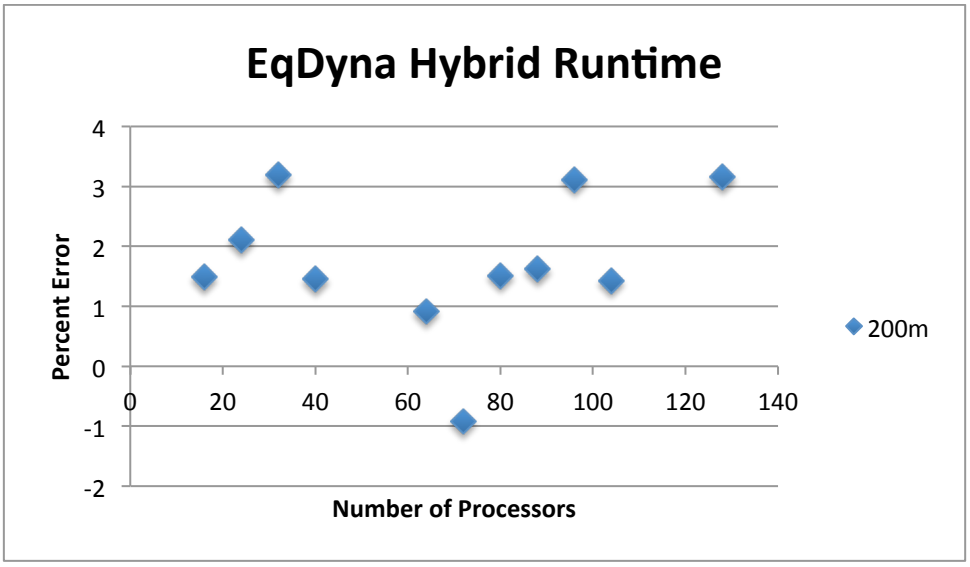

Figure 69. Scatterplot of EqDyna Hybrid for Runtime

In Figure 69 we provide an overview of the distribution of each prediction point for the runtime of EqDyna hybrid application. The performance counters used for predicting runtime are able to provide for consistent values for predicting the runtime

across this application. Predictions across all application inputs for the EqDyna hybrid application are largely positive except for one prediction point for this application.

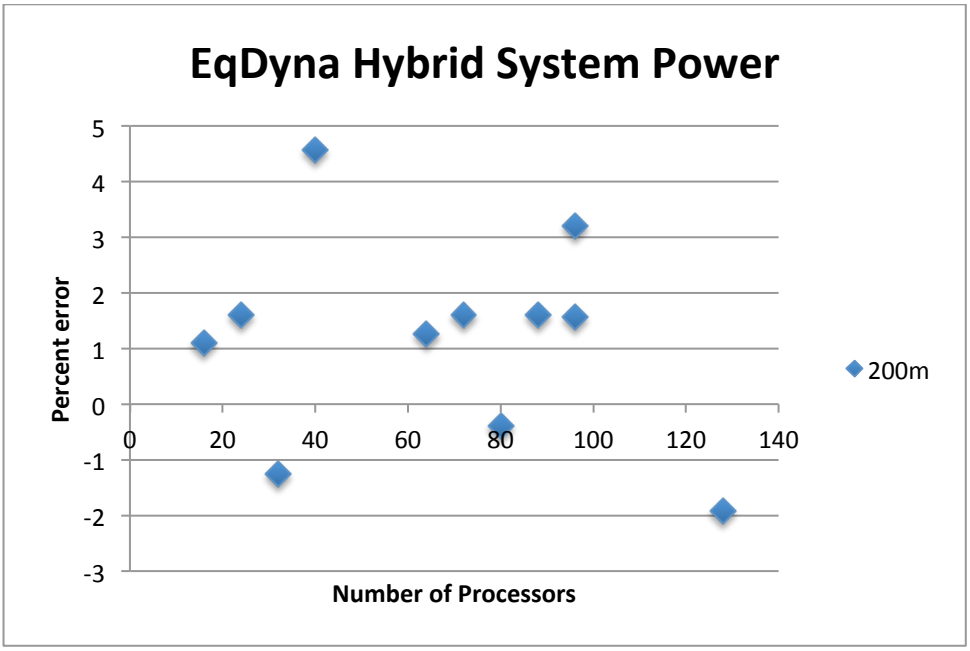

Figure 70. Scatterplot of EqDyna Hybrid for System Power Consumption

Figure 70 shows the predicted values for the system power consumption for the EqDyna hybrid application. The predictions values across the application show a strong concentration in the range of +2.0% to 0% error, with a few outliers. The outliers shown in the scatterplot are likely a result of changes in the trends for the performance counters based on the workload distribution.

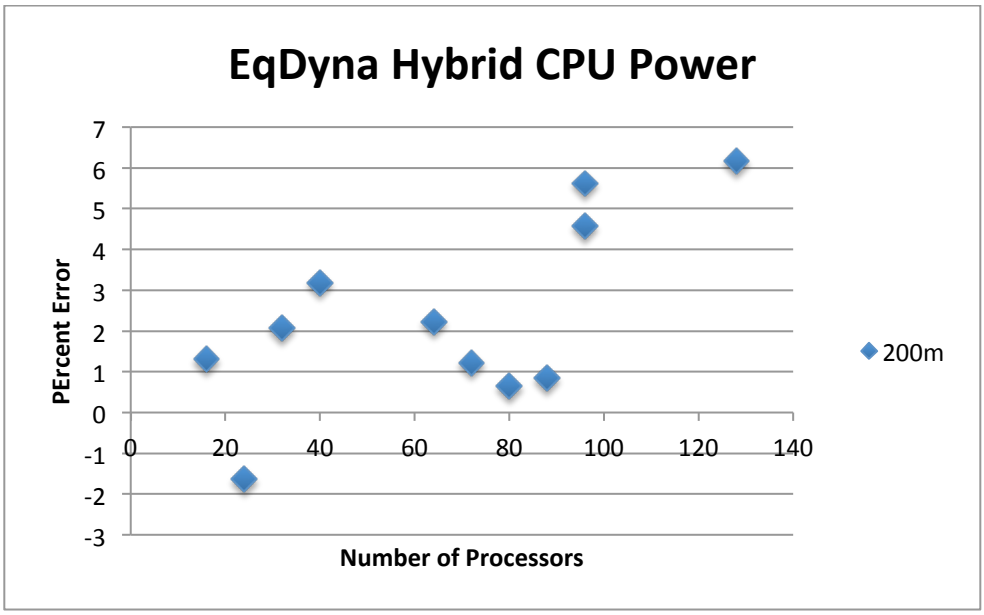

Figure 71. Scatterplot of EqDyna Hybrid for CPU Power Consumption

Figure 71 and Figure 72 provide the predicted values for the CPU and Memory power consumption for the EqDyna application. The figures for both CPU and Memory power consumption show that our modeling methodology predicts the power consumption of the application within a range of +4% to 0% as the number of processors increase. There are a few points in which our model slightly over predicts or under-predicts, however, this error is no larger than 2% of the predicted range for the majority of the data points.

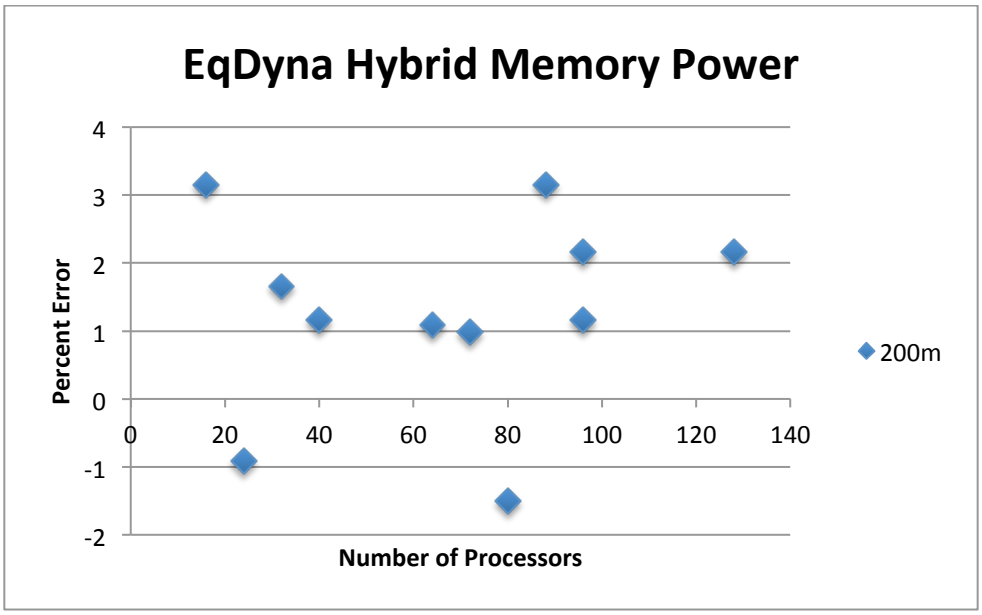

Figure 72. Scatterplot of EqDyna Hybrid for Memory Power Consumption

4.3.6.2 MPI

In this section, we use our performance-tuned principal component analysis method to develop accurate models for the MPI implementation of the large-scale EqDyna application. Table 29 shows the regression coefficients that are needed to accurately model each component. The PAPI_L1_TCM and PAPI_L2_TCH counters have the largest regression coefficients that are used in modeling the runtime for the EqDyna MPI application. System power shows that PAPI_L2_TCA and PAPI_L2_TCH have the largest regression coefficients in modeling the system power consumption for the MPI implementation of the application. With regards to modeling of the CPU power consumption, the PAPI_TOT_INS and PAPI_L2_TCH counters have the largest regression coefficients. For modeling the memory power consumption of the EqDyna

MPI application, the PAPI_L2_TCH counter has the largest regression coefficient of the hardware counters. For the EqDyna MPI application, it is interesting to note that the PAPI_TOT_INS and PAPI_L2_TCH counters are used in the modeling of all of the performance components for the EqDyna MPI application.

Table 29. Regression Coefficients for EqDyna MPI

Time		System Power		CPU Power		Memory Power	
Frequency	0.0037213	**Frequency**	0.07983	**Frequency**	0.843792	**Frequency**	0.100675
PAPI_TOT_INS	0.18344	**PAPI_TOT_INS**	0.24029	**PAPI_TOT_INS**	1.12323	**PAPI_TOT_INS**	0.18929
PAPI_L1_TCA	0.06432	**PAPI_L1_TCM**	0.17338	**PAPI_L1_TCM**	0.243472	**PAPI_L1_TCA**	0.23559
PAPI_L1_TCM	0.218925	**PAPI_L2_TCA**	0.50729	**PAPI_L2_TCA**	0.562903	**PAPI_L2_TCH**	0.56545
PAPI_L2_TCH	0.492188	**PAPI_L2_TCH**	0.41156	**PAPI_L2_TCH**	0.342892	**PAPI_RES_STL**	0.218375

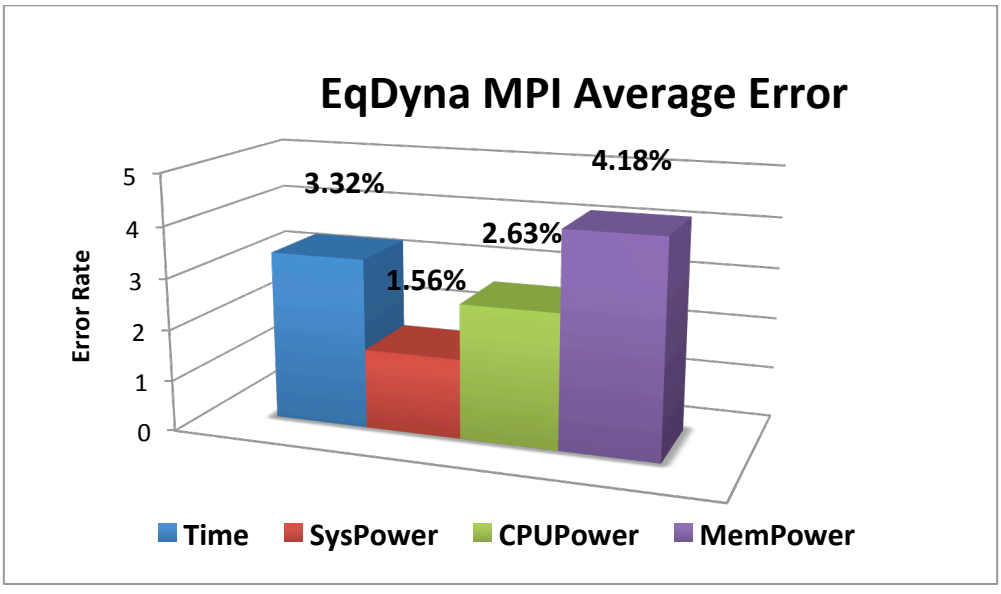

Figure 73. Average Error of EqDyna MPI

Figure 73 shows the average error resulting from the modeling of the EqDyna MPI application. The performance components modeled for the EqDyna MPI application had an average prediction error of 2.92%. Specifically, the smallest error was found for predicting the system power consumption (1.56%) for the application. On the other hand, the highest error rate occurred in modeling the memory power consumption (4.18%).

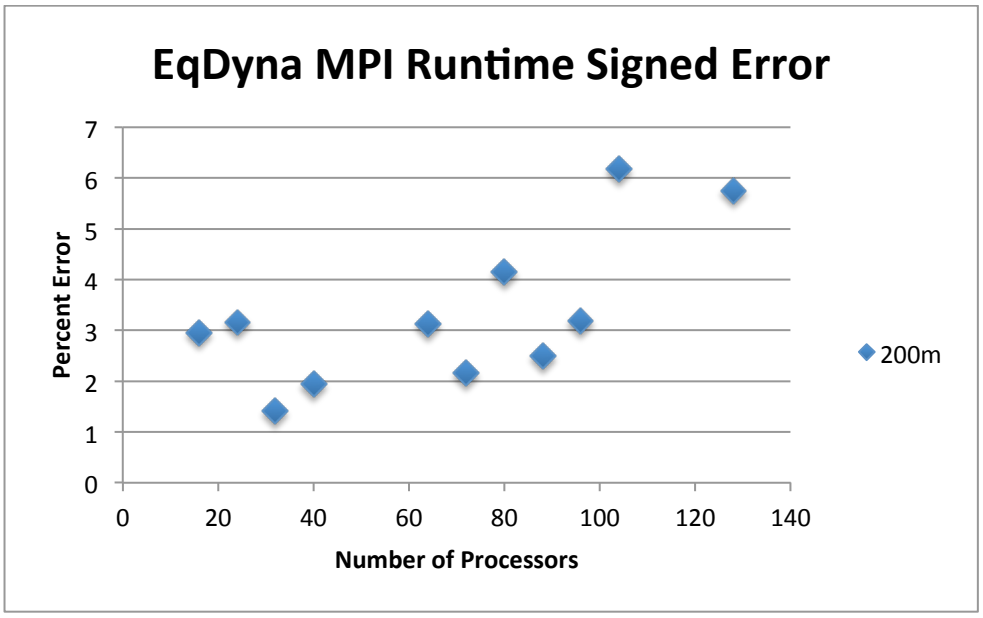

Figure 74. Scatterplot of EqDyna MPI for Runtime

Figure 74 and Figure 75 provide an overview of the distribution of each prediction point for the runtime and system power consumption of the EqDyna MPI application. The performance counters used for predicting runtime are able to provide for consistent values for predicting the runtime across the application as the number of

processors increase within the range of +4% to +1%. Predictions across all application inputs for the EqDyna MPI application for system power consumption are largely positive with a few negative error predictions that occur for the smaller processor configurations. These negative percent error predictions are no larger than 3% as the number of processor increases. The cause of the negative percent error predictions is likely the application training set not predicting well at those processor configurations.

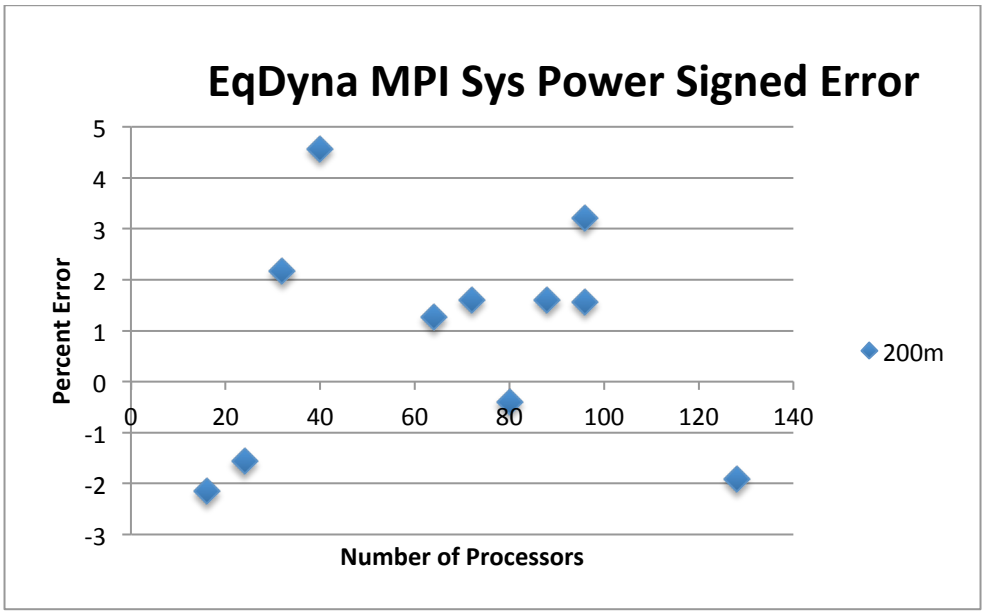

Figure 75. Scatterplot of EqDyna MPI for System Power Consumption

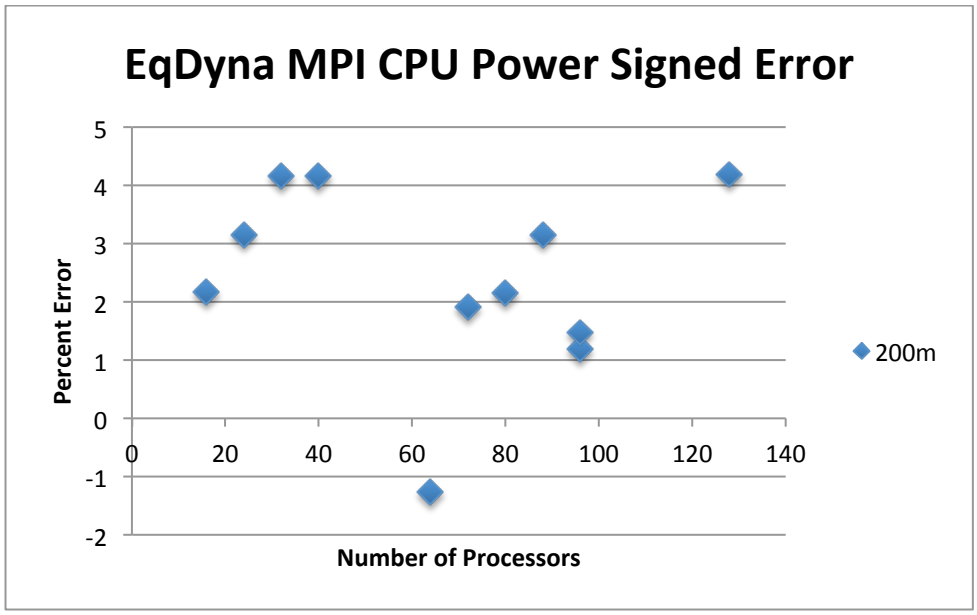

Figure 76. Scatterplot of EqDyna MPI for CPU Power Consumption

Figure 76 and Figure 77 provide the predicted values for the CPU and Memory power consumption for the Parallel EqDyna MPI application. The figures for CPU power consumption show that our modeling methodology predicts the power consumption of the application within a range of +4% to 0% as the number of processors increase. In the case of the memory power consumption there are clusters of points that show positive percent error for the predictions and as the application predicts for larger than 72 processors there are negative percent error predictions. For the larger number of processors the negative percent error is no larger than 3%, which shows a good prediction error.

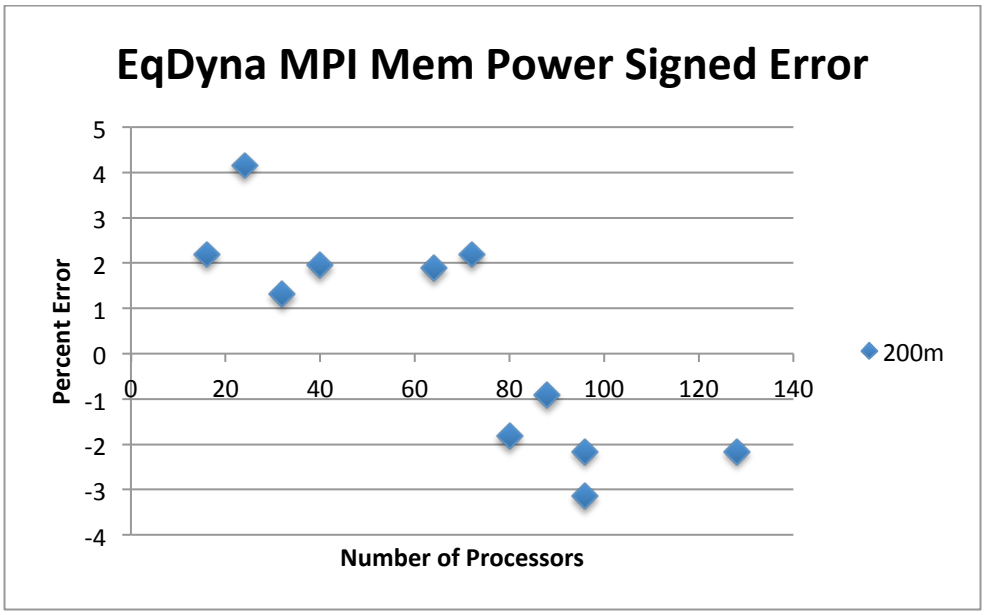

Figure 77. Scatterplot of EqDyna MPI for Memory Power Consumption

4.4 Summary

We presented a modeling scheme for developing predictive performance models to analyze the performance characteristics of Hybrid and MPI scientific applications in terms of runtime, system power, CPU power, and memory power. The predictive models are able to determine the performance characteristics that affect each respective performance component. Most importantly, our method identifies the different performance counter measurements that are needed to accurately predict application performance and provide insight to improve performance for each application.

Our models make use of the Multicore Application Modeling Infrastructure, MuMI, which utilizes Prophesy, PowerPack, and PAPI to provide systematic measurement, and modeling of power consumption and performance-power tradeoffs on

multicore systems. Our predictive models are +92% accurate across six hybrid and MPI scientific applications for up to 128 processors and can be used to obtain insight into improving applications for better performance on multicore systems. Using our predictive models the performance of a scientific application can be predicted across different frequency configurations and for different application inputs.

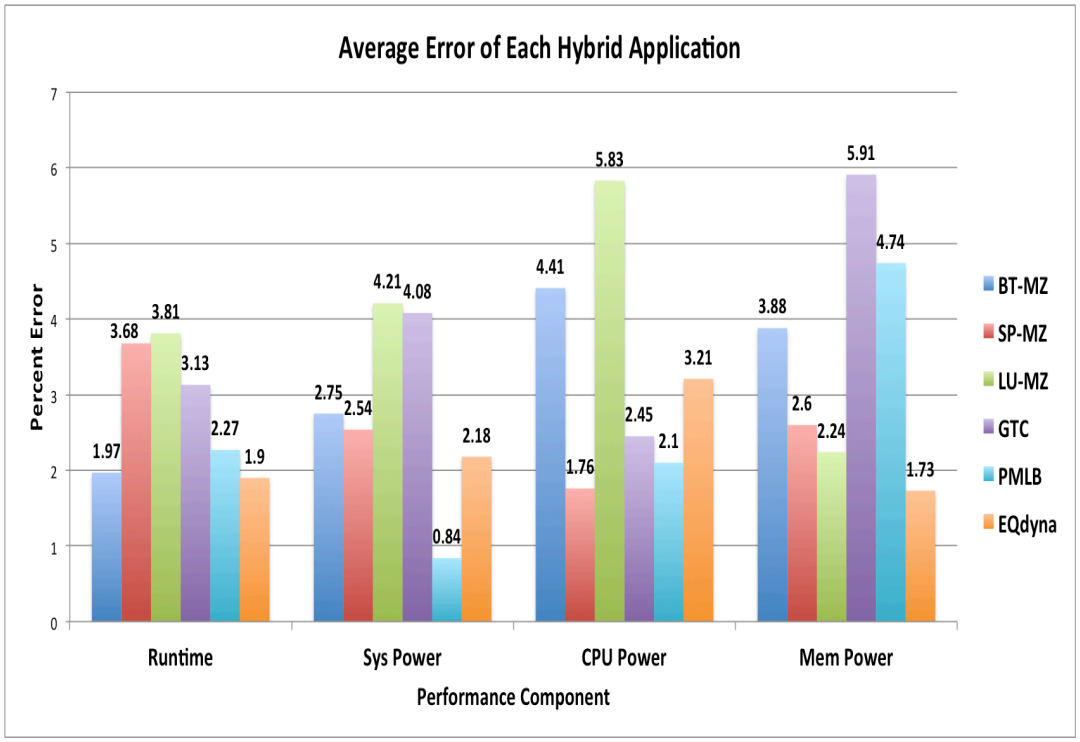

Figure 78. Average Error of All Hybrid Applications

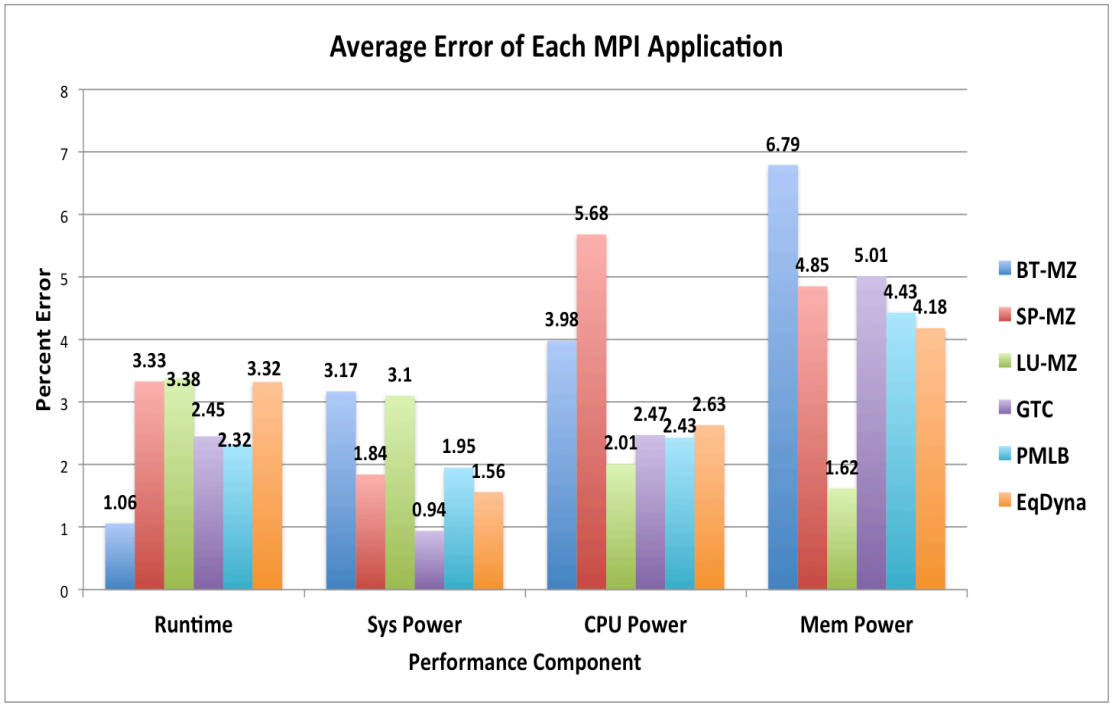

Figure 79. Average Error of All MPI Applications

Our models also are able to identify the commonalities found across the applications in regards to using the same performance counters. For example, the BT-MZ, SP-MZ, LU-MZ, and GTC use the PAPI_TOT_INS and PAPI_L2_TCH counters for modeling runtime. For modeling the system power consumption the PAPI_L2_TCH counter is used for BT-MZ, SP-MZ, LU-MZ, GTC, and PMLB. Identifying the common characteristics of these applications indicates that optimizations made to utilization of the L2 cache should be taken into consideration for reducing runtime and improving power consumption. In Section 5, we will utilize the performance models presented in this section to make optimizations to the application to reduce runtime and reduce power consumption.

5. OPTIMIZATION OF HYBRID AND MPI SCIENTIFIC APPLICATIONS

In Section 3, we provided the motivation for exploring performance, power, and energy tradeoffs across different implementations of scientific applications. Section 4 introduced the E-AMOM modeling methodology utilizing the performance-tuned principle component analysis method and experimental models of scientific applications. In this section, we present the E-AMOM optimization methodology and results for our hybrid and MPI scientific applications.

5.1 Software-Based Power Reduction Methods

As large-scale parallel systems continue to evolve and incorporate additional cores onto chips additional methods must be used to reduce performance and power consumption. In this section, we incorporate two common software-based approaches for reducing power consumption in large-scale parallel systems, Dynamic Voltage and Frequency Scaling (DVFS) [30][36] and Dynamic Concurrency Throttling (DCT)[19].

DVFS can be used to scale down the frequency of a HPC application's workload resulting in lower power consumption. DVFS is most beneficial when applied to regions within an application where communication does not overlap with computation. When DVFS is applied to applications that exhibit slack (lack of overlap between

communication and computation) it results in reduced power consumption with minimal increases in application performance.

Additionally, dynamic concurrency throttling (DCT) is used to control the number of threads assigned to a segment of a parallel code. DCT can be applied to a code region with a reduced workload that would not benefit from using the maximum number of cores on a chip. When applying DCT to an application executed on SystemG we look at the settings of having 1, 2, 4, 6, or 8 threads executing per node. Depending on the application and workload requirements of the application utilizing fewer threads can reduce power consumption without impacting performance significantly.

The result of applying DCT effectively results in reduced energy consumption with minimal increases in application performance for thread-based applications, such as OpenMP or hybrid applications. In this section, we use our performance-tuned principal component analysis method to make optimizations to the application kernels of scientific applications.

5.2 Optimization Methodology for Application Kernels

In our work we adjust the configurations of our HPC application kernels with regards to the number of OpenMP threads used for the hybrid applications dynamically. We also lower the frequency of the kernels to reduce power consumption. We define the configuration for a given application run to include the CPU frequency setting and concurrency configuration of the application kernel.

For estimating the appropriate DCT setting for each application kernel we evaluate the performance of the kernel given the application model developed in Section 4. The performance with regards to runtime and system power consumption is predicted using the performance models for each application presented in Section 4. The performance of kernel *i* is predicted using the multivariate linear regression equation:

$$K_{comp_{i_f}-mxn} = \beta_0 + \beta_{fq} * r_{fq} + \beta_i * r_i + \cdots + \beta_n * r_n \quad (12)$$

where coefficients to account for the frequency (β_{fq}) and each performance counter (β_i) are included.

To predict the expected outcome of a performance counter event rate during a kernel's execution the following multivariate regression equation is utilized for each performance counter needed:

$$K_{counter\_i\_f\_mxn} = \beta_0 + \beta_{nde} * r_{nde} + \beta_{tds} * r_{td} \quad (13)$$

where the terms correspond to coefficients to account for the intercept (β_0), number of nodes (β_{nde}), number of threads or MPI tasks per node (β_{tds}), and number of instructions per cycle (β_{ins}). The number of instructions per cycle is recorded for each kernel using the application's training set.

In predicting the performance of each application component (runtime or system power consumption), equation (14) represents the relationship for each kernel in the scientific application:

$$P_{total} = \sum_{i=0}^{n-1} K_i \quad (14)$$

where P represents the performance of the application and K_i represents the performance of kernel *i*. The performance of the application is represented in terms of runtime and power consumption. The sum of each of these kernels represents the performance for the application. Figure 73 presents an overview of the methodology that is used to determine the application configuration that can be used incorporating DCT and DVFS to improve application power consumption.

Our framework evaluates the performance of the application based on the following steps:

1. Take as input a given Hybrid or MPI-only HPC application.
2. Develop the performance model of each application kernel in terms of runtime and system power consumption.
3. Determine the appropriate configuration settings
    a. DVFS Setting
        i. Compute expected power consumption and execution time at lower frequency settings.
        ii. If frequency setting results in a 10% saving in power consumption, without increasing runtime more than 3% then use reduced frequency.
    b. DCT Setting
        i. Compute expected power consumption and execution time at concurrency settings using 1, 2, 4, and 6 threads.

ii. Identify the concurrency setting that enables the best saving in power consumption and runtime.

4. Determine the total application runtime and system power consumption including synchronization overhead costs from changing application settings using α_i.
5. Use new configuration settings for running application.

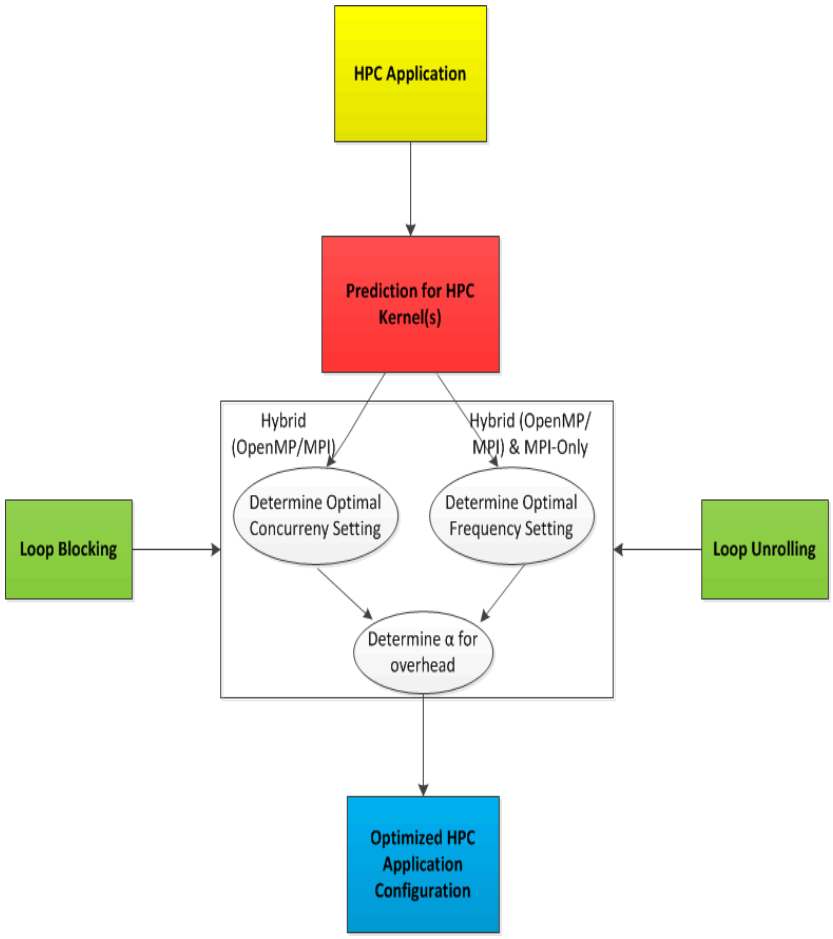

Figure 80. Overview of Optimization Scheme

Figure 80 provides an overview of the optimization scheme utilized in this chapter. We make use of the modeling methodology presented in section 4 to model the kernel performance of each application in terms of execution time, and system power consumption. Figure 81 provides a depiction of an example application's control flow. The typical scientific application consists of an initialization and final kernel, with computational kernels within a timestep loop.

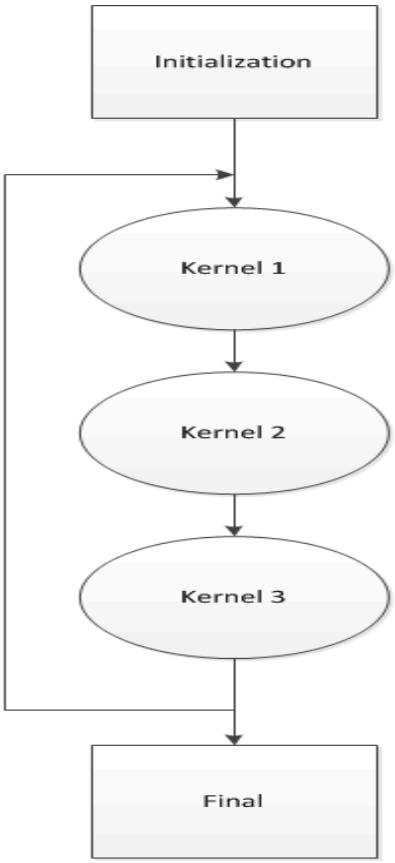

Figure 81. Example Application Control Flow

In predicting the performance of each application component considering the determined configuration from our modeling framework, equation (15) represents the expected execution time for each kernel and the synchronization overhead costs that would be incurred from lowering and increasing the frequency of the kernel in the HPC application:

$$P_{total\_optimized} = \sum_{i=0}^{n-1}(K_{ti} + \alpha_i) \quad (15)$$

We utilize our multivariate linear regression equation presented in Section 4 to determine the appropriate configuration based on frequency and number of threads. The frequency, number of nodes, and threads per node, are incorporated into the regression equation with the performance counters to predict the performance of the application kernel at two frequency settings (2.4 Ghz and 2.8 Ghz) and at concurrency settings of 1, 2, 4, 6, and 8 threads. Figure ** shows how the performance of each kernel within the application is predicted using the performance counters for each performance components (runtime and system power). The performance for each kernel is predicted to determine if a 10% saving in power can be achieved without increasing the runtime more than 3%. If a kernel is not able to achieve 10% reduction in power with no more than 3% runtime increase then the original configuration setting for that kernel is used. A decrease in power consumption greater than 10% provides for a measurable improvement that cannot be attributed to error or system noise. Also, a runtime increase less than 3% does not largely affect the runtime of the application, which means that a reduction in power can be obtained without a large increase in runtime.

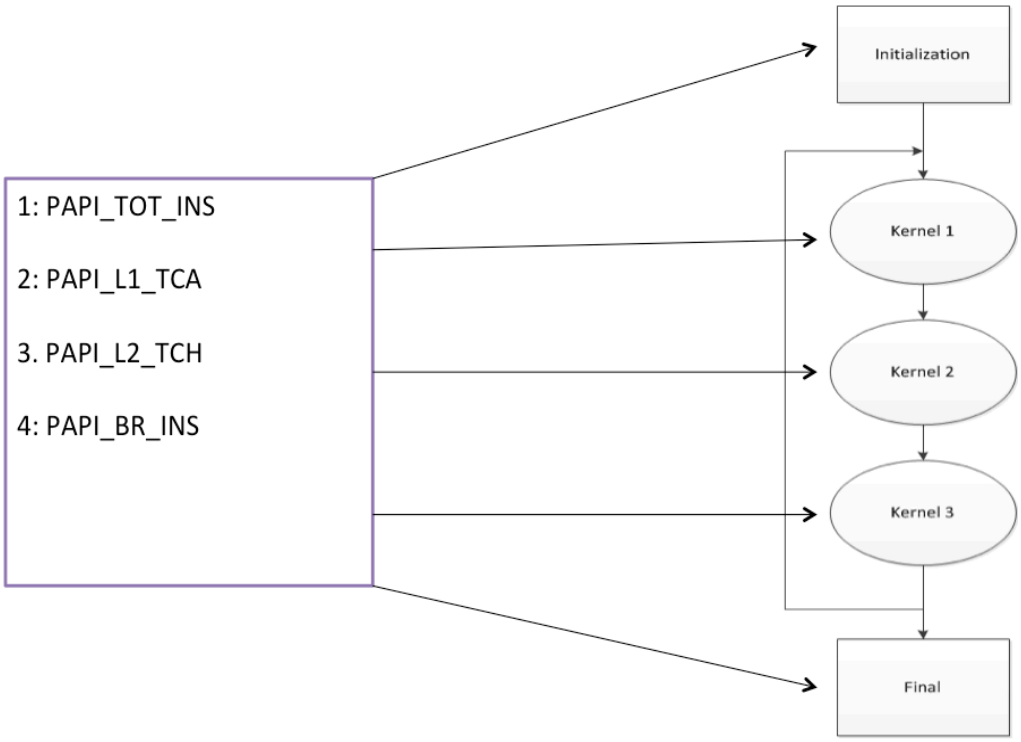

Figure 82. Per Kernel Predictions for Applications

We utilize the following equation to approximate the expected average power consumption of the application when applying DVFS and DCT to reduce application performance:

$$P_{sys\_power} = \frac{\sum_{i=0}^{n-1}(K_{sysp\_i})}{n-1} \quad (16)$$

where K_{sysp_i} represents the predicted system power consumption of kernel *i* and n is the number of kernels in the application. Energy consumption of a HPC application is modeled based on energy being the product of power and time as presented in Equation 6 in Section 3.

We have determined the following scenarios in which it would be appropriate to apply changes to configuration settings in our applications:

1. DVFS and DCT changes to specific kernels in the application
2. DVFS-only applied to specific kernels in the application
3. DCT-only applied to specific kernels in the application
4. DVFS applied to a limited number of time-steps within an application.

We use E-AMOM models to determine predicted runtime and power consumption for the application kernels based on equation 1. An application kernel that is able to provide for a power savings of at least 10% with a limited increase in runtime (less than 4%) will be executed at the new refined configuration.

5.3 Loop Optimizations

Based upon models developed with E-AMOM, we identify the type of algorithmic changes that should be applied to our HPC applications to improve utilization of the memory hierarchy. Much of the computation involved in the kernels of HPC applications occurs within nested loops. Therefore, loop optimization is fundamentally important for such applications. In this section, we discuss how loop blocking and loop unrolling can be used to optimize the performance of HPC applications.

Loop blocking is a well-known loop optimization technique to aid in taking advantage of memory hierarchy [47][60]; its main purpose is to eliminate as many cache misses as possible. This technique transforms the memory domain of an application into smaller chunks, such that computations are executed on the chucks that easily fit into

cache to maximize data reuse. The optimal loop block size varies with different applications on different systems. In this work, we apply the following loop block sizes: 2x2, 4x4, 8x8 and 16x16 to our HPC applications to measure which loop block size is optimal. To determine the best block size for each application we measure the performance of the application for each block size using a reduced number of iterations to approximate the best block size. Previous work [68] has identified these block sizes as optimal sizes for achieving performance improvements within scientific applications. Future work will determine how larger block sizes might affect application performance.

Loop unrolling is a well-known code transformation technique that replicates the original loop body multiple times, adjusts the loop termination code and eliminates redundant branch instructions. Outer loop unrolling can increase computational intensity and minimize load/stores, while inner loop unrolling can reduce data dependency and eliminate intermediate loads and stores.

5.4 Experimental Results

To improve the performance of our hybrid and MPI HPC applications we apply the methodology outlined in Section 5.2. Our methodology is applied to six HPC applications: the NAS Multizone benchmarks (BT-MZ, SP-MZ, LU-MZ), GTC, PLMB, and EqDyna. For each of the applications we present the execution time, energy consumption per node, and average system, CPU, and memory power consumption. The results show performance and power reductions across both implementations of the applications and identify which application provides for the best energy savings. Figure

83 provides an overview of how E-AMOM is used to optimize the applications by determining appropriate configurations for each kernel. This figure illustrates that initialization, kernel 1, and the final kernel have DVFS and DCT applied to them at settings of 2.4Ghz and 2 threads. Kernel 1, kernel 2, and kernel 3 have loop optimizations applied to them to improve performance. For each application different configurations are used that represent improved performance. In this section the configurations used are presented before the optimization results.

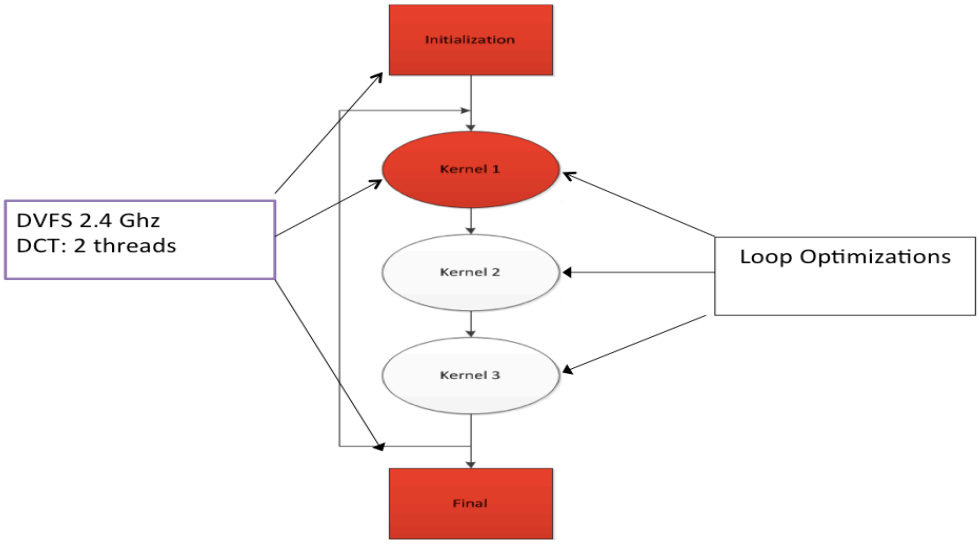

Figure 83. Applying Optimizations to an Application

5.4.1 BT-MZ

Applying DVFS and DCT to select application kernels to reduce power consumption and execution time for the application improves the performance of the

hybrid NAS BT-MZ. During the initialize solutions kernel and the exchange of boundary conditions, which contains significant MPI communication, DVFS is applied. DCT is applied during the BT solver kernel reducing the power consumption during this phase for an optimal configuration using 4 threads. Loop optimizations are applied to class C (block size = 4x4) and class D (block size = 4x4). Table 30 and Table 31 present the performance results for the Hybrid BT-MZ application for Class C and Class D.

Table 30. Performance of Hybrid BT-MZ Application and Optimization (Class C)

#Cores	BT-MZ Type	Runtime(s)	Total Energy (KJ)	Total Power (W)
1x8	Hybrid	231.88	80.921	348.98
	Optimized-Hybrid	217.19 (-6.78%)	68.901 (-17.45%)	317.24 (-10.00%)
2x8	Hybrid	115.83	40.356	348.41
	Optimized-Hybrid	107.19 (-4.49%)	34.559 (-16.77%)	322.41 (-8.1%)
4x8	Hybrid	58.04	20.222	347.91
	Optimized-Hybrid	52.12 (-11.35%)	17.032 (-18.75%)	326.79 (-6.46%)
6x8	Hybrid	38.80	13.507	348.11
	Optimized-Hybrid	35.11 (-10.5%)	11.342 (-19.1%)	323.03 (-7.76)
8x8	Hybrid	29.23	10.172	348.00
	Optimized-Hybrid	26.45 (-4.49%)	8.700 (-16.9%)	328.91 (-5.8%)
10x8	Hybrid	23	7.999	347.80
	Optimized-Hybrid	21 (-9.52%)	6.782 (-17.95%)	322.95 (-7.69%)

Table 31. Performance of Hybrid BT-MZ Application and Optimization (Class D)

#Cores	BT-MZ Type	Runtime(s)	Total Energy (KJ)	Total Power (W)
6x8	Hybrid	655	228.401	348.70
	Optimized-Hybrid	632 (-3.64%)	205.027 (-11.4)	324.41 (-7.49%)
8x8	Hybrid	493	171.573	348.73
	Optimized-Hybrid	440 (-12%)	14.1754 (-21.0%)	322.17 (-8.24%)
16x8	Hybrid	339	117.911	347.82
	Optimized-Hybrid	319 (-6.27%)	103.072 (-14.39%)	323.11 (-7.65%)
32x8	Hybrid	201	69.570	346.12
	Optimized-Hybrid	193 (-4.14)	62.902 (-10.60%)	325.92 (-6.2%)
64x8	Hybrid	119	41.325	347.27
	Optimized-Hybrid	112 (-6.25)	36.338 (-13.7%)	324.45 (-7.03%)

During the initialize solutions kernel and the exchange of boundary conditions, which contains significant MPI communication, DVFS is applied. Additional loop optimizations are applied to class C (block size = 2x2) and class D (block size = 4x4). Table 32 and Table 33 present the performance results for the hybrid BT-MZ application for Class C and Class D.

Table 32. Performance of MPI BT-MZ Application and Optimization (Class C)

#Cores	BT-MZ Type	Runtime(s)	Total Energy (KJ)	Total Power (W)
1x8	MPI	245	82861.45	338.21
	Optimized-MPI	232.83 (-12.17)	75481.18 (-9.78%)	324.19 (-4.32%)
2x8	MPI	127	42945.05	338.15
	Optimized-MPI	118 (-7.62%)	38024.32 (-12.94%)	322.24 (-4.93%)
4x8	MPI	65.83	22316.37	339
	Optimized-MPI	57.12 (-15.24)	18460.04 (-20.89%)	323.18 (-4.89%)
6x8	MPI	44.13	14595.11	330.7322
	Optimized-MPI	40.18 (-9.8%)	13010.69 (-12.17%)	323.81 (2.14%)
8x8	MPI	35.23	11520.7	327.014
	Optimized-MPI	32.19 (-9.41%)	10306.27 (-11.78%)	320.17 (2.14%)
10x8	MPI	27.38	9011.31	329.12
	Optimized-MPI	25.21 (-8.61%)	8021.56 (-12.34%)	318.19 (3.44%)

Table 33. Performance of MPI BT-MZ Application and Optimization (Class D)

#Cores	BT-MZ Type	Runtime(s)	Total Energy (KJ)	Total Power (W)
6x8	MPI	729	241103.76	330.7322
	Optimized-MPI	700 (-4.14%)	226667.17 (-6.36%)	323.81 (-2.14%)
8x8	MPI	545	178223.13	327.014
	Optimized-MPI	489 (-11.45%)	156563.19 (-13.83%)	320.17 (-2.14%)
16x8	MPI	387	117910.98	329.12
	Optimized-MPI	329 (-14.15%)	104684.51 (-12.63%)	318.19 (-3.44%)
32x8	MPI	233.14	76546.85	328.33
	Optimized-MPI	220.78 (-5.59)	68525.70 (-11.70)	310.38 (-5.72%)
64x8	MPI	138.58	44395.67	327.45
	Optimized-MPI	125.73 (-10.22%)	38703.47 (-14.71%)	307.83 (-6.37%)

The results of applying our methodology across the BT-MZ applications are able to show that savings in energy consumption can be obtained by reducing the execution time of the application and reducing power consumption. There is a trade-off between the savings achievable by the application in terms of reduced runtime and power consumption. For BT-MZ, the hybrid application provided for the best performance in terms of runtime and energy consumption across all data inputs.

5.4.2 SP-MZ

Applying DVFS and DCT to the application to reduce power consumption and workload requirements optimizes the hybrid NAS SP-MZ application. SP-MZ represents a fairly balanced workload. To reduce the frequency of the application during execution we apply DVFS to the initial solutions kernel and take the approach of reducing the application frequency for the first 100 time steps of the application kernel to limit the additional overhead that would be introduced from lowering the frequency.

Table 34. Performance of Hybrid SP-MZ Application and Optimization (Class C)

#Cores	SP-MZ Type	Runtime(s)	Total Energy (KJ)	Total Power (W)
2x8	Hybrid	117.82	37201.32	317.96
	Optimized-Hybrid	104.15 (-13.12)	31577.24 (-17.8%)	303.19
4x8	Hybrid	60	19111.9655	318.53
	Optimized-Hybrid	56 (-7.1%)	17120.32 (-11.63)	305.72
6x8	Hybrid	39.87	12541.8947	313.54
	Optimized-Hybrid	35.27	10644.13 (-17.8%)	301.79
8x8	Hybrid	30.00	9355.17	311.83
	Optimized-Hybrid	29.19 (-2.77%)	8449.62 (-10.7%)	289.47
10x8	Hybrid	25.00	7591.6304	303.66
	Optimized-Hybrid	24.08 (-3.82%)	7120.70 (-6.61%)	295.71 (-2.69%)

DCT is applied during the SP solver kernel reducing the power consumption during this phase. Additional loop optimizations are applied to class C and class D (block size = 4x4). Table 34 and Table 35 present the performance results for the hybrid SP-MZ application for Class C and Class D on the SystemG platform.

Table 35. Performance of Hybrid SP-MZ Application and Optimization (Class D)

#Cores	SP-MZ Type	Runtime(s)	Total Energy (KJ)	Total Power (W)
6x8	Hybrid	862	293403.75	340.453
	Optimized-Hybrid	819 (5.25%)	262235.61 (11.89%)	320.19 (6.32%)
8x8	Hybrid	653	222408.29	340.59
	Optimized-Hybrid	607 (7.57%)	196133.84 (13.39%)	323.12 (5.4%)
16x8	Hybrid	389	132703.46	341.14
	Optimized-Hybrid	344 (13%)	110964.08 (19.59%)	322.57 (5.71%)
32x8	Hybrid	225	76524.75	340.11
	Optimized-Hybrid	205 (-9.76%)	64581.15 (-18.49)	315.03 (-7.96%)
64x8	Hybrid	154	52535.56	341.14
	Optimized-Hybrid	142 (-8.45%)	44995.54 (-16.75%)	316.87 (-7.66%)

Table 36. Performance of MPI SP-MZ Application and Optimization (Class C)

#Cores	SP-MZ Type	Runtime(s)	Total Energy (KJ)	Total Power (W)
1x8	MPI	174.89	59646.23	341.05
	Optimized-MPI	168.29 (-3.9%)	54315.60 (-9.81%)	322.75 (-5.67%)
2x8	MPI	95.83	31354.62	327.19
	Optimized-MPI	91.70 (-4.5%)	29260.55 (-7.15%)	319.09 (-2.53%)
4x8	MPI	47	15531.15	330.45
	Optimized-MPI	43 (-9.30%)	13631 (-13.94%)	317 (-4.2%)
6x8	MPI	38.19	12506.46	327.48
	Optimized-MPI	35.73 (-6.88%)	10940.18 (-14.31%)	306.19 (-6.95%)
8x8	MPI	31.63	10422.48	329.51
	Optimized-MPI	29.84 (-6.00%)	9077.04 (-14.82%)	304.19 (-8.32%)
10x8	MPI	25.89	8496.83	328.19
	Optimized-MPI	24.39 (-6.15%)	7350.90 (-15.50%)	301.39 (-8.89)

Table 36 and Table 37 present the performance results for the MPI SP-MZ application for Class C and Class D. To reduce the power consumption of the application during execution, we apply DVFS to the initialization kernel and first 150 time steps of the application to limit the additional overhead that would be introduced from lowering the frequency throughout different application kernels as the program executes. In

addition, loop optimizations are applied to class C (block size = 4x4) and class D (block size = 8x8) with loop unrolling being applied to the inner loops of the SP solver kernel.

Table 37. Performance of MPI SP-MZ Application and Optimization (Class D)

#Cores	**SP-MZ Type**	Runtime(s)	Total Energy (KJ)	Total Power (W)
6x8	**MPI**	881	298782.34	339.14
	Optimized-MPI	807 (-9.18%)	254221.14 (-17.53%)	315.02 (-6.59%)
8x8	**MPI**	689	233198.94	338.46
	Optimized-MPI	619 (-11.3%)	198742.33 (-17.33%)	321.07 (-5.41%)
16x8	**MPI**	413	139366.85	337.45
	Optimized-MPI	369 (-11.9%)	118984.05 (-17.13%)	322.45 (-4.65%)
32x8	**MPI**	241	81491.74	338.14
	Optimized-MPI	229.53 (-5.0%)	72336.38 (-12.66%)	315.15 (-7.30%)
64x8	**MPI**	173.87	58703.73	337.63
	Optimized-MPI	165.81 (-4.84%)	51780.80 (-13.37%)	312.29 (-8.11%)

5.4.3 LU-MZ

Applying DVFS to the application to reduce power consumption during execution optimizes the NAS LU-MZ application. LU-MZ represents a fairly balanced workload that scales well using OpenMP threads; therefore DCT was not applied, as it would substantially increase execution time. To reduce the frequency of the application

during execution we apply DVFS to the initialization and first 50 time steps of the application during execution to limit the additional overhead that would be introduced from lowering the frequency throughout the entire application. Additional loop optimizations are applied to class C (block size = 4x4). Table 38 and Table 39 present the performance results for the LU-MZ application for Class C for both hybrid and MPI implementations.

Table 38. Performance of Hybrid LU-MZ Application and Optimization (Class C)

#Cores	LU-MZ Type	Runtime(s)	Total Energy (KJ)	Total Power (W)
1x8	Hybrid	199	64653.51	324.89
	Optimized-Hybrid	175	53222.75 (-21.48%)	304.13
2x8	Hybrid	99	32418.21	327.46
	Optimized-Hybrid	95	29460.45 (-10.04%)	310.11
3x8	Hybrid	241	70563.89	292.80
	Optimized-Hybrid	234	67345.20 (-4.77%)	287.80
4x8	Hybrid	50.01	11031.30	216.30
	Optimized-Hybrid	52.91	10681.99 (-3.27%)	201.89
6x8	Hybrid	127	27810.46	218.98
	Optimized-Hybrid	137	27832.92 (0.001%)	203.16

Table 39. Performance of MPI LU-MZ Application and Optimization (Class C)

#Cores	LU-MZ Type	Runtime(s)	Total Energy (KJ)	Total Power (W)
1x8	MPI	574	124156.2000	216.3
	Optimized-MPI	545 (-5.3%)	109534.1 (-13.3%)	200.98
2x8	MPI	292.67	63375.9000	216.30
	Optimized-MPI	281.42	56998.80 (-11.18%)	202.54

5.4.4 GTC

Applying DVFS and DCT to the application to reduce power consumption during execution optimizes the hybrid GTC application. To reduce the frequency of the application during execution, we apply DVFS to the initialization kernel and predict that applying DVFS to the first 25 time steps of the application during execution will provide the optimal execution setting to limit the additional overhead that would be introduced from lowering the frequency throughout the entire application. Additional loop optimizations are applied to 50ppc (block size = 2x2) and 100ppc (block size = 4x4). The inner-most loops of the pushi and chargei subroutines are the most computationally intensive kernels of the application and are unrolled four times. Additionally, Table 40 and Table 41 present the performance results for the hybrid GTC application for 50ppc and 100ppc input sizes.

Table 40. Performance of Hybrid GTC Application and Optimization (50ppc)

#Cores	BT-MZ Type	Runtime(s)	Total Energy (KJ)	Total Power (W)
1x8	Hybrid	427	122753.96	287.48
	Optimized-Hybrid	415 (-2.89%)	114602.25 (-7.11%)	276.15 (-4.0%)
2x8	Hybrid	430	124351.7	289.19
	Optimized-Hybrid	408 (-5.39%)	111783.84 (-11.24%)	273.98 (-5.55%)
4x8	Hybrid	432	125448.48	290.39
	Optimized-Hybrid	410 (-5.36)	112307.2 (-11.7)	273.92 (-6.01%)
6x8	Hybrid	437	128836.34	294.82
	Optimized-Hybrid	419 (-4.29%)	114713.82 (-12.31%)	273.78 (-7.68%)
8x8	Hybrid	444	131130.96	295.34
	Optimized-Hybrid	417 (-6.47%)	114779.25 (-14.25%)	275.25 (-7.30%)
16x8	Hybrid	453	132815.07	293.19
	Optimized-Hybrid	421 (-7.6%)	116343.35 (-14.16%)	276.35 (-6.1%)
32x8	Hybrid	455	134033.9	294.58
	Optimized-Hybrid	424 (-7.31%)	118444.4 (-13.16%)	279.35 (-5.45%)
64x8	Hybrid	436	128528.44	294.79
	Optimized-Hybrid	423 (-3.1%)	114717.6 (-12.03%)	271.12 (-8.73%)

Table 41. Performance of Hybrid GTC Application and Optimization (100ppc)

#Cores	GTC Type	Runtime(s)	Total Energy (KJ)	Total Power (W)
1x8	Hybrid	912	295.58	324.11
	Optimized-Hybrid	885 (-3.1%)	270.02 (-9.45%)	305.11 (-6.22%)
2x8	Hybrid	918	304.70	331.92
	Optimized-Hybrid	883 (-3.96%)	270.9 (-23.55)	306.79 (-6.23)
4x8	Hybrid	922	304.08	329.81
	Optimized-Hybrid	891 (-3.48%)	268.00 (-13.46)	300.79 (-9.64%)
6x8	Hybrid	928	306.74	330.54
	Optimized-Hybrid	904 (-2.65%)	272.51 (-12.29%)	301.45 (-9.65)
8x8	Hybrid	934	311.02	333
	Optimized-Hybrid	902 (-3.55%)	274.21 (-13.42)	297 (-12.12%)
16x8	Hybrid	947	316.30	334
	Optimized-Hybrid	906 (-4.53%)	269.99 (-17.15%)	298 (-12.1%)
32x8	Hybrid	954	313.76	328.89
	Optimized-Hybrid	918 (-3.92%)	272.46 (-15.16%)	296.80 (-10.81%)
64x8	Hybrid	958	314.98	328.79
	Optimized-Hybrid	923 (-3.79%)	271.5 (-16.01%)	294.15 (-11.77%)

Table 42 and Table 43 present the performance results for the MPI implementation of the GTC application for 50ppc and 100ppc input sizes. The application is optimized by applying DVFS to the application to reduce power consumption during execution. To reduce the frequency of the application during execution we apply DVFS to all kernels that are executed during the first 30 time steps of the application to limit the additional overhead that would be introduced from lowering the frequency throughout the entire application. Loop blocking is applied to the MPI implementation with an optimal block size of 4x4 for both input sizes of 50 ppc and 100 ppc. Similar to the hybrid implementation, the inner-most loops of the pushi and chargei subroutines are unrolled four times. The manual loop optimizations are able to achieve strong reductions in execution time for 50 ppc, but smaller optimization benefits are obtained in terms of execution time for 100 ppc input size. It is important to note that the GTC application benefits greatly for the use of OpenMP threads during parallelization. Therefore, the hybrid implementation of the code provides significant savings in power, energy and runtime when compared to the MPI implementation.

Table 42. Performance of MPI GTC Application and Optimization (50ppc)

#Cores	GTC Type	Runtime(s)	Total Energy (KJ)	Total Power (W)
1x8	**MPI**	679.35	213920.52	314.89
	Optimized-MPI	640.42 (-6.10%)	182000.96 (-17.54%)	284.19 (-10.80%)
2x8	**MPI**	682.19	217857.38	319.35
	Optimized-MPI	622.89 (-9.52%)	178775.66 (-21.86%)	287.01 (-11.27%)
4x8	**MPI**	689.73	217761.56	315.72
	Optimized-MPI	618.18 (-11.57%)	177547.48 (-22.65%)	287.21 (-9.93%)
6x8	**MPI**	699.123	222363.06	318.06
	Optimized-MPI	622.92 (-12.23%)	178391.83 (-24.64%)	286.38 (-11.06%)
8x8	**MPI**	709.54	223639.91	315.19
	Optimized-MPI	655.32 (-8.27%)	185757.01 (-20.39%)	283.46 (-11.19%)
16x8	**MPI**	731.95	230359.30	314.72
	Optimized-MPI	673.89 (-8.61%)	192307.99 (-19.79)	285.37 (-10.28%)
32x8	**MPI**	735.72	233164.38	316.92
	Optimized-MPI	679.11 (-8.34%)	192969.11 (-20.83%)	284.15 (-11.53)
64x8	**MPI**	745.14	237021.58	318.09
	Optimized-MPI	684.29 (-8.89%)	194050.96 (-22.15)	283.58 (-12.17)

Table 43. Performance of MPI GTC Application and Optimization (100ppc)

#Cores	GTC Type	Runtime(s)	Total Energy (KJ)	Total Power (W)
1x8	**MPI**	1387.71	466.14	335.91
	Optimized-MPI	1375.19 (-1.0%)	420.66 (-10.81)	305.89 (-9.81%)
2x8	**MPI**	1390.54	468.32	336.79
	Optimized-MPI	1376.91 (-1.0%)	419.86 (-11.5%)	304.93 (-10.44%)
4x8	**MPI**	1397.93	472.51	338.01
	Optimized-MPI	1382.55 (-1.11%)	422.24 (-11.91%)	305.41 (-10.67%)
6x8	**MPI**	1413.19	477.93	338.19
	Optimized-MPI	1389.38 (-1.71%)	419.76 (-13.86%)	302.12 (-11.94%)
8x8	**MPI**	1440.02	488.63	339.32
	Optimized-MPI	1401.9 (-2.71%)	429.78 (-13.69%)	306.57 (-10.68%)
16x8	**MPI**	1456	494.24	339.45
	Optimized-MPI	1413.34 (-3.02%)	432.75 (-14.21%)	306.19 (-10.86%)
32x8	**MPI**	1483.13	502.96	339.12
	Optimized-MPI	1451.39 (-2.19%)	441.50 (13.92%)	304.19 (-11.48%)
64x8	**MPI**	1513.39	513.14	339.05
	Optimized-MPI	1459.10 (-3.72%)	439.74 (-16.69%)	301.38 (-12.50%)

5.4.5 PMLB

The hybrid PMLB application is optimized by applying DVFS and DCT to reduce power consumption during execution. We apply DVFS to the initialization and final kernels of the applications. Additional loop optimizations are applied to execute the application using a block size of 4x4 and nested loops with in the application are unrolled four times. The inner-most loops of the pushi and chargei subroutines are the most computationally intensive kernels of the application and are unrolled four times. Table 44 and Table 45 present the performance results for the hybrid PMLB application for 128 and 256 input sizes.

Table 44. Performance of Hybrid PMLB Application and Optimization (128)

#Cores	PMLB Type	Runtime(s)	Avg Total Energy (KJ)	Avg Total Power (W)
1x8	Hybrid	128.79	37.24	289.19
	Optimized-Hybrid	123.49 (-4.29%)	34.12 (-9.1%)	276.29 (-4.71%)
2x8	Hybrid	91.16	26.66	292.45
	Optimized-Hybrid	85.87 (-6.16%)	23.65 (-12.73%)	275.38 (-6.19%)
4x8	Hybrid	52.32	15.36	293.58
	Optimized-Hybrid	46.36 (-12.83%)	12.70 (-20.94%)	273.91 (-7.11%)
8x8	Hybrid	35.19	10.37	294.57
	Optimized-Hybrid	31.27 (-12.53%)	8.64 (-20.02%)	276.19 (-6.65%)

Table 45. Performance of Hybrid PMLB Application and Optimization (256)

#Cores	PMLB Type	Runtime(s)	Avg Total Energy (KJ)	Avg Total Power (W)
1x8	Hybrid	1878.15	528.12	281.19
	Optimized-Hybrid	1761.03 (-6.65%)	476.34 (-10.87%)	270.49 (-3.96%)
2x8	Hybrid	935.22	261.35	279.45
	Optimized-Hybrid	901.71 (-3.72%)	241.83 (-8.07%)	268.19 (-4.2%)
4x8	Hybrid	416.83	116.87	280.37
	Optimized-Hybrid	398.17 (-4.69%)	103.74 (-12.65%)	260.53 (-7.61%)
8x8	Hybrid	195.31	55.01	281.67
	Optimized-Hybrid	184.39 (-5.92%)	47.05 (-16.9%)	255.19 (-10.37%)
16x8	Hybrid	104.18	29.23	280.53
	Optimized-Hybrid	97.13 (-7.26%)	25.75 (-13.51%)	265.14 (-5.80%)
32x8	Hybrid	57.72	15.97	276.71
	Optimized-Hybrid	56.81 (-1.6%)	15.34 (-4.1%)	270.19 (-2.41%)

The MPI PMLB application is optimized by applying DVFS to reduce power consumption during application execution. To reduce the frequency of the application during execution we apply DVFS to the initialization, communication, and final kernels of the applications. Additional loop optimizations are applied to execute the application using a block size of 4x4 and nested loops with in the application are unrolled four times. Table 46 and Table 47 present the performance results for the hybrid PMLB application for 128 and 256 input sizes.

Table 46. Performance of MPI PMLB Application and Optimization (128)

#Cores	PMLB Type	Runtime(s)	Avg Total Energy (KJ)	Avg Total Power (W)
1x8	MPI	105	31.31	298.19
	Optimized-MPI	98.76 (-6.31%)	27.77 (-12.48%)	281.23 (-6.03%)
2x8	MPI	58.79	17.54	298.34
	Optimized-MPI	54.39 (-8.08%)	15.19 (-15.47%)	279.19 (-6.84%)
4x8	MPI	41.13	12.33	299.85
	Optimized-MPI	37.89 (-8.55%)	10.65 (-15.77%)	281.12 (-6.66%)
8x8	MPI	25.79	7.72	299.17
	Optimized-MPI	23.44 (-10.03%)	6.57 (-17.5%)	280.54 (-6.64%)

Table 47. Performance of MPI PMLB Application and Optimization (256)

#Cores	PMLB Type	Runtime(s)	Avg Total Energy (KJ)	Avg Total Power (W)
1x8	MPI	1259.87	381.44	302.76
	Optimized-MPI	1247.13 (-1.02%)	355.27 (-7.37%)	284.90 (-6.27%)
2x8	MPI	689.31	208.85	302.98
	Optimized-MPI	664.19 (-3.78%)	187.39 (-11.45)	282.14 (-7.39%)
4x8	MPI	379.12	114.18	301.18
	Optimized-MPI	362.29 (-4.65%)	102.29 (-11.62%)	282.33 (-6.68%)

Table 47: Continued

#Cores	PMLB Type	Runtime(s)	Avg Total Energy (KJ)	Avg Total Power (W)
8x8	MPI	185.35	55.75	300.79
	Optimized-MPI	180.13 (-2.90%)	50.64 (-10.1%)	281.13 (-6.99%)
16x8	MPI	88.93	26.75	300.84
	Optimized-MPI	89.46 (0.59%)	25.51 (-4.86%)	285.14 (-5.51%)
32x8	MPI	43.12	12.99	301.29
	Optimized-MPI	46.79 (7.84%)	13.42 (3.2%)	286.91 (-5.01)

5.4.6 Parallel EqDyna

We reduce the execution time and lower power consumption of the hybrid and MPI EqDyna applications by applying DVFS (hybrid and MPI) and DCT (hybrid only). To reduce the power consumption of the application during execution we apply DVFS to the initialization, hourglass, and final kernels of the applications. Additional loop optimizations are applied to execute the application using a block size of 8x8 and nested loops within the application are unrolled four times. For the hybrid application, DCT is applied to the hourglass and qdct3 kernels so that they are executed using 2 threads per node to reduce power consumption during execution. Table 48 and Table 49 present the performance results for the EqDyna application.

Table 48. Performance of Hybrid EqDyna Application and Optimization (e200m)

#Cores	EqDyna Type	Runtime(s)	Avg Total Energy (KJ)	Avg Total Power (W)
2x8	**Hybrid**	3156	880.36	278.95
	Optimized-Hybrid	2980 (-5.9%)	784.31 (-12.25%)	263.19 (-5.98%)
3x8	**Hybrid**	2166	602.49	278.16
	Optimized-Hybrid	2031 (-6.65%)	527.84 (-14.14%)	259.89 (-7.03%)
4x8	**Hybrid**	1681	473.91	281.92
	Optimized-Hybrid	1559 (-7.83%)	409.85 (-15.63%)	262.89 (-7.24%)
8x8	**Hybrid**	839	237.54	283.12
	Optimized-Hybrid	783 (-7.15%)	207.00 (-14.75%)	264.37 (-7.09%)
16x8	**Hybrid**	458	132.36	289.03
	Optimized-Hybrid	422 (-8.5%)	111.83 (-18.35%)	265 (-9.1%)
32x8	**Hybrid**	261	75.37	288.79
	Optimized-Hybrid	246 (-6.1%)	64.23 (-17.34%)	261.11 (-10.6%)
64x8	**Hybrid**	151	42.08	278.67
	Optimized-Hybrid	145 (-4.14%)	36.23 (-16.15%)	249.89 (-11.52%)

Table 49. Performance of MPI Parallel EqDyna Application and Optimization (e200m)

#Cores	EqDyna Type	Runtime(s)	Avg Total Energy (KJ)	Avg Total Power (W)
2x8	**MPI**	3067	966.66	315.18
	Optimized-MPI	2889 (-6.16%)	870.80 (-11.0%)	301.42 (-4.57%)
3x8	**MPI**	2013	637.90	316.89
	Optimized-MPI	1963 (-2.54%)	589.47 (-8.22%)	300.29 (-5.52%)
4x8	**MPI**	1591	510.28	320.73
	Optimized-MPI	1475 (-7.86%)	450.16 (-13.35)	305.19 (-5.09%)
8x8	**MPI**	788	250.17	317.47
	Optimized-MPI	734 (-7.36%)	220.30 (-13.56%)	300.13 (-5.77%)
16x8	**MPI**	427	134.86	315.83
	Optimized-MPI	404 (-5.69%)	123.27 (-9.4%)	305.12 (-3.51%)
32x8	**MPI**	271	85.32	314.83
	Optimized-MPI	249 (-8.83%)	75.45 (-13.1%)	303 (-3.9%)
64x8	**MPI**	167	52.70	315.56
	Optimized-MPI	154 (-8.44%)	47.26 (-11.51%)	306.89 (-2.83%)

5.5 Summary

We presented a methodology to optimize performance of HPC applications on multicore systems by reducing runtime and power consumption. There are two software-based approaches for reducing power consumption in HPC application, DVFS and DCT. Our performance framework is utilized to determine execution configurations of HPC application kernels with regards to the number of OpenMP threads to utilize to execute each application kernel in the hybrid applications.

Further, the kernels of hybrid and MPI applications can be adjusted to execute at lower CPU frequencies to reduce power consumption. Our framework defines a configuration for a given application run to include the CPU frequency setting and concurrency configuration of the application kernel to determine appropriate power and energy savings. If it is possible to obtain power reductions without increasing the execution time of more than 4% but saving 10% in power consumption then the frequency can be lowered.

Experimental results obtained for six hybrid and MPI HPC applications show reductions in execution time and savings in energy consumption based on our modeling framework. Our work illustrates that runtime and power savings can be obtained which reduce application energy consumption. Additionally, we are able to identify the trends exhibited by each application's implementations to determine which will provide for the best energy consumption.

6. SUMMARY AND FUTURE WORK

6.1 Summary

Using the E-AMOM framework in Figure 7, it is our goal to provide for an accurate methodology for predicting and improving the performance and power consumption of HPC applications. In section 3 we provided experimental results, which provided the motivation for further analysis and investigating the energy and performance characteristics of different parallel implementations of scientific applications on multicore systems. This initial experimental work used the power profiling tool PowerPack to collect power profiling data for four scientific applications: a hybrid NAS parallel BT benchmark, a hybrid Lattice Boltzmann application PMLB and a hybrid Gyrokinetic Toroidal Code for our comparative analysis of energy and performance on multicore clusters. Our experimental results show that there are various ways to save energy and improve performance of parallel application codes.

In section 4 we present the E-AMOM modeling scheme for developing predictive performance models to analyze the performance characteristics of hybrid and MPI scientific applications in terms of runtime, system power, CPU power, and memory power. Specifically, our predictive models are able to determine the performance characteristics that affect each respective performance component. The utilization of the Multicore Application Modeling Infrastructure, MuMMI, enables detailed modeling of the application's power consumption and performance-power tradeoffs on multicore

systems. Overall, our E-AMOM predictive models are +95% accurate across six hybrid and MPI scientific applications.

The E-AMOM optimization methodology presented in Figure 80 allows for HPC applications on multicore systems to be improved in terms of reducing runtime and power consumption. The E-AMOM optimization methodology includes two software-based approaches for reducing power consumption in HPC applications, DVFS and DCT. Specifically, E-AMOM determines efficient execution configurations of HPC application kernels with regards to the number of OpenMP threads to utilize to execute each application kernel in the hybrid applications.

Experimental results obtained for six hybrid and MPI scientific applications show reductions in execution time and savings in energy consumption based are achievable. Our work illustrates that runtime and power savings can be obtained which reduce application energy consumption. Additionally, we are able to identify the trends exhibited by each application's implementations to determine which will provide for the best energy consumption. Our work is able to obtain up to 18% in energy savings in hybrid and MPI HPC applications.

6.2 Future Work

6.2.1 Power-Aware Optimization on Heterogeneous Systems

As shown in Table 1, the use of heterogeneous computing systems is increasing and currently top systems in the Top 500 contain a combination of multicore chips and

GPGPUs. Importing HPC applications onto the heterogeneous systems has been seen as a problem until the recent incorporation of traditional parallel programming languages such as OpenMP [33-34]. Being able to execute applications on heterogeneous systems will allow for increased performance on these systems but will also open up a new dimension to exploring savings in power and energy consumption. We will focus on extending the E-AMOM framework to apply towards heterogeneous systems in terms of reducing runtime and saving energy.

6.2.2 Power-Aware Energy Reduction Techniques

In this work we focused our attention on modeling the performance of HPC applications with regards to runtime, and power consumption of the System, CPU, and memory. Additional detailed information can be obtained from MuMMI to model power consumption of the hard disk and motherboard. For the applications presented in this work these components were not a large consumer of total power consumption, however, there are HPC applications that are IO intensive and thus would be affected by these hardware components. Our new research will focus on obtaining detailed power and energy profiles to determine the power consumption of the application in terms of utilization of the system, CPU, memory, motherboard, and hard disk. Furthermore, future work will focus on identifying appropriate optimization strategies to handle these alternative classes of applications to include in E-AMOM.

6.2.3 Power-Aware Scheduling Strategies

Appropriate scheduling in parallel computing has a large effect on the performance of the application on the multicore system. Future work will focus on identifying predictive methods that can be used by schedulers to provide appropriate information pertaining to application performance in regards to the execution time and power consumption of the application. Power consumption is an increasing factor affecting scheduling of jobs on multicore systems. Efficient scheduling algorithms will have to take into consideration the expected execution time and power consumption of a target application.

Predictive scheduling methods can be used be used to obtain required performance statistics that are needed to execute an efficient scheduling algorithm for a given system and an application. Previously collected data on available systems can be used to determine the expected behavior of a given application on other systems. Furthermore, our predictive models can be applied to understanding of various applications can be scheduled with regards to performance and power consumption in relation to CPU and memory utilization.

REFERENCES

[1] S. R. Alam, J. S. Vetter, "A framework to develop symbolic performance models of parallel applications," *Proc. 20th IEEE Int'l Parallel & Distributed Processing Symp.,* pp. 368-377, May 2006.

[2] S. R. Alam, J. S. Vetter. "An Analysis of System Balance Requirements for Scientific Applications", *Proc. 35th Int'l Conf. on Parallel Processing,* pp. 229-236, Aug. 2006.

[3] G. S. Almasi, C. Cascaval, J. G. Castanos, M. Denneau, W. Donath, et al., "Demonstrating the Scalability of a Molecular Dynamics Application on a Petaflops Computer," *Proc. 15th Int'l Conf. Supercomputing,* pp. 393-406, June 2001.

[4] K. Asanovic, R. Bodik, J. Demmel, T. Keaveny, K. Keutzer, et al., "The Landscape of Parallel Computing Research: A View from Berkeley," *Comm. ACM,* vol. 9, no. 52, pp. 56-67, Oct. 2009.

[5] T. Austin, E. Larson, D. Ernst, "SimpleScalar: An Infrastructure for Computer System Modeling," *IEEE Computer,* vol. 35, no. 2, pp. 59-67, Feb. 2002.

[6] D. H. Bailey and A. Snavely, "Performance Modeling: Understanding the Past and Predicting the Future," *Proc. 11th Int'l Euro-Par Conf. on Parallel Processing,* pp. 185-195, Aug. 2005.

[7] E. Bair, T. Hastle, D. Paul, and R. Tibshirani, "Prediction by Supervised Principal Components," *J. of the American Statistical Ass.,* vol. 101, no. 473, pp. 119-137, 2006.

[8] M. Banikazemi, D. Poff, and B. Abali, "PAM: a novel performance/power aware meta-scheduler for multi-core systems," *Proc. 2008 ACM/IEEE Conf. Supercomputing (SC '08),* pp. 39-41, Nov. 2008.

[9] F. Bellosa, "The Benefits of Event-Driven Energy Accounting in Power-Sensitive Systems," *Proc. 9th Workshop on ACM SIGOPS European Workshop,* pp. 37-42, Sep. 2000.

[10] W. Lloyd Bircher, Lizy K. John, "Complete System Power Estimation: A Trickle-Down Approach Based on Performance Events, " *Proc. IEEE Int'l Symp. on Performance Analysis of Systems and Software (ISPASS '07),* pp. 158-168, Apr. 2007.

[11] D. Campbell, M. Hall, W. Harrod, J. Hiller, D. Koester, et al., "ExaScale Software Study: Software Challenges in Extreme Scale Systems Exascale Software Study: Software Challenges in Extreme Scale Systems," *Government PROcurement,* vol. 14, pp. 1-159, 2009.

[12] J. Candy and M. Fahey, "GYRO Performance on a Variety of MPP Systems," *Proc. 47th Cray User Group Conf.,* pp. 1-10, May 2005.

[13] L. Chai, A. Hartono, and D. K. Panda, "Designing High Performance and Scalable MPI Intra-node Communication Support for Clusters," *Proc. IEEE Int'l Conf. on Cluster Computing,* pp. 1-10, Sept. 2006.

[14] L. Chai, P. Lai, H. Jin, and D. K. Panda, “Designing an Efficient Kernel-Level and User-Level Hybrid Approach for MPI Intra-Node Communication on Multi-Core Systems”, *Proc. Int’l Conf. on Parallel Processing (ICPP ’08),* pp. 222-229, Sept. 2008.

[15] J. Chen, L. K. John, and D. Kaseridis, “Modeling Program Resource Demand Using Inherent Program Characteristics,” *Proc. Int’l Symp. Measurement and Modeling of Computer Systems (SIGMETRICS ’11),* pp. 1-12, June 2011.

[16] Y. Chen, Z. Shao, Q. Zhuge, C. Xue, B. Xiao, et al., "Minimizing Energy via Loop Scheduling and DVS for Multi-Core Embedded Systems," *Proc. 11th Int’l Conf. Parallel and Distributed Systems (ICPADS '05),* pp. 2-6, July 2005.

[17] B. Cmelik, D. Keppel, “Shade: A Fast Instruction Set Simulator for Execution Profiling,” *Proc. Int’l Symp. Measurement and Modeling of Computer Systems (SIGMETRICS ’04),* pp. 128-137, June 1994.

[18] Cray XT4/XT5 Supercomputer, http://www.cray.com/, 2008.

[19] M. Curtis-Maury, J. Dzierwa, C. Antonopoulos, and D. Nikolopoulos, “Online Power-Performance Adaptation of Multithreaded Programs using Hardware Event-Based Prediction,” *Proc. Int’l Conf. on Supercomputing (ICS ’06),* pp. 157-166, June 2006.

[20] M. Curtis-Maury et al., “Prediction-Based Power-Performance Adaption of Multithreaded Scientific Codes,” *Proc. IEEE Transactions on Parallel and Distributed Systems (TPDS ’08),* vol. 19, no. 10, pp. 1396-1410, Oct. 2008.

[21] M. Curtis-Maury, A. Shah, F. Blagojevic, D. Nikolopoulos, B. R. de Supinski, et al., “Prediction Models for Multi-dimensional Power-Performance Optimization of Many Cores,” *Proc. 17th Int’l Conf. Parallel Architectures and Compilation Techniques (PACT ’08),* pp. 250-259, Oct. 2008.

[22] J. Dongara, P. Beckman, T. Moore, P. Aerts, G. Aloisio, et al., “The International Exascale Software Project Roadmap,” *Int’l J. of High Performance Computing Applications,* vol. 25, no. 1, pp. 3-60, Feb. 2011.

[23] S. Ethier, “First Experience on BlueGene/L,” *BlueGene Applications Workshop,* pp. 1-8, April 2005.

[24] S. Farfeleder, A. Krall, N. Horspool, “Ultra Fast Cycle-Accurate Compiled Emulation of Inorder Pipelined Architectures,” *J. Systems Architecture,* vol. 53, no.8, pp. 501-510, Aug. 2007.

[25] M. R. Fahey and J. Candy, “GYRO: Analyzing New Physics in Record Time on the Cray X1,” *Proc. 46th Cray User Group Conf.,* pp. 1-10, May 2004.

[26] M. R. Fahey and J. Candy, “GYRO: A 5-D Gyrokinetic-Maxwell Solver,” *Proc. 2004 ACM/IEEE Conf. Supercomputing (SC ‘04),* pp. 1-26, Nov. 2004.

[27] W.-C. Feng and K. Cameron, "The Green500 List: Encouraging Sustainable Supercomputing," *IEEE Computer*, vol. 40, no. 12, pp. 50-55, Dec. 2007.

[28] X. Feng, R. Ge, K. W. Cameron, "Power and Energy Profiling of Scientific Applications on Distributed Systems," *Proc. 19th IEEE International Parallel & Distributed Processing Symposium*, pp. 34-42, May 2005.

[29] V. Freeh, D. Lowenthal, F. Pan, N. Kappiah, R. Springer, B. Rountree, and M. Femal. “Analyzing the Energy-Time Trade-Offs in High-Performance Computing Applications,” *Proc. IEEE Transactions on Parallel and Distributed Systems (TPDS ’08),* pp. 835-848, Oct. 2007.

[30] V. Freeh, Feng Pan, D. Lowenthal, and N. Kappiah. “Using Multiple Energy Gears in MPI Programs on a Power-Scalable Cluster,” *Proc. 10th ACM Symp. Principles and Practice of Parallel Programming (PPOPP ‘05),* pp. 164-173, June 2005.

[31] C. Hsu and W. Feng, “A Power-Aware Run-Time System for High-Performance Computing,” *Proc. 2005 ACM/IEEE Conf. Supercomputing (SC ‘05),* pp. 1-8, Nov. 2005.

[32] IBM Blue Gene, http://www-03.ibm.com/systems/deepcomputing/solutions/bluegene/, 2012.

[33] Intel MPI Benchmarks, Users Guide and Methodology Description (Version 3.2.3), http://software.intel.com/en-us/articles/intel-mpi-benchmarks/, 2012.

[34] C. Isci, A. Buyuktosunoglu, C. Cher, P. Bose and M. Martonosi, “An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget,” *Proc. 39th ACM/IEEE Int’l Symp. Microarchitecture (MICRO-39)*, pp. 347-358, Dec. 2006.

[35] H. Jin, R. F. Van der Wijngaart, "Performance Characteristics of the Multi-Zone NAS Parallel Benchmarks," *J. Parallel Distributed Computing*, vol. 66, no. 5, pp. 674-685, May 2004.

[36] N. Kappiah, V. Freeh, and D. Lowenthal. “Just In Time Dynamic Voltage Scaling: Exploiting Inter-Node Slack to Save Energy in MPI Programs,” *Proc. 2005 ACM/IEEE Conf. Supercomputing (SC ‘05),* pp. 33-41, Nov. 2005.

[37] D. J. Kerbysn, A. Hoisie, and H. J. Wasserman, “Modeling the Performance of Large-Scale Systems,” *IEE Proceedings: Software*, vol. 150, no. 4, pp. 214-221, Aug. 2003.

[38] P. M. Kogge (editor), "ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems," *Univ. of Notre Dame, CSE Dept. Tech. Report TR-2008-13,* Sept. 2008, http://users.ece.gatech.edu/mrichard/ExascaleComputingStudyReports/exascale_final_report_100208.pdf.

[39] B. C. Lee, D. Collins, H. Wang, D. Brooks, “CPR: Composable performance regression for scalable multiprocessor models,” *Proc. 41th ACM/IEEE Int’l Symp. Microarchitecture (MICRO-41),* pp. 270-281, Dec. 2008.

[40] S. Lee and R. Eigenmann. “OpenMPC: Extended OpenMP Programming and Tuning for GPUs,” *Proc. 2010 ACM/IEEE Conf. Supercomputing (SC ‘10),* pp. 1-11, Nov. 2010.

[41] S. Lee, S. Min, and R. Eigenmann. “OpenMP to GPGPU: A Compiler Framework for Automatic Translation and Optimization,” *Proc. 14th ACM Symp. Principles and Practice of Parallel Programming (PPOPP ‘09),* pp. 101-110, Feb. 2009.

[42] J. Levesque, J. Larkin, M. Foster, J. Glenski, G. Geissler, et al., “Understanding and Mitigating Multicore Performance Issues on the AMD Opteron Architecture,” *Tech Report LBNL-62500*, March 2007.

[43] D. Li, B. de Supinski, M. Schulz, K. Cameron and D. Nikolopoulos, “Hybrid MPI/OpenMP Power-Aware Computing,” *Proc. 24th IEEE Int’l Parallel & Distributed Processing Symp.,* pp. 1-12, May 2010.

[44] D. Li, D. Nikolopoulos, K. Cameron, B. de Supinski, and M. Schulz, “Power-Aware MPI Task Aggregation Prediction for High-End Computing Systems,” *Proc. 24th IEEE Int’l Parallel & Distributed Processing Symp.,* pp. 1-12, May 2010.

[45] Y. Li, B. Lee, D. Brooks, H. Zhigang, K. Skadron, "CMP Design Space Exploration Subject to Physical Constraints," *Proc. 12th Int’l Symp. High-Performance Computer Architecture (HPCA),* pp. 17-28, Feb 2006.

[46] M. Lim, A. Porterfield, and R. Fowler, “SoftPower: Fine-Grain Power Estimations Using Performance Counters,” *Proc. 19th Int’l Symp. High Performance Distributed Computing (HPDC ‘10),* pp. 308-311, June 2010.

[47] C. Lively, “Performance Analysis and Modeling of GYRO”, Master’s thesis, Texas A&M University, 2006.

[48] C. W. Lively, S. R. Alam, J. S. Vetter, and V.E. Taylor, "A Methodology for Developing High Fidelity Communication Models for Large-Scale Applications Targeted on Multicore Systems," *Proc. 20th Int’l Symp. Computer Architecture and High Performance Computing,* pp.55-62, Oct. 2008.

[49] Message Passing Interface (MPI), http://www-unix.mcs.anl.gov/mpi/, 2012.

[50] A. Miyoshi, C. Lefurgy, E. Hensbergen, R. Rajamony, and R. Rajkumar, “Critical power slope: Understanding the runtime effects of frequency scaling,” *Proc. 2005 ACM/IEEE Conf. Supercomputing (SC ‘05),* pp. 35–44, Nov. 2002.

[51] Multiple Metrics Modeling Infrastructure, http://www.mummi.org, 2012.

[52] NERSC Bassi, http://www.nersc.gov, 2007.

[53] K. Olukotun, L. Hammond, and J. Laudon, *Chip Multiprocessor Architecture: Techniques to Improve Throughput and Latency.* Morgan & Claypool Publishers, 2007.

[54] OpenMP.org. The OpenMP API Specification For Parallel Programming. http://openmp.org/wp/openmp-specifications, 2011.

[55] D. Poeter, “Cray’s Titan Supercomputer for ORNL Could Be World’s Fastest,” http://www.pcmag.com/article2/0,2817,2394515,00.asp, 2011.

[56] Performance Application Programming Interface, papi, http://icl.cs.utk.edu/papi/, 2012.

[57] SciDAC-DOE’s Scientific Discovery through Advanced Computing, http://www.scidac.gov, 2011.

[58] S. Sharma, C.-H. Hsu, and W.-C. Feng, "Making a Case for a Green500 List," *Proc. 20th IEEE Int’l Parallel & Distributed Processing Symp.,* pp. 343-351, May 2006.

[59] K. Singh, M. Bhadhauria, and S. A. McKee, "Real Time Power Estimation and Thread Scheduling via Performance Counters," *ACM SIGARCH Computer Architecture News*, vol. 37, no. 2, pp. 46-55, May 2009.

[60] Six-Core AMD Opteron Processor, http://www.amd.com/us/products/server/processors/six-core-opteron/Pages/six-core-opteron.aspx, 2010.

[61] A. Snavely, L. Carrington, N. Wolter, J. Labarta, R. Badia, et al., "A Framework for Performance Modeling and Prediction," *Proc. 2002 ACM/IEEE Conf. Supercomputing (SC '02)*, pp. 1-17, Nov. 2002.

[62] R. Springer, D. Lowenthal, B. Rountree, and V. Freeh, "Minimizing Execution Time in MPI Programs on an Energy-Constrained, Power-Scalable Cluster," *Proc. 11th ACM Symp. Principles and Practice of Parallel Programming (PPOPP '06),* pp. 230-238, March 2006.

[63] V. Taylor, X. Wu, and R. Stevens, "Prophesy: An Infrastructure for Performance Analysis and Modeling of Parallel and Grid Applications," *ACM SIGMETRICS Performance Evaluation Review,* vol. 30, no. 4, pp. 13-18, Mar. 2003.

[64] TOP 500 Supercomputers Sites. TOP 500. http://www.top500.org, 2010.

[65] B. Tu, M. Zou, J. Zhan, X. Zhao, and J. Fan, "Multi-Core Aware Optimization for MPI Collectives," *Proc. IEEE Int'l Conf. Cluster Computing,* pp. 322-325, Sept. 2008.

[66] M. Velamati, A. Kumar, N. Jayam, G. Senthikumar, P. K. Baruah, et al., "Optimization of Collective Communication in Intra-Cell MPI," *Proc. 14th Int'l Conf. High Performance Computing (HiPC '07),* pp. 488-499, Dec. 2007.

[67] X. Wu, V. Taylor, J. Geisler, and R. Stevens, "Isocoupling: Reusing Coupling Values to Predict Parallel Application Performance," *Proc. 18th IEEE Int'l Parallel & Distributed Processing Symp.,* pp. 1-10, Apr. 2004.

[68] X. Wu, V. Taylor, C. Lively, and S. Sharkawi, "Performance Analysis and Optimization of Parallel Scientific Applications on CMP Clusters," *Scalable Computing: Practice and Experience,* vol. 10, no. 1, pp. 61-74, 2009.

[69] X. Wu, V. Taylor, S. Garrick, D. Yu, and J. Richard, "Performance Analysis, Modeling and Prediction of a Parallel Multiblock Lattice Boltzmann Application Using Prophesy System," *Proc. IEEE Int'l Conf. Cluster Computing*, pp. 1-8, Sept. 2006.

VITA

Name:	Charles Wesley Lively III
Address:	301 Harvey R. Bright Building, College Station, TX 77843-3112
Email Address:	clively@cse.tamu.edu
Education:	B.S.E., Computer Engineering, Mercer University, 2004
	M.S., Computer Engineering, Texas A&M University, 2006
	Ph.D., Computer Engineering, Texas A&M University, 2012