PERFORMANCE PROJECTIONS OF HPC APPLICATIONS ON CHIP MULTIPROCESSOR (CMP) BASED SYSTEMS

A Dissertation

by

SAMEH SH SHAWKY SHARKAWI

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 2011

Major Subject: Computer Science

PERFORMANCE PROJECTIONS OF HPC APPLICATIONS ON CHIP MULTIPROCESSOR (CMP) BASED SYSTEMS

A Dissertation

by

SAMEH SH SHAWKY SHARKAWI

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,	Nancy Amato
	Vivek Sarin
	R. Lee Panetta
Head of Department,	Valerie Elaine Taylor

May 2011

Major Subject: Computer Science

ABSTRACT

Performance Projections of HPC Applications on

Chip Multiprocessor (CMP) Based Systems. (May 2011)

Sameh Sh Shawky Sharkawi, B.S., The American University in Cairo;

M.S., Texas A&M University

Chair of Advisory Committee: Dr. Valerie Elaine Taylor

Performance projections of High Performance Computing (HPC) applications onto various hardware platforms are important for hardware vendors and HPC users. The projections aid hardware vendors in the design of future systems and help HPC users with system procurement and application refinements. In this dissertation, we present an efficient method to project the performance of HPC applications onto Chip Multiprocessor (CMP) based systems using widely available standard benchmark data. The main advantage of this method is the use of published data about the target machine; the target machine need not be available.

With the current trend in HPC platforms shifting towards cluster systems with chip multiprocessors (CMPs), efficient and accurate performance projection becomes a challenging task. Typically, CMP-based systems are configured hierarchically, which significantly impacts the performance of HPC applications. The goal of this research is to develop an efficient method to project the performance of HPC applications onto systems that utilize CMPs. To provide for efficiency, our projection methodology is automated (projections are done using a tool) and fast (with small overhead).

Our method, called the surrogate-based workload application projection method, utilizes surrogate benchmarks to project an HPC application performance on target systems where computation component of an HPC application is projected separately from the communication component. Our methodology was validated on a variety of systems utilizing different processor and interconnect architectures with high accuracy and efficiency. The average projection error on three target systems was 11.22% with standard deviation of 1.18% for twelve HPC workloads.

DEDICATION

To my mother who gives me all the love, care and support

ACKNOWLEDGEMENTS

Throughout my graduate school, there have been many people who helped and influenced me that I would like to thank; however, I have to acknowledge first that it is the grace of my Lord and Savior Jesus Christ that gave me the ability to reach where I am now. My mother, Amal, how do I ever do enough to thank you? Maybe this PhD is a start. If there is one person in this world that deserves the ultimate acknowledgement for this work, it is you. You lovingly sacrificed everything for me and never asked for anything in return except my happiness. I would also like to thank the two Monicas in my life, my sister, Monica and my fiancée, Monica; My sister for always believing in me and my judgment and my fiancée for being my motivator.

There are many people to thank in my academic life, first and foremost is my advisor and mentor, Dr. Valerie Taylor, who taught me what research really means. You always pushed me to reach perfection in all aspects of my studies and your critique and guidance will always be my guidelines in my future research. I would also like to thank Dr. Xingfu Wu and Charles Lively in my research group for all the constructive discussions and feedback. Thanks are also in order for my past research group member Dr. Ayodeji Coker.

I would like to also thank numerous people at IBM. Stephen Stevens whom I consider my Godfather at IBM, THANK YOU. You were always looking for ways to facilitate my research work and genuinely wanting to help me progress in my career. I also want to thank Don DeSota and Raj Panda for spending long hours with me to discuss research problems. The insight I gained from you both is invaluable. I also want to thank

Rajeev Indukuru, Lorena Pesantez, Joseph Robichau, Jason Cantin and Bradley Elkin for all their help and patience with my questions.

TABLE OF CONTENTS

Page

Page

Page

LIST OF FIGURES

Page

Page

Page

LIST OF TABLES

Page

1. INTRODUCTION

Performance projections of High Performance Computing (HPC) applications onto various hardware platforms are important for hardware vendors and HPC users. The projections aid hardware vendors in the design of future systems, allowing them to compare the application performance across different existing and future systems; the projections help HPC users with system procurement and application refinements. In this dissertation, we present an efficient method to project the performance of HPC applications onto Chip Multiprocessor (CMP) based systems using widely available standard benchmark data. The main advantage of this method is the use of published data about the target machine; the target machine need not be available.

Performance projection of HPC applications allows designers of future systems to explore design trade-offs, such as in-order-execution versus out-order-execution, to develop a system that better matches the performance requirements of the HPC clients. Designers need projections because, in most cases, availability of platforms for HPC applications measurements is limited, especially for competitors' future systems. Further, simulations of such complicated applications on future systems are extremely time consuming. For HPC users, knowing in advance how applications will perform on different platforms help with selecting the best platform for procurement and execution. Further, the projections can aid in identifying areas for performance refinements.

The contributions of this dissertation to the current literature of performance projections can be summarized in the following points:

This dissertation follows the style of *IEEE Systems Journal.*

1. The proposed method uses publicly available benchmarks performance data for the target system in projecting the performance of HPC applications with average error rate of 11.22%. The proposed projection method does not require any access to the target system.
2. The projection method requires low overhead resulting from the use of one base system to characterize the properties of an HPC application and the benchmarks in combination with the publicly available benchmark data.
3. In this work, we model the micro-architecture of the base system and the impact of the micro-architecture on the application performance using detailed hardware counters. Modeling an application behavior independent of the micro-architecture specifications of the system it is executing on may yield inaccurate results [1].
4. For the communication projections, the proposed method uses MPI profiles of the HPC application instead of MPI traces resulting in very little storage requirements and eliminating the need for extensive IO during execution.

The publications resulting from this work are the following:

- S. Sharkawi, D. DeSota, R. Panda, S. Stevens, V. Taylor, X. Wu, *Using MPI Benchmarks to Project MPI Communication Performance of HPC Applications*, Submitted to ICPP 2011.
- S. Sharkawi, D. DeSota, R. Panda, R. Indukuru, S. Stevens, V. Taylor, X. Wu, *Performance Projection of HPC Applications Using SPEC CFP2006 Benchmarks*, in Proceedings of IEEE IPDPS, Rome, Italy, 2009.

The remainder of this section identifies the different requirements and challenges with respect to performance projections and provides a discussion about related work. The remainder of this dissertation is organized as follows: The second Section provides an overview of the proposed performance projection scheme and presents some background on the problem. Section three describes the computation component projection method. Section four describes the communication component projection method and Section six provides the method that we use to integrate the projections of both components. The final section summarizes the overall research and presents areas of future work.

1.1 Performance Projection Requirements

In this dissertation, we present an efficient and accurate method for performance projection of HPC applications. An efficient performance projection method has low overhead in data collection and in the projection process [2], [3]. In addition, a performance projection method is considered accurate when projection error rate is below 20% [1].

1.1.1 Performance Data Requirements: HPC Application and Systems

In this dissertation, we efficiently collect performance data for the HPC applications and benchmarks on one base system. We use this performance data to characterize the micro-architectural characteristics of the base system and the impact of

the base system on the HPC application performance. In order to be efficient, the data collection process must satisfy the following requirements:

(a) Low overhead and low cost. Acceptable low overhead is typically within 5% of the total execution time of the application [2], [3]. Also, the total size of the data collected needs to be moderate for current systems' memories and storage.

(b) No manual intervention for recompilation and instrumentation so as to reduce the requirements on the user for the projections [4]. When instrumentation is inserted before compilation, the presence of instrumentation may inhibit optimization, in which case it will not measure the performance of optimized code; alternatively, optimization may move instrumentation, in which case the measured interval may not include all or only the code from the region of interest.

To produce accurate projections, performance data is required to have the following characteristics:

(a) Performance data needs to capture the major factors that impact the HPC application performance on the system used for execution. The measurement of time or one species of system event seldom identifies performance characteristics [1], [4]. Since HPC applications can be typically viewed as a combination of computation and communication [5], [6], the performance data should accurately represent the single core computation performance as well as the communication between the cores. From a computation perspective, each component of the processor and the memory hierarchy will

have an impact on application performance. As for communication, each communication routine performance and characteristics need to be identified.

(b) Raw event counts are seldom the desired metrics. Derived metrics such as cache miss ratios or cycles per floating point operation are far more useful than raw events in performance modeling [7], [8]. Also, for each application different performance metrics have different impact and this impact varies from one system to another.

1.1.2 Benchmarks Requirements

It is important that the benchmarks used for the projections satisfy the following requirements. First, the set of benchmarks must cover most of the performance space of HPC applications and HPC architectural characteristics. In addition, the benchmark set needs to be able to capture the micro-architectural features of the system it is testing. Second, the benchmark set needs to be a standard benchmark that is used by the industry, academia and HPC users. This allows for abundance of public data for the target systems. Even for future systems, vendors typically publicize about their systems performance using standard benchmarks. Finally, the benchmarks must be scalable to thousands of cores to allow for projections onto systems with different numbers of cores.

1.2 Performance Projection Challenges

HPC systems are becoming more complex and hierarchical utilizing nodes of CMPs [9]. These systems can be configured to scale up to peta- or exa-scale systems. In

addition, CMP based systems utilize different processor architectures and interconnect schemes; thus, mapping from one base system to a target system imposes challenges described in the following subsections.

1.2.1 Complex and Hierarchical System Design

The hierarchical system design has a significant impact on the application performance [10]. The hierarchical design entails the sharing of resources and memory at the level of the cores. In addition, the hierarchical design involves the use of different interconnects for the different levels of the hierarchy. Thus, the projection method needs to take into account these features of the hierarchy.

1.2.2 Mapping Architectural Characteristics from Base to Target System

Recall, published benchmark performance data about target system should suffice to understand the architectural characteristics of the target system in relation to the base. Typically, published benchmark data is a one metric value representing the performance of certain benchmark on a specific system [11]. This one performance metric, although represents some aspect of the system, masks many architectural characteristics of the system [1]. In order to map from the base system to the target, one needs to understand the behavior of the benchmark on the base system, the effect of the base system architecture on the benchmark performance, and the differences in the micro-architecture of the base versus that of the target system as depicted by the differences in the performance metrics of the benchmarks on the two systems.

1.3 Related Work

Several techniques for performance projection have been proposed and used. Most common techniques include program modeling, code analysis, or architectural simulation. Code analysis, in addition to requiring significant expertise and abundant time, may not yield accurate performance projections due to the complexity of the hierarchical systems and applications. Performance projection using program modeling, on the other hand, needs to be a function of (at least) algorithm, implementation, compiler, operating system, underlying processor architecture, and interconnect technology to produce accurate projections. This may be impossible to achieve with the current complex systems and architectures. Architectural simulation yields extremely accurate projection results. Simulation provides the capability to observe component and system characteristics (e.g. performance and power) in order to make vital design decisions. However, simulating high-fidelity models can be very time consuming and even prohibitive when evaluating large-scale systems.

1.3.1 Code Analysis

Manual source code analysis of HPC applications is a very complicated task. It is time consuming, requires high level of expertise and may lack accuracy [4]. To produce accurate projections, it is important to understand the performance characteristics of computation, communication and message passing software used for parallelization. Further, it is important to understand the interaction of the underlying architecture and interconnect with the application. All these keys require significant amount of expertise in parallel programming paradigms and techniques, parallel systems' architecture,

interconnect architecture and significant understanding of the source code of the application. Thus, manual source code analysis can be difficult and time consuming, especially with the hierarchical CMP based systems.

K¨uhnemann et al in [12] proposed automatic source code analysis, which entails compile time prediction of execution time. They use *SUIF* (Stanford University Intermediate Format), which automates performance prediction of HPC applications. For each target system, a system profile is created for computation and communication. Their tool then generates a corresponding runtime function modeling the CPU execution time and the message passing overhead for the source code. This static runtime prediction lacks accuracy especially for MPI calls since collective calls can be translated differently at runtime. There are other automation tools for source code analysis [13]-[17]; however, such tools are better suited for compiler optimizations, identifying bottlenecks and code optimizations.

Miller et al in [15], [17] proposed ParaDyn tool, which uses DynInst [18] to automatically instrument the application binary. ParaDyn's goal is to automate the search for performance bottlenecks in parallel applications, not to project their performance. This is accomplished by attaching ParaDyn instrumentation manager to an application process. ParaDyn then scans the application binary image for procedures entry and exit points. Once this is done, performance metrics, such as CPU usage from the operating system and performance counter data using hardware performance counters, are collected for each procedure to identify performance bottlenecks. Further, Reed et al in [14] proposed SvPablo, which uses Pablo [13] in identifying bottlenecks. Pablo's main difference to ParaDyn is that Pablo instruments source code instead of binary.

Mellor-Crummy et al have developed HPCToolKit [16] that identifies bottlenecks without requiring any code instrumentation. HPCToolKit incorporates three components that are used in the performance analysis process. The first component, *hpcrun*, is responsible for collecting the hardware performance counters; *hpcrun* uses statistical sampling of hardware performance counters, and attributes metrics to both, the calling context and program structure. This is done using stack unwinding techniques to relate performance metrics to source code. The second component, *hpcstruct*, is responsible for analyzing the application binary to recover information about files, functions, loops etc. Finally, *hpcprof* correlates dynamic performance metrics to code structure. Again, the focus is on identifying bottlenecks, not performance predictions.

1.3.2 Simulators

Simulation is often used to predict the performance of key applications to be executed on new systems. Simulation provides the capability to observe component and system characteristics (e.g. performance and power) in order to make vital design decisions. Simulating high-fidelity models, however, can be very time consuming and even prohibitive when evaluating large-scale systems. Simulations can require one to two orders of magnitude more time than the actual application. Currently, there are three major types of simulators:

1. Cycle-accurate node-level simulators [19]-[21]. These simulators are extremely accurate, slow and can only simulate the node level performance of a system. Scaling down an HPC application to fit on the simulator is nearly impossible since these cycle accurate simulators are typically designed to simulate only the

processor core performance. Hardware vendors use these simulators to measure their systems node level performance using industry standard benchmarks such as SPEC CPU.

2. Stochastic network models. These simulators are typically accurate in measuring the interconnect capabilities. They can only simulate network calls such as MPI calls; thus, they can't be used to execute a full HPC application. Typically, benchmarks such as IMB are executed on these simulators to measure the interconnect properties. Prakash and Bagrodia in [22] introduced MPI-SIM, which simulates the MPI communication library using a detailed contention model. The computation component of an application is executed on an actual processor. This limits prediction to systems using the same processor as the base machine. Also Wilmarth et al. in [23] introduced POSE, which uses a detailed network contention model to simulate the communication library only, i.e. no computation component projection.
3. Pseudo-accurate simulators. These simulators attempt to simulate the entire system, computation and communication, with less accuracy to reduce the simulation overhead. Zheng et al in [24] introduced BigSim, which is designed to predict performance using parallel simulation for massively large systems. In BigSim each target processor is emulated using a thread. Since each thread has a significant memory requirement to emulate a full processor, BigSim is well suited for systems that have low memory to processor ratio such as BlueGene systems. For other systems, simulations can be extremely slow. BigSim uses an optimistic approach for communication simulation, which assumes determinism in program

execution rather than an accurate contention model. In addition, BigSim uses heuristics to predict computation (not cycle-accurate). Also Susukita et al in [25] proposed macro-level simulation of communication. Mainly communication is macro simulated by replacing each MPI call by predicted runtime (compile time prediction based on target machine interconnect specification). As for computation, it is skeletonized into single processor execution. In [25], interleaved communication and computation, such as non-blocking MPI calls with computation blocks executed before the MPI call completion, cannot be accurately simulated. Other researches have proposed simulating only the critical components of an HPC application [26]. However, these simulators are not cycle-accurate.

1.3.3 Application Modeling

Performance projection of HPC applications using application modeling provides for a faster method than simulations. Performance projection using program modeling must incorporate algorithm, implementation, compiler, operating system, underlying processor architecture, and interconnect technology in the model to produce accurate projections. Furthermore, since the performance enhancing features of novel processing devices may be significantly different from a conventional microprocessor system, current performance modeling schemes have limited applicability on systems such as vector supercomputers and parallel systems with accelerator devices, systems with deeper memory hierarchy and different multi-core configurations. Thus, performance models

need to employ an application modeling paradigm that allows a user to develop not only "architecture aware" but also "application aware" performance models.

One approach is to build an analytical model for the application on the target platform using one of the known modeling techniques such as LogGP [27] or LogP [28]. The LogP models a parallel application based on four parameters: computational bandwidth, communication bandwidth, communication delay and efficiency of coupling of computation and communication. Since LogP was originally designed for short messages, LogGP extends the LogP model by adding the additional parameter G which reflects the bandwidth for long messages. Both the LogP and LogGP models assume a uniprocessor machine architectures where task placement effect on performance is ignored. Task placement on current SMP systems with CMP hierarchical designs has a major effect on performance specially due to the different communication protocols, shared memory or network interconnect, used by the communication library when communicating with inter vs. intra node processors. We can summarize the main advantages of this work over the LogP and LogGP models in the following points. First, LogP and LogGP models ignore the network topology and the routing algorithm. In current systems, network topologies have complicated and hierarchical designs, which have significant effect on communication performance. In our scheme, network topology effects are reflected in IMB values. Second, in our prediction methodology, support for collective communication acceleration in the hardware on the target system is captured by IMB; however, LogP and LogGP models assume that a processor will only do Send/Recv. Finally, we model *WaitTime* defined in section 4.3 which is typically due to

load imbalance between computation and communication among different tasks. The LogP and LogGP don't model such WaitTime.

Blasko in [29] proposed a modeling technique that would significantly reduce the simulation time. Blasko used hierarchical modeling to analyze the program call structure. He utilized a call graph of the parallel program for the hierarchical performance analysis. In his bottom-up model, all procedures that do not contain any procedure calls are evaluated and simulated at stage one. These are the leaves of the graph. In the second stage, which concerns the callers, the callers are simulated and leaves already simulated in stage one are represented by delays. He then simulates parents all the way up to the root of the graph. This model was tightly coupled to VFCS and also doesn't accurately simulate the contention in the interconnect.

Taylor et al in [30] introduced the Prophesy system. In Prophesy, there are three program modeling techniques, curve fitting which is used to explore application scalability on the same system, parameterization which requires manual analysis of source code kernels to predict performance on target systems and kernel coupling which explores the interaction and sharing between different kernels. Clement and Quinn in [31] proposed modeling an application as a function of compiler effects, memory effects, communication overhead and floating point trends. Their work focused on projecting parallel speedup of an application rather than its performance on different systems.

Another application modeling technique achieves cross platform prediction using partial execution. Yang et al. in [2] proposed the use of partial execution of parallel applications on different systems in performance projection. They used partial execution

of application on target system to indicate its relative performance. This technique requires access to the target system.

2. PROPOSED PERFORMANCE PROJECTION SCHEME AND BACKGROUND

2.1 Performance Projection Framework

In this work we propose an efficient and accurate approach to projecting HPC applications' performance onto systems utilizing CMPs. Our method, called the surrogate-based workload application projection method, utilizes surrogate benchmarks (the surrogate is a linear equation of the appropriate benchmarks) to project an HPC application performance on target systems. Figure 1 depicts the high level framework of our projection scheme. As indicated in Figure 1, our projection scheme entails five steps:

1. Find the surrogate for the computation component of an HPC application. The metrics used to identify the surrogate are described in detail in Section 3.
2. Use the performance data of the surrogate on the target system to project the compute performance of the HPC application on the target system.
3. Find set of surrogates for the communication component of HPC application. The metrics used to identify the set of surrogates are described in detail in Section 4.
4. Use the performance data of the surrogate on the target system to project the communication performance of the HPC application on the target system.
5. Combine the communication and the computation performance projections to come up with the entire HPC application performance projection on the target system.

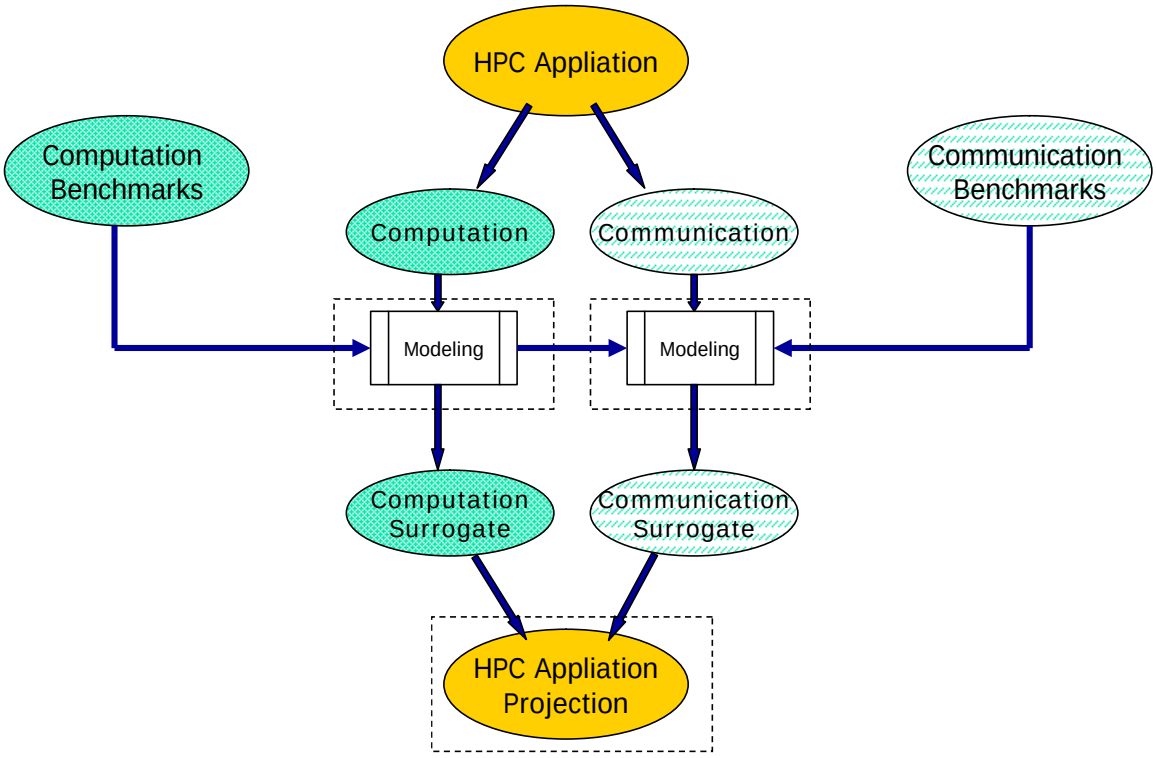

Figure 1: Projection scheme high-level framework

As indicated in Figure 1, this work encompasses modeling the HPC application as two separate components, the computation and the communication. Each component is further modeled as a combination of benchmarks. Compute performance is projected using CPU intensive benchmarks, specifically the SPEC CPU2006 suite [11], and hardware performance counter metrics; communication performance is projected using MPI benchmarks for MPI communications. The components in Figure 1 that are in dashed squares are the main focus of this research. The three main components are given below:

a. Modeling the computation component of an HPC application

b. Modeling the communication component of an HPC application

c. Combining the two models to come up with an HPC application projection

We use the actual executions of surrogates on existing systems to model the HPC application as a linear function of the benchmarks. For the case of non-existing future systems, the only option is to use the simulated runtimes of the benchmarks. In addition, the complex hierarchical design of systems is accounted for in the linear function of the benchmarks. Different configurations for the benchmarks, such as binding one task per core or two tasks per core, exploit different system characteristics.

As for the communication component, we project the communication performance of HPC applications onto different systems using MPI benchmark data on the different systems as well as a base system, and the communication profile of the application on the base system. In particular, we use the Intel MPI Benchmarks (IMB) [32] as we find it the most comprehensive MPI benchmark suite. The arrow from the computation modeling to the communication modeling indicates that communication modeling partially depends on computation modeling.

The use of surrogates for performance projection of HPC applications has been proposed in [33]-[35]. The novelty of our approach in comparison to previous work that used surrogate based projections can be summarized in the following points:

Computation Projection:

1. The use of published benchmarks performance data for the target system suffices to project the performance of HPC application using only one base system.
2. Hardware performance counter metrics are grouped into several groups where each group has certain rank (in comparison to other metric groups) based on

the application properties and the system characteristics. Other research in the literature that used performance counter metrics and benchmarks for HPC projections do not rank metrics. In [33]-[35], all metrics have the same weight and have equal importance to the application.

3. Our approach incorporates available published benchmark data to model the target system in contrast to other work [26], [36], [37], [38] where access to at least one node of target system is required. Further, our approach allows for ranking the metric groups differently for each target system based on the system characteristics. The ranking of metric groups based on system characteristics provides for better projections for different architectures and systems.
4. Our method is independent of any benchmark suite. In this work, we used SPEC CPU2006 suite; however, adding any other benchmark suite(s) is very simple and doesn't require any change to the method.

Communication Projection:

5. The main advantage of our method is the use of MPI profiles of the HPC application in contrast to MPI traces, which require significant storage and are very complicated, hard to understand and parse. For example, our method requires 12KB storage for the communication profile of the NAS BT benchmark [39], in contrast to 2.6 GB storage for an MPI trace of the communication behavior of the same benchmark for 128 tasks. Further, our method does not involve any simulations.

6. Another advantage is that we don't require any simulations of network events as in [5], [6], [40]. Such simulators are typically time consuming. Also creating an accurate contention model for all MPI routines [41]-[43], especially collective calls, can be extremely challenging.

2.2 Proposed Projection Framework Challenges

The separation of the communication and the computation components introduces some challenges and opportunities. MPI calls are reflected in the hardware performance counter metrics. To reduce the noise, we execute the HPC application using four tasks to reduce the communication. It is noted that some applications can't be executed on four tasks due to their large datasets. Also, some applications may have significant communication even with four tasks. A solution to this challenge would be to collect hardware performance counters in between MPI calls. That would require turning off counter collections during the MPI calls. However, there is a huge opportunity in this separation of computation and communication. Such separation allows for more scalability. If computation and communication were coupled, we would have to, one, find surrogates that match the application in communication and computation, two, be limited to parallel benchmarks, and, three, be limited to how scalable the benchmarks are.

2.3 Background

In this section, we discuss how CMP based systems are built and how their design affects the performance of HPC applications. Also, we discuss the base system that is

used throughout this research as well as its architectural characteristics. Finally, we present the benchmark suites of choice for this work that meets the requirements and challenges presented in section 1.2 as well as the HPC applications that are used for validation of our proposed method.

2.3.1 CMPs and Hierarchical System Design

The current trend in high performance computing systems is shifting towards cluster systems with CMPs; further, the CMPs are usually configured hierarchically (e.g., multiple CMPs compose an MCM (multi-chip module), multiple MCMs compose a node, and multiple nodes compose a system). To illustrate, Figures 2 and 3 depict an Intel Nehalem CMP illustrative diagram and a Nehalem CMP chip micro-photograph, respectively. From figure 2, the Nehalem CMP has the following components: four identical compute cores, Cache Interface Unit (CIU) (switch connecting the 4 cores to the 4 L3 cache segments), level-3 (L3) cache controller and data block memory, 1 integrated memory controller (IMC) with 3 DDR3 memory channels and 2 Quick Path Interconnect (QPI) ports [44]. As indicated in the figure, the bandwidth to the main memory, 31.992 GB/s, is quite different from the bandwidth to the interconnect through the QPIs, 25.6 GiB/s ~ 27.5 GB/s. Furthermore, the bandwidth to the interconnect is affected by the geometry of the system and number of hops from source to destination.

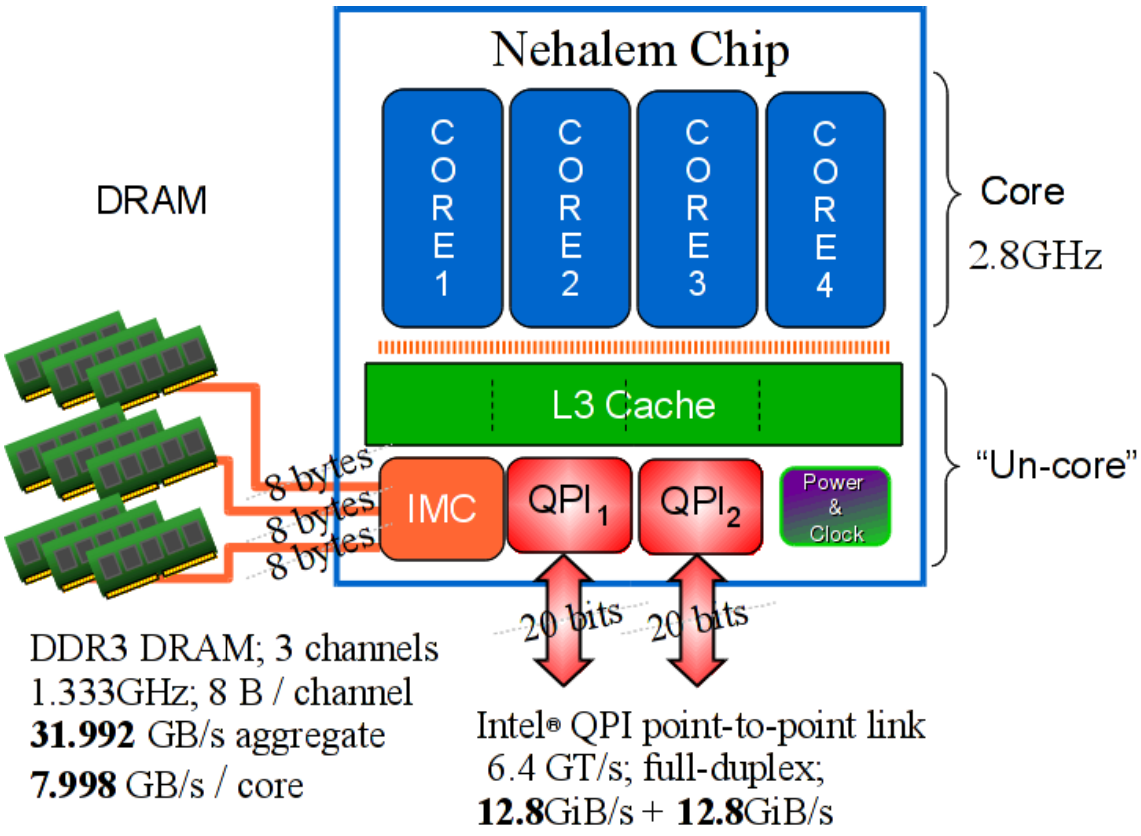

Figure 2: A Nehalem processor and memory module [45], [46]

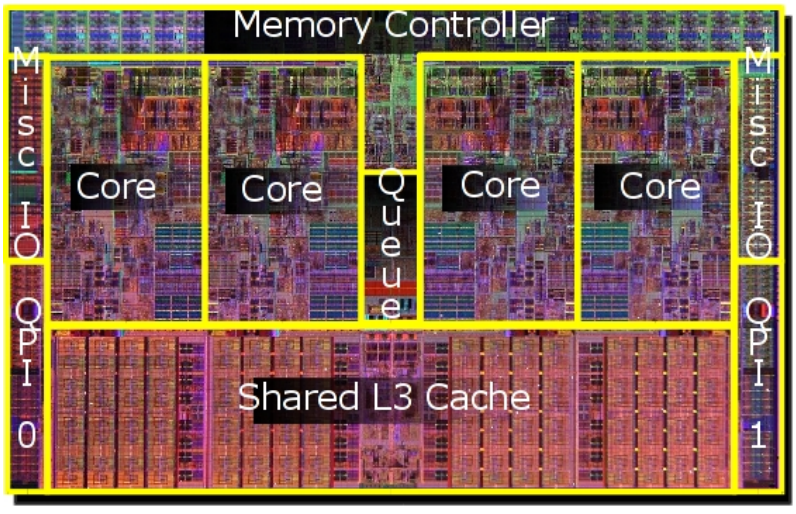

Figure 3: A Nehalem processor chip micro-photograph [45], [46]

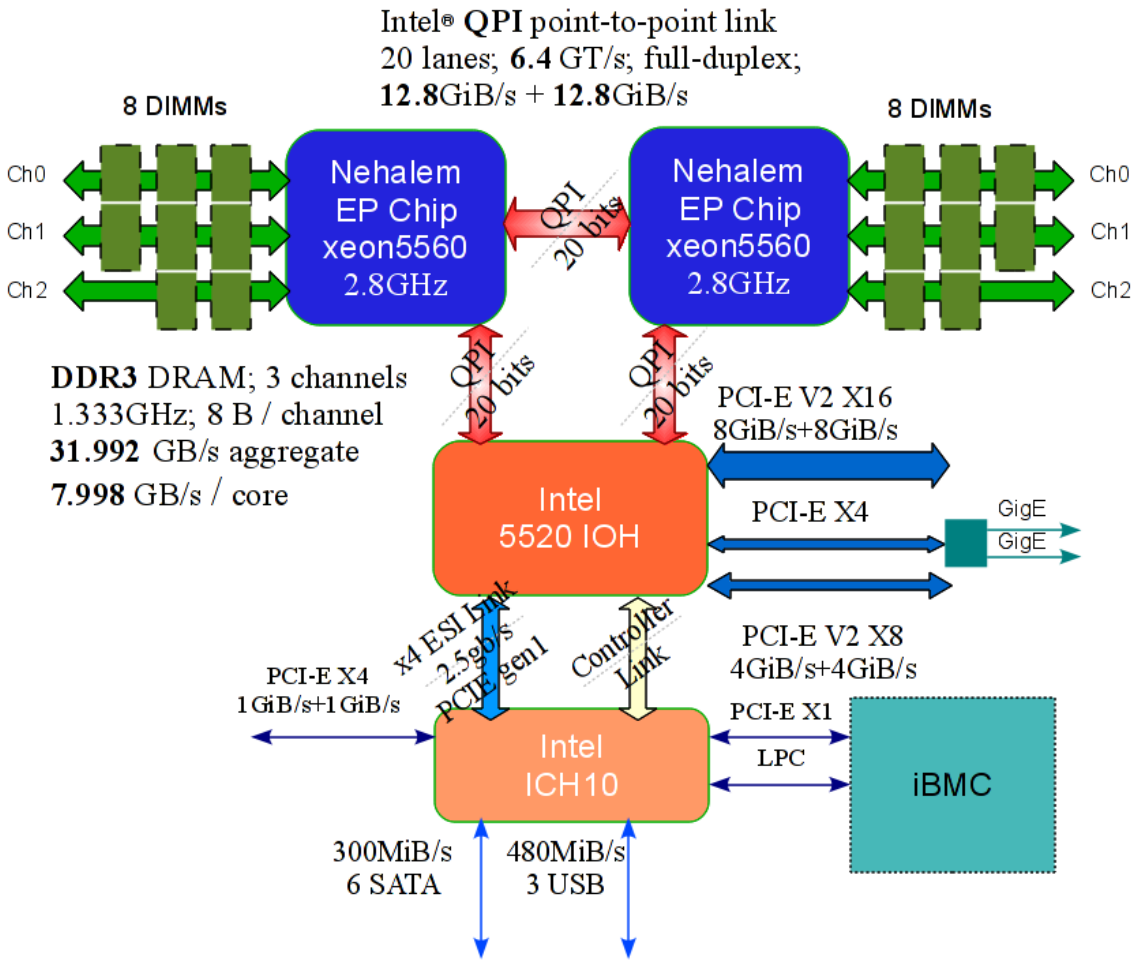

Figure 4: An Intel Nehalem CMP node

As indicated in Figure 4, two CMPs are stacked together to form a CMP node. This CMP node is used a building block to build a larger cluster system connected through an interconnect switch. Typically, an interconnect switch using an InfiniBand technology will use a Fattree topology. This topology has an impact on an HPC application performance since the bandwidth and latency differ from one core to another depending on the distance between cores, number of hops, and layered switch latency.

Hierarchical system design imposes a major challenge for performance projection of HPC applications. The hierarchical design allows for sharing of resources and memory

at the lowest level involving the processing cores, L3 as in Figure 3. In addition, the hierarchical design provides for different interconnect characteristics for different levels in the hierarchy, i.e. cores on the same chip (latency of ~40 cycles), cores on different chips (~32GB/s) or cores on different nodes (~27.5GB/s). These issues must be considered in performance projection schemes of HPC applications. The application interaction with the system needs to be modeled. For example, a memory intensive application performance would be negatively impacted from memory and bandwidth sharing among cores on a CMP; however, the performance of a compute intensive application with minimal memory requirements could be greatly improved. Also, the interconnect characteristics have a major impact on HPC application performance due to the variation in bandwidth and latency among different interconnect levels as explained above. Furthermore, the architecture of interconnect between cores on the same CMP may also be different than the architecture of interconnect between CMPs on an MCM e.g. direct link vs. shared bus based. Therefore, the projection scheme has to be architecture and application aware.

2.3.2 Base System

The base machine used for this work is the p575 POWER5+ cluster system, TAMU Hydra. The POWER5+ chip features single-threaded and multi-threaded execution for higher performance. A single die contains two identical processor cores, each of which uses simultaneous multithreading (SMT) to support two logical threads. The result is a single dual-core POWER5+ chip that appears to be a four-way symmetric multiprocessor to the operating system. Both threads share execution units if both have

work. To the operating system, each thread executes on a logical processor. In our work, the data was collected in SMT mode, with configurations of one thread per core and two threads per core. We refer to the one thread per core configuration as Pseudo Single Thread mode (PST) and two threads per core as Simultaneous Multi Threading (SMT) mode [47].

The motivation for using PST and SMT metrics is to capture the behavioral changes in the application when running under different computing environments or with different set of resources. For example, when running an application in SMT mode, the bandwidth and cache available for each task is different than when running in PST mode. Also when the pipeline resources in the core are shared between threads, there are fewer resources available to each thread; thus, the behavior of the application is likely to change between these modes. The benchmarks that behave similarly to the HPC application under different computing conditions are a better representation for the application on different architectures.

The POWER5+ microprocessor provides Performance Monitor Unit (PMU) counters and a number of Performance Monitor Counters (PMC) to monitor and record several performance events. The POWER5+ has six PMCs per thread. The POWER5+ has 900 total events, 500 unique events, and 230 events per counter [48], [49]. We use the HPMCOUNT [50] tool on IBM systems to collect our hardware counter data. Figures 5 and 6 depict the POWER5+ CMP and the p575 node, respectively.

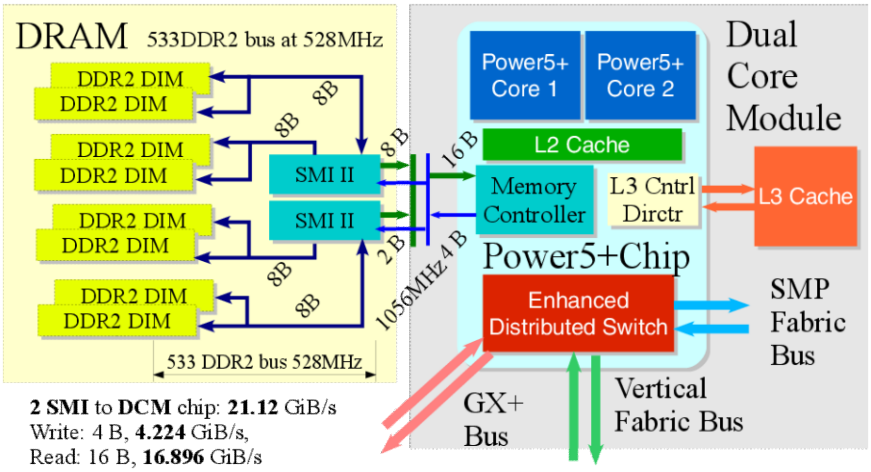

Figure 5: Power5+ CMP and a shared 36MiB L3 cache

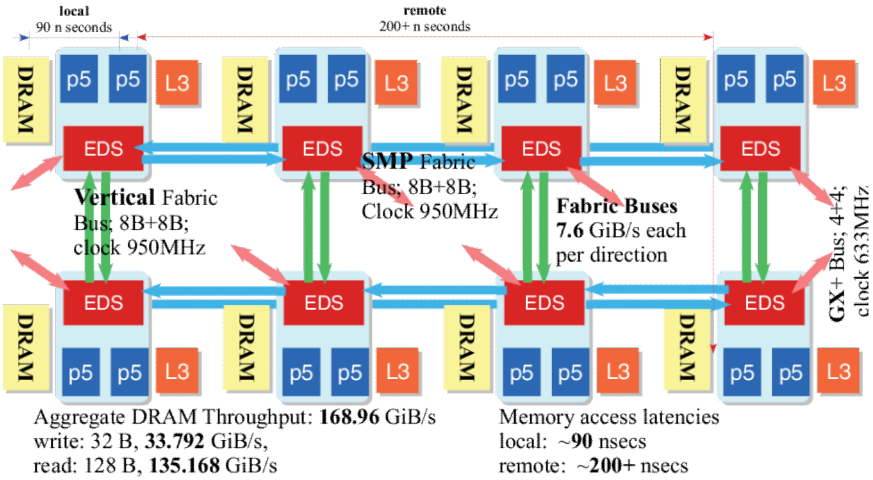

Figure 6: A 16-way p5-575 node (8 CMPs with 2 POWER5+ cores per CMP)

2.3.3 Benchmarks

Recall from section 1.1.2 that the benchmark suites need to be an industry standard, have abundant published data, and span a majority of the performance characteristics space of HPC applications and CMP based systems. Also, to achieve scalability, we presented in section 2.1 our proposed projection method where the computation component is modeled in isolation of the communication component. Thus,

we use a suite of compute intensive benchmarks, specifically SPEC CPU2006, and a suite of communication intensive benchmarks, specifically Intel MPI Benchmarks (IMB).

(a) Compute Intensive Benchmarks: SPEC CPU2006

CPU2006 is SPEC's next-generation, industry-standardized, CPU-intensive benchmark suite, stressing a system's processor, memory subsystem and compiler [11]. SPEC designed CPU2006 to provide a comparative measure of compute-intensive performance across the widest practical range of hardware using workloads developed from real user applications.

SPEC CPU2006 focuses on compute intensive performance, which means these benchmarks emphasize the performance of: the computer processor (CPU), the memory architecture, and the compilers. It is important to remember the contribution of the latter two components. SPEC CPU performance intentionally depends on more than just the processor.

SPEC CPU2006 contains two components that focus on two different types of compute intensive performance: the CINT2006 suite measures compute-intensive integer performance, and the CFP2006 suite measures compute-intensive floating point performance. SPEC CPU2006 is not intended to stress other computer components such as networking, the operating system, graphics, or the I/O system. CINT2006 and CFP2006 are based on compute-intensive applications provided as source code. CINT2006 contains 12 benchmarks: 9 use C, and 3 use C++. The benchmarks are provided in Table 1 below. CFP2006 has 17 benchmarks: 4 use C++, 3 use C, 6 use Fortran, and 4 use a mixture of C and Fortran. The benchmarks are provided in Table 2 below.

Table 1: CINT2006 benchmark suite

Benchmark	Language	Area/Field
400.perlbench	C	PERL Programming Language
401.bzip2	C	Compression
403.gcc	C	C Compiler
429.mcf	C	Combinatorial Optimization
445.gobmk	C	Artificial Intelligence: go
456.hmmer	C	Search Gene Sequence
458.sjeng	C	Artificial Intelligence: chess
462.libquantum	C	Physics: Quantum Computing
464.h264ref	C	Video Compression
471.omnetpp	C++	Discrete Event Simulation
473.astar	C++	Path-finding Algorithms
483.xalancbmk	C++	XML Processing

Table 2: CFP2006 benchmark suite

Benchmark	Language	Area/Field
410.bwaves	Fortran	Fluid Dynamics
416.gamess	Fortran	Quantum Chemistry
433.milc	C	Physics: Quantum Chromodynamics
434.zeusmp	Fortran	Physics/CFD
435.gromacs	C/Fortran	Biochemistry/Molecular Dynamics
436.cactusADM	C/Fortran	Physics/General Relativity
437.leslie3d	Fortran	Fluid Dynamics
444.namd	C++	Biology/Molecular Dynamics
447.dealII	C++	Finite Element Analysis
450.soplex	C++	Linear Programming, Optimization
453.povray	C++	Image Ray-tracing
454.calculix	C/Fortran	Structural Mechanics
459.GemsFDTD	Fortran	Computational Electromagnetics
465.tonto	Fortran	Quantum Chemistry
470.lbm	C	Fluid Dynamics
481.wrf	C/Fortran	Weather Prediction
482.sphinx3	C	Speech recognition

The SPEC CPU2006 is a benchmark suite composed of serial applications. It can be run in throughput mode with multiple instances of a workload to understand multiprocessor behavior. In parallel applications, the execution processes (threads) are distributed across different parallel computing cores. Often the dataset is divided among processors. In contrast, serial applications have one execution process working on the entire dataset. One way to account for this difference is to use throughput data for SPEC. This still leaves the issue that when the number of threads in parallel application changes the dataset per thread changes while with serial applications the working set is a constant size. When running in PST mode, we run four serial tasks of SPEC each bound to a separate core of the POWER5+ chip, thereby using two chips. In this case each task gets a dedicated CPU and L2 resources and the L3 is shared between two tasks. On the other hand, when we run in SMT mode, we run four serial tasks of SPEC each bound on a logical CPU (thread), thus using one chip. In this case two tasks share the CPU and L2 resources and the L3 is shared between four tasks. As for the HPC applications, all runs are configured as four parallel tasks each bound to a separate core on two chips in PST mode or each bound to a separate thread on one chip in SMT mode. Since both runs use four tasks, the dataset size per task remains constant on the parallel application as in the SPEC throughput runs. The effective cache size for each task changes proportionally between PST and SMT modes. Using such configurations we guarantee that the working set size per thread doesn’t change from SMT mode to PST mode.

(b) Communication Intensive Benchmarks: IMB

The idea of Intel MPI Benchmark is to provide a concise set of elementary MPI benchmark kernels. Its objective is to provide a concise set of benchmarks targeted at measuring the most important MPI functions. The IMB-MPI1 contains the benchmarks:

- *PingPong*: is the classical pattern used for measuring startup and through-put of a single message sent between two processes.
- *PingPing*: measures startup and throughput of single messages, with the crucial difference from PingPong that messages are obstructed by oncoming messages.
- *Sendrecv*: Based on MPI_Sendrecv, the processes form a periodic communication chain. Each process sends to the right and receives from the left neighbor in the chain.
- *Exchange*: The group of processes is seen as a periodic chain, and each process exchanges data with both left and right neighbor in the chain.
- *Bcast*: A root process broadcasts X bytes to all.
- *Allgather*: Every process inputs X bytes and receives the gathered X*(#processes) bytes.
- *Allgatherv*: Functionally is the same as Allgather. However, with the MPI_Allgatherv function it shows whether MPI produces overhead due to the more complicated situation as compared to MPI_Allgather.
- *Scatter*: The root process inputs X*(#processes) bytes (X for each process); all processes receive X bytes.

- *Scatterv*: The root process inputs X*(#processes) bytes (X for each process); all processes receive X bytes.
- *Gather*: All processes input X bytes, the root process receives X*(#processes) bytes (X from each process).
- *Gatherv*: All processes input X bytes, the root process receives X*(#processes) bytes (X from each process).
- *Alltoall*: Every process inputs X*(#processes) bytes (X for each process) and receives X*(#processes) bytes (X from each process).
- *Alltoallv*: Every process inputs X*(#processes) bytes (X for each process) and receives X*(#processes) bytes (X from each process).
- *Reduce*: Reduces a vector of length L = X/sizeof(float) float items where X is message size. The MPI data-type is MPI_FLOAT, the MPI operation is MPI_SUM.
- *Reduce_scatter*: Reduces a vector of length L = X/sizeof(float)float items. The MPI data-type is MPI_FLOAT, the MPI operation is MPI_SUM. In the scatter phase, the L items are split as evenly as possible.
- *Allreduce*: Reduces a vector of length L = X/sizeof(float) float items. The MPI data-type is MPI_FLOAT, the MPI operation is MPI_SUM.
- *Barrier*: MPI_Barrier

In addition to the IMB-MPI1 (MPI1 Standard), IMB provides the extended IMB suite which includes a set of benchmarks to measure the MPI2 standard MPI functionalities. In this work, we only focus on the one sided routines of MPI from the

MPI2 standard. The remaining set of routines in the MPI2 standard is MPI-IO related which is not addressed in this dissertation. The benchmarks in the IMB-EXT are classified into three categories [32]:

1. Single transfer: The benchmarks in this class focus on a single data transferred between one source and one target. Single transfer IMB-EXT benchmarks only run with 2 active processes. Single transfer benchmarks, roughly speaking, are local mode. The particular pattern is purely local to the participating processes. There is no concurrency with other activities. Best case results are to be expected.
2. Parallel transfer: These benchmarks focus on global mode, say, patterns. The activity at a certain process is in concurrency with other processes, the benchmark timings are produced under global load. The number of participating processes is arbitrary.
3. Collective: This class contains benchmarks of functions that are collective in the proper MPI sense. Not only is the power of the system relevant here, but also the quality of the implementation for the corresponding higher level functions.

The benchmarks of interest in IMB-EXT that we focus on in this work are:

- *Unidir_Put*: Benchmark for the MPI_Put function.
- *Unidir_Get*: Benchmark for the MPI_Get function.
- *Bidir_Put*: Benchmark for MPI_Put, with bi-directional transfers.
- *Bidir_Get*: Benchmark for MPI_Get, with bi-directional transfers.

- *Accumulate*: Benchmark for the MPI_Accumulate function. Reduces a vector of length L = X/sizeof(float) float items. The MPI data-type is MPI_FLOAT, the MPI operation is MPI_SUM.
- *Window*: Benchmark measuring the overhead of an MPI_Win_create / MPI_Win_fence / MPI_Win_free combination. In order to prevent the implementation from optimizations in case of an unused window, a negligible non trivial action is performed inside the window. The MPI_Win_fence is to properly initialize an access epoch.

2.3.4 HPC Applications

In this work, we used eight large-scale scientific applications and the three NAS Multi-Zone Parallel Benchmarks [39] to validate our projection method. The eight large-scale scientific applications are: AMBER [51], CHARMM [52], FLUENT [53], GAMESS [54], LS-DYNA [55], a seismic application that will be referred to as Seismic, STAR-CD [56] and WRF [57]. Some of these applications have multiple datasets totaling to 16 different workloads each having different computational and communication characteristics. Description of the applications is given in Table 3 below.

Table 3: HPC applications, their HPC areas, datasets and descriptions

Application Name	Category	Datasets	Description
Amber (*Assisted Model Building with Energy Refinement)*	Molecular Dynamics	•GB-COX2: Generalized Born model •Factor IX: Human Factor IX •JAC: Joint Amber and CHARMM	A suite of programs focused on molecular dynamics simulations, particularly on biomolecules.
CHARMM *(Chemistry at HARvard Macromolecular Mechanics)*	Molecular Dynamics	Alanine Dipeptide	A molecular simulation program that focuses on the study of molecules of biological interest, including peptides, proteins, prosthetic groups, small molecule ligands, nucleic acids, lipids, and carbohydrates.

Table 3: continued

Application Name	Category	Datasets	Description
FLUENT	Computational Fluid Dynamics	L1, L2, L3, M1, M2 and M3	A CFD solver for complex flows ranging from incompressible (low subsonic) to mildly compressible (transonic) to highly compressible (supersonic and hypersonic) flows.
GAMESS *(General Atomic and Molecular Electronic Structure System)*	Ab Initio Quantum Chemistry	L-ROTENON and SICCC	An electronic structure code with the primary focus on *ab initio* quantum chemistry calculations.
LS-DYNA	Crash Simulation	3 Car Crash	A general purpose transient dynamic finite element program focused on complex real world problems.
Seismic	Seismic	N/A	A finite-difference algorithm applied in the frequency domain focused on Seismic Migration.

Table 3: continued

Application Name	Category	Datasets	Description
STAR-CD	Computational Fluid Dynamics	Mercedes C-Class	A CFD code that focuses on performing powerful multi-physics (flow, thermal and stress) simulations.
WRF *(Weather Research and Forecasting model)*	Weather Simulation	ConUS	A next-generation mesoscale numerical weather prediction system designed to serve both operational forecasting and atmospheric research needs.

3. COMPUTE COMPONENT PERFORMANCE PROJECTION*

Recall that an HPC application performance is modeled as a function of its computation and communication. The behavior of the compute component of an HPC application is represented by a set of hardware performance counter metrics. This allows for modeling the compute component without the need for simulations or the need for code analysis. We use the SPEC CPU2006 benchmark suite, described in section 2.3.3, for our possible surrogates and the IBM p575 machine, described in section 2.3.2, as the base machine.

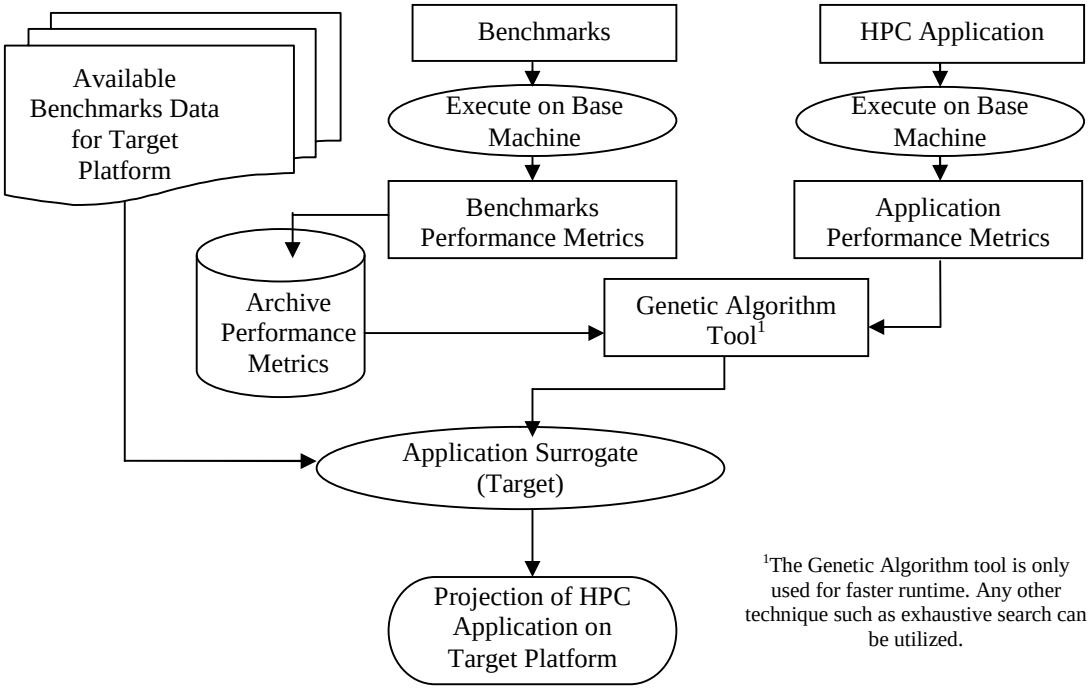

Figure 7: Compute performance projection framework

*Part of this section is reprinted with permission from "Performance Projection of HPC Applications Using SPEC CFP2006 Benchmarks", by Sameh Sharkawi, Don DeSota, Raj Panda, Rajeev Indukuru, Stephen Stevens, Valerie Taylor, and Xingfu Wu, in *Proc. IEEE IPDPS*, Rome, Italy, May 2009, Copyright 2009 by IEEE.

Figure 7 depicts the high level framework of our compute projection scheme. The SPEC CPU2006 benchmarks are executed once on the base machine in Simultaneous Multi Threading (SMT) mode, i.e. two threads per core, and in Single Threaded (ST) mode, i.e. one thread per core. The resultant hardware performance counter data is archived for use as needed. In addition, the HPC applications are executed once on the base machine in SMT and PST modes and the resultant hardware counter data is archived. A tool based on a Genetic Algorithm (GA) is then used to identify the "best" group of benchmarks that have similar behavior as the HPC application; this is done for each HPC application. The output of the GA is a linear equation of the group of benchmarks that best match the HPC application; this equation is given the term "surrogate". Performance data of the benchmarks that comprise the surrogate is then used to project the performance of the application onto a target machine. The target machine performance data for the surrogate is obtained from published data from actual execution or simulations (for future machines).

3.1 Hardware Performance Counter Metrics

We define the applications and benchmarks' behavior as a function of six groups of metrics. These six groups are:

- G_1 – Cycles Per Instruction (CPI) Completion Cycles [49],
- G_2 -- CPI Stall Cycles [49],
- G_3 -- Floating Point Instructions [48],

- G_4 – Data Effective to Real Address Translation (DERAT), Data Segment Lookahead Buffer (DSLB) and Data Table Lookahead Buffer (DTLB) caches miss rates [48],
- G_5 – Data Cache Reloads and [48]
- G_6 – Memory Bandwidth [48].

The choice of these groups of metrics is intended to characterize the application's behavior from a micro-architecture perspective. The first three groups focus on the computation behavior and the last three groups focus on the memory behavior. Group 1 (G_1) shows the portion of the total CPI that was spent by the processor to complete architected instructions, while group 2 (G_2) shows the portion of the total CPI that was spent in various stalling conditions. Group 3 (G_3) shows the distribution of the different types of floating point instruction. Group 4 (G_4) shows the miss rates for DERAT, DTLB and DSLB caches. Group 5 (G_5) shows the application's memory behavior due to cache hits/misses, cache configuration, memory access patterns and memory latency while group 6 (G_6) shows the application's memory bandwidth behavior. Table 4 has the detailed list of the metrics used in this work.

Table 4: Metrics used to capture application behavior

		Metric Name	**Metric Description**
G_1	$m_{1,1}$	CPI_CMPL_CYC	Completion cycles
G_2	$m_{2,1}$	CPI_GCT_EMPTY_IC _MISS	Pipeline Empty due to Instruction-Cache Miss

Table 4: continued

		Metric Name	Metric Description
	$m_{2,2}$	CPI_GCT_EMPTY_BR _MPRED	Pipeline Empty due to Branch MisPrediction
	$m_{2,3}$	CPI_GCT_EMPTY_OTHER	Pipeline Empty (Other)
	$m_{2,4}$	CPI_STALL_LSU_ERAT _MISS	Load,Store Translation Stalls
	$m_{2,5}$	CPI_STALL_LSU_REJECT _OTHERS	Load Store (Other Reject) Stalls
	$m_{2,6}$	CPI_STALL_LSU _DCACHE_MISS	Data Cache Miss Stalls
	$m_{2,7}$	CPI_STALL_LSU _OTHERS	Load/Store flush penalty and latency
	$m_{2,8}$	CPI_STALL_FXU_DIV	Stall by any form of DIV/MTSPR/MFSPR instruction
	$m_{2,9}$	CPI_STALL_FXU _OTHERS	Stall by FXU basic latency
	$m_{2,10}$	CPI_STALL_FPU_DIV	Stall by any form of FDIV/FSQRT instruction
	$m_{2,11}$	CPI_STALL_FPU _OTHERS	Stall by FPU basic latency

Table 4: continued

		Metric Name	Metric Description
	$m_{2,12}$	CPI_STALL_OTHERS	Stall by others (Completion Stall cycles - Stall by LSU Instruction - Stall by FXU Instruction - Stall by FPU Instruction)
G_3	$m_{3,1}$	FPU_FMA_PI	Floating Point multiply and add Per Instruction
	$m_{3,2}$	FPU_OTHER_PI	Floating Point other (div,sqrt,etc.) Per Instruction
	$m_{3,3}$	FPU_STF_PI	Floating Point stores Per Instruction
G_4	$m_{4,1}$	DERAT_MISS_RATE	Data Effective to Real Address Translation Cache miss rate
	$m_{4,2}$	DSLB_MISS_RATE	Data Segment Look-ahead Buffer Cache miss rate
	$m_{4,3}$	DTLB_MISS_RATE	Data Table Look-ahead Buffer Cache miss rate
G_5	$m_{5,1}$	DATA_FROM_L2_PI	Demand d-L1 Reloads from L2 per Instruction
	$m_{5,2}$	DATA_FROM_L3_PI	Demand d-L1 Reloads from L3 per Instruction

Table 4: continued

		Metric Name	Metric Description
	$m_{5,3}$	DATA_FROM_LMEM_PI	Demand d-L1 Reloads from Local Memory per Instruction
	$m_{5,4}$	DATA_FROM_RMEM_PI	Demand d-L1 Reloads from Remote Memory per Instruction
G_6	$m_{6,1}$	MEM_RD_BAND_PI	Memory Read Bandwidth (Bytes/Inst)
	$m_{6,1}$	MEM_WR_BAND_PI	Memory Write Bandwidth (Bytes/Inst)

3.2 Relating Metrics and Metrics' Groups to Runtime

Relating each metric to the runtime *R* of the application allows for understanding the contribution of these metrics in the overall behavior of the application. This relationship is dependent on the architecture of the base machine. The process of relating metrics to runtime *R* is accomplished in two steps: (1) local to each metric group and (2) across all metric groups. The first step entails finding the contribution of each metric to the overall group for that metric. The second step entails finding the contribution of a given group to the overall runtime. Details about each step are given below.

For the local step, we use the typical number of cycles that each metric uses to calculate the contribution of each metric in its respective group. For example, each metric in G_3 represents a different type of FPU instruction. Understanding the base machine architecture, we know how many cycles each of these different types of FPU instructions typically require. This can be represented mathematically by defining a function F_i for

each metric group G_i where F_i is directly proportional to runtime R. We define this function F_i as given below:

$$F_i = \sum_{j=1}^{M_i} c_{i,j} \times m_{i,j} \quad (1)$$

where M_i is the number of metrics in G_i and $c_{i,j}$ is a coefficient representing the contribution of metric $m_{i,j}$ to runtime R relative to other metrics within G_i. Each coefficient $c_{i,j}$ is determined based on the cycles associated with metric $m_{i,j}$; the values for $c_{i,j}$ are obtained from the specification of the base micro-architecture. To illustrate, using the same example of G_3, $c_{3,1}$ (FPU_FMA_PI), $c_{3,2}$ (FPU_OTHER_PI) and $c_{3,3}$ (FPU_STF_PI) have the values of X, Y, Z, respectively, corresponding to $X \approx 6$ cycles required for the floating-point multiply-add operation, $Y \approx 35$ cycles required for other floating-point operations, and $Z \approx 20$ cycles required for floating-point store operations.

The second step in the process of relating metrics to overall runtime is to find the contribution of each group of metrics to the overall application runtime. In other words, we need to find the function H that relates each group G_i to R using coefficients a_i such that $H \propto R$. H can be defined as follows:

$$H = a_1G_1 + a_2G_2 + a_3G_3 + a_4G_4 + a_5G_5 + a_6G_6, H \propto R \quad (2)$$

where a_i represents the contribution of each metric group G_i to the total runtime. The values for a_i are calculated using the cycles associated with each group relative to the runtime. These values are obtained from the micro-architecture specification of the base machine.

3.3 Computation Performance Projection Scheme

The process of performance projection of the compute component of HPC applications entails three steps. The first step involves characterizing/modeling each HPC application by providing ranks, corresponding to a_iG_i as in Equation 2, to the different metric groups on the base machine. The second step is to adjust these ranks for each target machine we want to project the performance on. These adjusted ranks provide for a performance model of the application on the target machine. Once we have the ranks in place for the target machines, we then use a genetic algorithm (GA) tool to identify the benchmarks and their respective coefficients that are similar to the HPC application based upon the performance on the base machine. The three steps allow for the HPC characterization/modeling on the target machine to be used with the similarity analysis to produce better results.

3.3.1 Calculating Ranks for Metrics' Groups on Base Machine

Our goal in this step is to find the rank for each metric group. In other words, we want to arrange the metrics' groups according to their contribution in runtime on the base machine in a descending order. Thus, the rank of each metric group reflects its significance to the application behavior/runtime. To illustrate, HPC applications can be broadly characterized as compute intensive (e.g., requires significant number of compute operations per memory operation) or memory intensive (e.g., requires significant number of memory operations per computation). Consequently, memory intensive applications may have groups G_5 (data cache reloads) ranked higher than G_3 (FPU instructions).

Calculating the ranks of metric groups follows directly from Equation 2. The coefficients a_i in Equation 2 are already calculated based on the architectural characteristics of the base machine as in Section 3.2. The values for G_i are calculated using the function F_i in Equation 1 corresponding to each group G_i. The rank of a group G_i then corresponds to the magnitude of the term a_iG_i for this group, the higher the magnitude of a_iG_i the higher the rank of the group G_i.

3.3.2 Calculating Ranks for Metrics' Groups on Target Machine

The significance of each metric group to the performance of the HPC application is relative to the architecture of the machine the application is running on. Since the rank of each metric group reflects the significance of this metric group to the performance of the HPC application in relation to the architecture of the machine, the rankings calculated on the base machine need to be adjusted for the target machine. The availability of performance counter metrics for the set of benchmarks on the base machine, and the availability of their runtimes on both the base and the target machine allows for mathematically adjusting the ranks of the metric groups from the base to the target.

Since the goal in this step is to adjust the ranks of metrics' groups on the base for the target machine, we need to identify the differences between these two machines. The architectural characteristics of a machine are reflected in the coefficients a_i in Equation 2; thus, we need to calculate coefficients a_i' for each group G_i on the target machine that will reflect the architectural difference of the target machine from the base. To calculate a_i', we define the set B which includes all the benchmarks, SPEC CFP2006 in this case. For each benchmark b_l in the set B, we define H_{bl} using Equation 2 as follows:

$$H_{b_l} = a_1 G_{1_{b_l}} + a_2 G_{2_{b_l}} + a_3 G_{3_{b_l}} + a_4 G_{4_{b_l}} + a_5 G_{5_{b_l}} + a_6 G_{6_{b_l}}, H_{b_l} \propto R_{b_l} \forall l \in B \quad (3)$$

where R_{bl} is runtime of benchmark b_l on base machine. Also we define H_{bl}' using Equation 2 as follows:

$$H_{b_l}' = a_1' G_{1_{b_l}} + a_2' G_{2_{b_l}} + a_3' G_{3_{b_l}} + a_4' G_{4_{b_l}} + a_5' G_{5_{b_l}} + a_6' G_{6_{b_l}}, H_{b_l}' \propto R_{b_l}' \forall l \in B \quad (4)$$

where R_{bl}' is runtime of benchmark b_l on target machine. From Equations (3) and (4), we can get the ratio between the runtimes of benchmark b_l on the base machine and the target and define H_{bl}' as follows:

$$H_{b_l}' = H_{b_l} \times \frac{R_{b_l}'}{R_{b_l}} \quad (5)$$

Since H_{bl} can be calculated for the base machine, and runtimes R_{bl} of the base and R_{bl}' of the target are known, we end up with a set of simultaneous linear equations each for a different benchmark b_l in B. In this set, we are solving for six unknowns, a_1', a_2', a_3', a_4', a_5' and a_6'. After solving the set of linear equations, we identify the values for the coefficients a_i' for each group G_i on the target machine. These coefficients reflect the architectural characteristics of the target machine and how different/similar it is from the base machine. Once the coefficients a_i' are identified, we calculate the ranks for the metrics' groups on the target machine in the same way we calculated the ranks for the base using Equation 2 where the rank of a group G_i corresponds to the magnitude of the term a_i'Gi for this group, the higher the magnitude of a_i'G_i the higher the rank of the group G_i.

3.3.3 Identifying Surrogates

In the last step of our projection methodology we attempt to select some benchmarks and coefficients for those benchmarks that will represent an application whose behavior is the closest to the HPC application at hand. These selected benchmarks that comprise the surrogate for a given application, *app*, form the set S_{app}. S_{app} is a subset of the set *B*, the set of SPEC CPU2006 benchmarks, and $S_{app} \subseteq B$. Each member s_k of S_{app} has a weight w_k. The combination of the surrogate called *comb_surrogates* is defined as follows:

$$comb\_surrogates = \sum_{k=1}^{|S_{app}|} w_k \times s_k \tag{6}$$

The smaller the error between the metrics of the application and the metrics of *comb_surrogates*, the closer is the behavior of *comb_surrogates* to the application. Thus, we identify *comb_surrogates* of an HPC application by attempting to minimize the error between the metrics of the HPC application and that of *comb_surrogates* for the base machine. A genetic algorithm (GA) tool is used to identify the members of S_{app} and their respective weights. We define the error between the metrics of the application and the metrics of *comb_surrogates* as in Equation 7

$$E_i = \sum_{j=1}^{M_i} \left(\left| m_{i,j(app)} - m_{i,j(comb\_surrogates)} \right| \times \frac{m_{i,j(app)}}{\sum_{q=1}^{M_i} m_{i,q(app)}} \right) \tag{7}$$

where E_i is the weighted sum of errors of all metrics in group G_i, M_i is the number of metrics in G_i, $m_{i,j}$ is metric j in group G_i as in Table 4. Also $m_{i,j(comb\_surrogates)}$ is calculated as follows:

$$m_{i,j(comb\_surrogates)} = \sum_{k=1}^{|S_{app}|}(m_{i,j(s_k)} \times w_{s_k} \times \frac{I_{s_k}}{I_{total}}))] \quad (8)$$

where I_{sk} is the total number of run instructions or the path length of surrogate s_k and I_{total} is the sum of all run instructions of all s_k in S_{app}. We multiply the weighted metric of a surrogate s_k by the term I_{sk}/I_{total} to account for the contribution of this surrogate in *comb_surrogates* since by combining surrogates we assume running the surrogates serially. The multiplication of this term accounts for the differences in runtimes of the surrogates.

Since we collect the metrics of the HPC application and the benchmarks in both SMT and ST modes as mentioned earlier above, the GA tool attempts to minimize the error in metrics for each of the six groups for both SMT and ST modes to be below a chosen limit. In this work, we chose the limit to be 10.0%. To achieve this, we defined the fitness function of the GA tool as follows:

$$\begin{aligned} &if\ (\frac{E_1}{\sum_{q=1}^{M_1} m_{1,q(app)}} < 0.1 \,\&\, \frac{E_2}{\sum_{q=1}^{M_2} m_{2,q(app)}} < 0.1 \,\&\, .. \,\&\, \frac{E_6}{\sum_{q=1}^{M_6} m_{6,q(app)}} < 0.1) \\ &\qquad return(0) \\ &else \\ &\qquad return(\frac{E_1}{\sum_{q=1}^{M_1} m_{1,q(app)}} + \frac{E_2}{\sum_{q=1}^{M_2} m_{2,q(app)}} + ... + \frac{E_6}{\sum_{q=1}^{M_6} m_{6,q(app)}}) \end{aligned} \quad (9)$$

The GA tool terminates when the return value of the fitness function is *0*; thus, the term $E_i / \sum_{q=1}^{M_i} m_{i,q(app)}$ for each of the six groups for both SMT and PST modes has to be below 0.1.

The goal of reducing the error in metrics below the 10.0% limit for all metrics' groups is not achievable in many cases. Nevertheless, the ultimate goal is to reduce the projection results error in runtime on the target machine. Consequently, reducing the error in metrics' groups with the highest contribution in runtime will yield better projection results than reducing the error in metrics' groups with the least contribution in runtime. Thus, in the cases where the limit of 10% is not achievable, we adjust the GA tool to reduce the error in metrics with the highest rank on the target machine first. Once the error limit is achieved in the highest ranked metric group, the GA tool attempts to reduce the error on the next highest ranked metric group and so on. These ranks were calculated in the previous step of projection methodology in Section 3.3.2. The HPC application characterization/modeling on the target machine using the metrics' group ranking in combination with the similarity analysis produces better projection results.

In our scheme, we use the SPEC CFP2006 performance throughput data of target machines and calculate the relative performance to our base machine. Once the set S_{app} of the surrogate and the respective weights for the HPC application is found using our scheme, we apply the following steps to get the application runtime on a target machine:

I. Multiply the surrogates with their respective weights to get the application relative performance on the target machine.

$$P_{app} = \sum_{k=1}^{|S_{app}|} (w_k P_k) \tag{10}$$

where P_k is the relative performance of surrogate s_k and w_k is the weight for surrogate s_k.

II. The scaling factor is then multiplied by the application runtime on the base system which gives the projection of the application on the target system.

3.3.4 Genetic Algorithm to Identify Surrogates

In the previous section, we discussed the details of the projection scheme used. To address time requirements, we use a probabilistic method to minimize the difference value discussed in the previous section. In particular, the complexity of the algorithm to find the best combination of surrogates and their respective weights is in the order of $O(wn^m)$ where w is the range of weights to try on each benchmark that comprise the surrogate, n is the number of surrogates and m the combination size. In this work, we used a string based Genetic Algorithm (GA) tool [58]. Parameters for the genetic algorithm are shown in Table 5. We defined the string as a sequence of bits representing the benchmarks and sequences of weights for each of these benchmarks. A “1” bit means choose this benchmark otherwise don’t choose it. The weights for the benchmarks range from 0.0009765625 (1/1024) to 1024. Figure 8 shows the framework for the genetic tool. Once the population is generated, each string is decoded and we get the surrogates and their weights. We use Equation 9 as our fitness as indicated earlier. After calculating fitness, we check for termination fitness (zero for perfect match) or we try other individuals. When the tool is stuck for several generations, cataclysm is performed until reaching max generations.

Table 5: Genetic Algorithm parameters

Parameter	Value
Tournament Size	7
Population Size	10000
Mutation Rate	0.03
Reproduction Rate	0.10- 1/Pop Size
Elite Reproduction	1/Pop Size
Crossover rate	0.77
Max Generations	2000
Max Generations No Progress	15
Termination Fitness	0.0

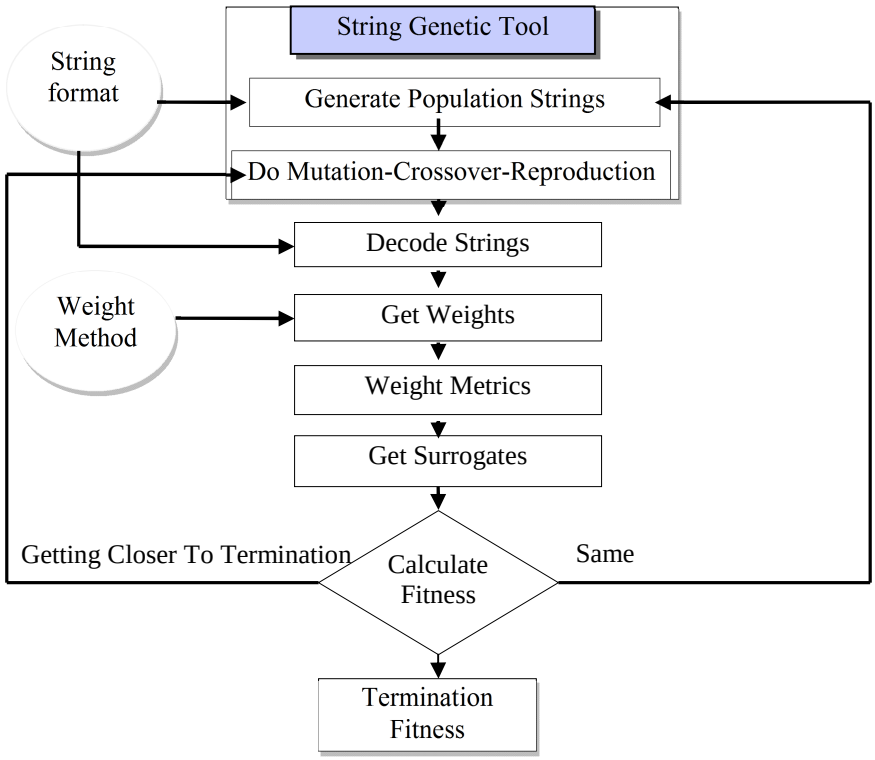

Figure 8: Genetic tool framework

3.4 Experimental Results

We used our method given in Figure 7 to project the performance of the following eight large-scale scientific applications: AMBER [51], CHARMM [52], FLUENT [53], GAMESS [54], LS-DYNA [55], a seismic application that will be referred to as Seismic, STAR-CD [56] and WRF [57] on four different systems.. Table 3 in section 2.3.4 lists the input datasets and the category for the applications we used in our experiments. Table 6 presents the different systems that we projected on and their respective properties. We chose the systems to be quite different from the base as well as from each other. The POWER6 chip utilized in the two JS22 and p570 systems, although having the same ISA as the base machine, has an extremely different micro-architecture than the POWER5+ chip. As indicated in Table 6, POWER5+ chip utilizes two out-of-order execution cores, while POWER6 chip utilizes two in-order execution cores. Also, the two POWER6 systems have quite different cache and memory subsystems. On the other hand, the Intel Woodcrest chip has dual out-of-order execution cores that have different ISA and micro-architecture than the POWER5+. The Clovertown is a multi-chip module (MCM) with dual Woodcrest chips that run at a slower frequency and share the memory bandwidth.

Table 6: Computation base system and target systems used for validation

Machine	Processor	Number of Cores	Processor Frequency	Memory Per Core	L2 Cache/core
IBM p575	Out of Order Execution POWER5+	2	1.9 GHz	4GB	1.9 MB (shared)
IBM JS22	In Order Execution POWER6	2	4.0 GHz	4GB	4 MB
IBM p570	In Order Execution POWER6	2	4.7 GHz	8GB	4 MB
IBM x3550	Out of Order Intel Woodcrest	2	3 GHz	2GB	2 MB
IBM x3650	Out of Order Intel Clovertown	4	2.4 GHz	2GB	2 MB

In the results figures below we show the signed value of the error in runtime; however, in this work, we focused on reducing the magnitude of the runtime error and all averages are based on absolute errors. The number of selected benchmarks that comprise the surrogate for each application was ranging between a minimum of one and a maximum of four as indicated in Table 7. Typically, the benchmark(s) with the highest weights were from the same scientific area of the HPC application.

Table 7 indicates the different surrogates of the HPC applications for the POWER6 IBM JS22 system. In the area column the abbreviations for areas are as follows: QC – Quantum Chemistry, FD – Fluid Dynamics, MD – Molecular Dynamics, WS – Weather Simulation, P – Physics, RT – Ray Tracing, O – Optimization, SM – Structural Mechanics and S – Seismic. For the properties column the abbreviations for properties are as follows: CI – Compute Intensive, MI – Memory Intensive, IB – In Between (it lies in the middle between compute intensive and memory intensive), LB – Low Bandwidth and HB – High Bandwidth. The numbers in the surrogates' column correspond to the SPEC CFP2006 numbers and their weights respectively. As it indicates in Table 7, all surrogates for HPC applications are typically from the same area of the HPC application. From a micro-architectural perspective, the combined benchmarks in the surrogate always have the micro –architectural properties as the HPC application. For example, in the case of AMBER-COX2, the combined surrogates are computing intensive with very low bandwidth requirements as AMBER-COX2; in addition, the bigger component of the combined surrogates (435.gromacs) is Molecular Dynamics benchmark as AMBER-COX2. Another point worth mentioning is that in some cases the individual benchmarks have different micro-architectural properties compared to the HPC application; however, when combined, the resultant combined surrogate has very similar properties to the HPC application. To illustrate, in the case of LS-DYNA, 416.gamess is a compute intensive benchmark with very low bandwidth requirement, while LS-DYNA is a memory intensive application; nevertheless, when 416.gamess is combined with the other surrogates such as 470.lbm, 436.cactusADM, the resultant combined surrogates have very similar properties to LS-DYNA.

Table 7: Surrogates for the HPC applications on POWER6 IBM JS22 system

	Area	Prop.	Surrogates	Surr. Area	Surr. Prop.
AMBER-COX2	MD	CI	*416* x 0.877 + *435* x 1.712	QC – MD	CI
AMBER-FIX	MD	MI - LB	*416* x 0.737 + *436* x 1.479 + *444* x 1.936	QC – P – MD	MI – LB
AMBER-JAC	MD	MI – LB	*410* x 0.092 + *435* x 0.633 + *444* x 2.359	FD – MD	MI – LB
CHARMM	MD	CI	*416* x 0.907 + *444* x 1.487	QC – MD	CI
FLUENT-L1	FD	MI – HB	*416* x 1.035 + *444* x 1.393 + *447* x 1.251 + *450* x 1.356	QC – MD – O	MI – HB
FLUENT-L2	FD	MI – HB	*410* x 1.169 + *416* x 0.742 + *437* x 1.113 + *465* x 1.583	FD – QC	MI - HB
FLUENT-L3	FD	MI – HB	*410* x 1.106 + *416* x 0.573 + *454* x 0.792 + *465* x 2.104	FD – QC – SM	MI – HB
FLUENT-M1	FD	IB	*465* x 1.569 + *481* x 0.138	QC – WS	IB
FLUENT-M2	FD	IB	*416* x 0.229 + *437* x 0.674 + *465* x 2.938	QC – FD	IB
FLUENT-M3	FD	IB	*437* x 0.337 + *465* x 1.853 + *481* x 0.489	FD – QC – WS	IB
GAMESS-LROT	QC	CI	*453* x 0.079 + *465* x 1.094	RT – QC	CI
GAMESS-SICC	QC	CI	*465* x 0.860	QC	CI
LS-DYNA	FD	MI – HB	*416* x 0.793 + *436* x 1.837 + *450* x 0.158 + *470* x 1.414	QC – O – P – FD	MI - HB
Seismic	S	MI – HB	*416* x 1.004 + *470* x 1.518	QC – FD	MI - HB
STAR-CD	FD	MI – HB	*410* x 1.782 + *481* x 0.736	FD – WS	MI – HB
WRF	WS	MI – HB	*410* x 0.971 + *436* x 0.491 + *454* x 1.187 + *481* x 1.153	FD – P – SM – WS	MI – HB

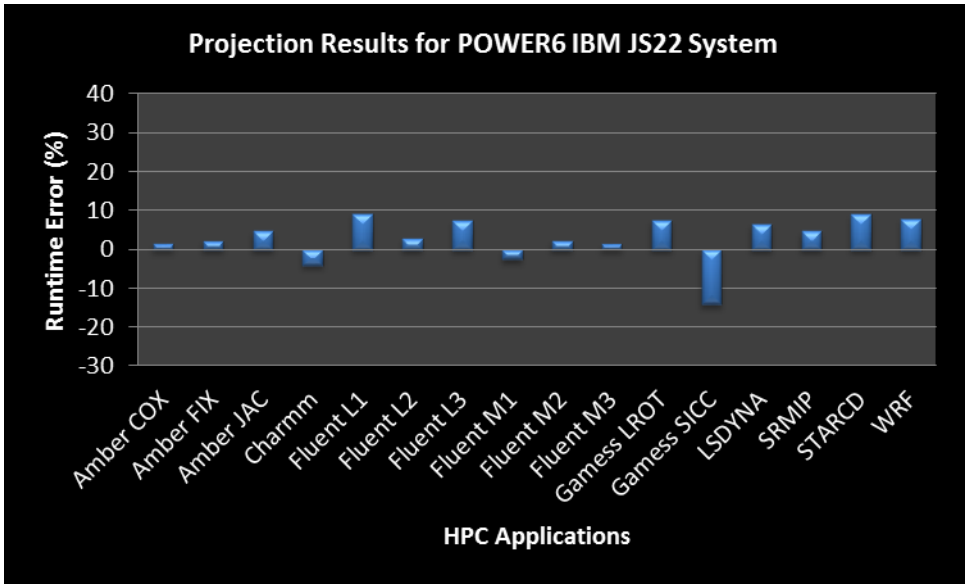

Figure 9: Projection results for POWER6 IBM JS22 system

Figure 9 shows that our scheme was able to predict the performance of the HPC applications within 5.5% average error (based upon magnitudes of the error) on IBM JS22 POWER6 system. Our projection errors are less than 10.0% for all workloads with the exception of GAMESS SICC. GAMESS SICC projection error, although still low, is the only error above 10.0% (14.1%) on JS22. GAMESS SICC requires very little memory bandwidth. When applying our ranking scheme in Section 3.2 for JS22, groups G_2 and G_1 are ranked the highest respectively; however, the GA tool couldn't find a combination of surrogates that are similar to GAMESS SICC G_2 and G_1.

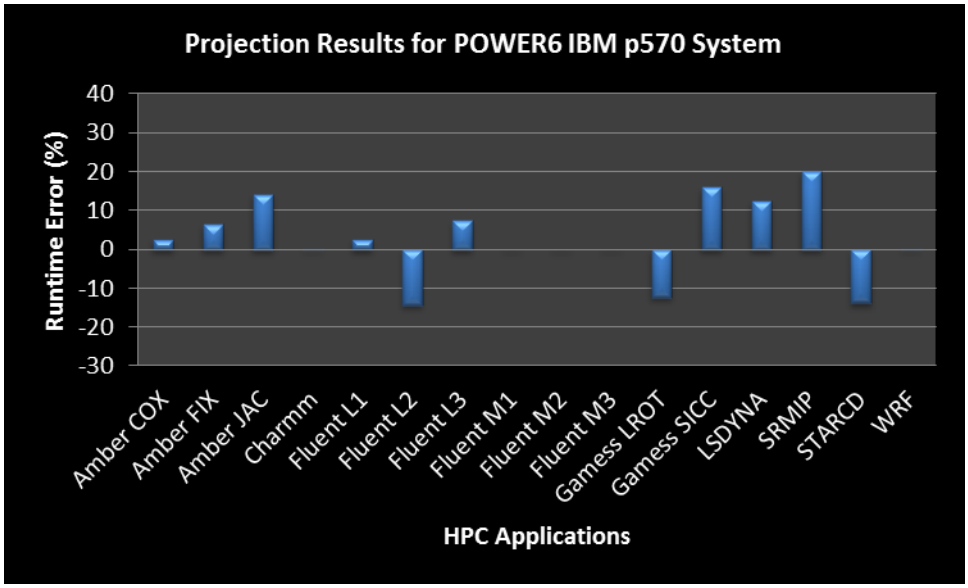

Figure 10: Projection results for POWER6 IBM p570 system

Figure 10 shows the projection results on the IBM p570 POWER6 system. The average projection error for the 16 workloads on the p570 system was 9.8%. With the exception of Seismic and GAMESS SICC, all projection errors are below 15.0%. In fact, only five applications had their projection error between 10.0% and 15.0% while the rest were below 10.0%. The reason GAMESS SICC projection error is 15.9% is due to the same reason as in JS22 system. The Seismic application on the other hand exhibits unique behavior that SPEC CFP2006 doesn't have an application that copies it. Seismic is a bandwidth intensive application; however, good prefetching on the base machine hides the effect of the high bandwidth. Thus, the application doesn't stall waiting on the load-store unit (LSU) and the CPI is low. No benchmark in the SPEC CFP2006 suite exhibits the same behavior and the best combination of surrogates was off on many metrics' groups specially G_1 and G_2. This will be reflected in the other machines as in Figures 11 and 12.

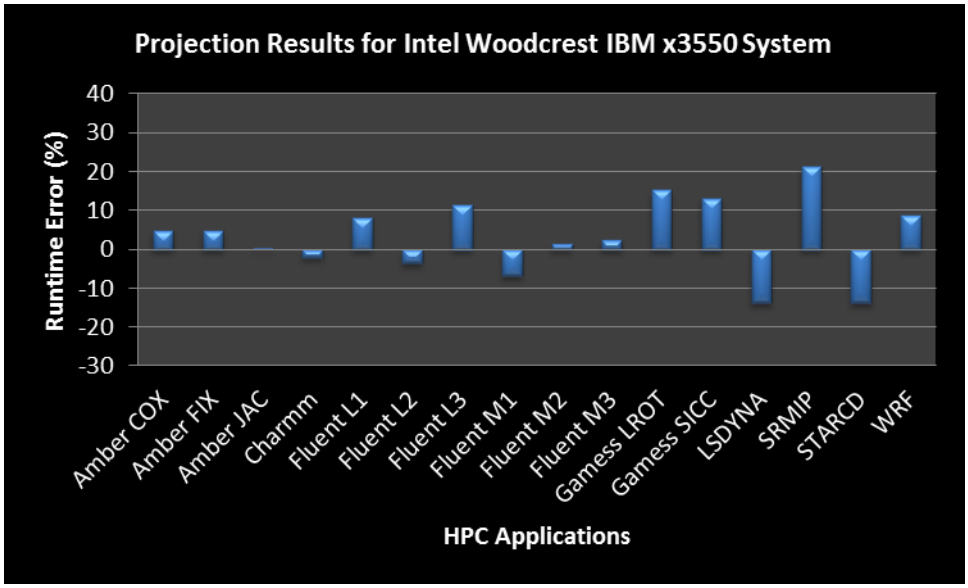

Figure 11: Projection results for Intel Woodcrest IBM x3550 system

Figure 11 shows the projection results on the IBM x3550 system with Intel Woodcrest chip. The average error for the 16 applications is 8.3%. This is very interesting since our scheme projected the performance using hardware performance counters collected on a different system using a chip that has different micro-architecture and different ISA with such accuracy. With the exception of Seismic, for the reasons mentioned above, all projection errors are below 15.0%. In fact, 10 applications had projection errors less than 10.0%.

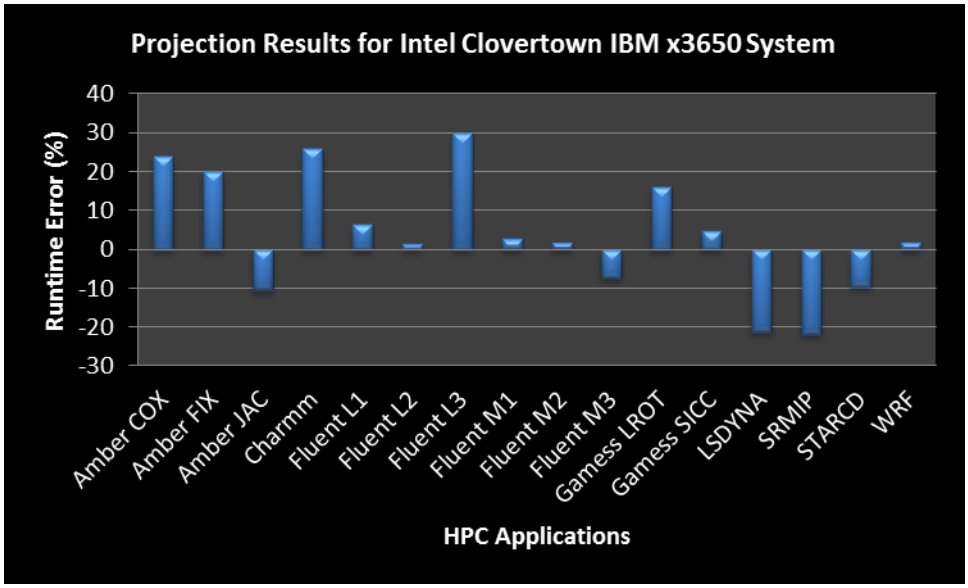

Figure 12: Projection results for Intel Clovertown IBM x3650 system

Figure 12 shows the projection results on the IBM x3650 system with Intel Clovertown MCM. Recall, the Intel Clovertown is an MCM with two Woodcrest chips that share the bandwidth. Thus, Clovertown has a significantly limited bandwidth compared to the base machine. The average projection error for the IBM x3650 system is 12.8%. AMBER FIX and CHARMM are both memory intensive applications. Due to the nature of the Clovertown MCM, G_5 is ranked as the highest group for these two applications; however, the GA tool couldn't identify a combination of surrogates that is quite similar to these two applications in G_5. This dissimilarity between the combined surrogates and these two applications didn't have a significant effect in the projection results for the other systems since G_5 was ranked as high as it is ranked on the IBM x3650 system with the Clovertown MCM. This explains the 19.9% and 25.9% error for AMBER FIX and CHARMM respectively. As for FLUENT L3, with projection error of 29.9%, the highest ranked group was G_6 followed by G_5. The mismatch, however, was in G_6. FLUENT L3 bandwidth in the PST mode is higher than the bandwidth in SMT mode.

This happens in the case of FLUENT L3 because in PST mode each task has more resources than in SMT mode and this allows prefetching to prefetch more data in PST mode than in SMT mode. Again in this case no benchmark in the SPEC CFP2006 suite exhibits the same bandwidth behavior as FLUENT L3. This mismatch effect is exacerbated on Clovertown and doesn't show on other systems since G_6 is ranked as the highest group only on Clovertown. A point worth mentioning, GAMESS SICC mismatch for groups G_1 and G_2 didn't have an effect on the projection on Clovertown because G_1 and G_2 are ranked lower for GAMESS SICC on Clovertown than on the other machines.

Overall, our projection scheme projected with high accuracy using micro-architecture dependent metrics collected on one base system to four different systems utilizing different micro-architecture and different ISA in the case of the Intel systems. When the highly ranked metrics' groups on a system for a certain application has significantly high metrics' error, the projection results for this application on that system are comparatively high and our scheme indicate that those projections are not very accurate.

3.5 Related Work

Several researches have been done on using surrogate workloads to predict application performance. SPEC benchmarks suite was often proposed as the benchmarks of choice due to the abundance of published data but it was not used for HPC applications. NAS Parallel [59] benchmarks, on the other hand, were used more often with HPC applications due to their parallel nature. Also curve fitting on runtimes is extensively used in industry to project performance using surrogate workloads.

Todi and Gustafson [35] mapped applications to the HINT benchmark curve and then used the HINT curve for a given machine to predict the application performance. They showed that HINT is a superset for the other benchmarks included in the study, NAS Parallel, SPEC, STREAM and others. The main goal of their work was to find the correlation between HINT and the other benchmarks indicating that HINT is a superset of these benchmarks and then using it in prediction.

Phansalkar [60] used hardware performance counter experimentation to categorize the SPEC CPU2006 benchmarks. His work used statistical techniques such as principal component analysis and clustering to draw inferences on the similarity of the benchmarks and the redundancy in the suite and arrive at meaningful subsets. In his paper, he didn't extend the work to involve performance projection.

Hoste [34] proposed the use of SPEC CPU2000 in performance projection of applications. His scheme was to measure a number of micro-architecture-independent characteristics from the application of interest, and relate these characteristics to the ones of the programs from SPEC 2000. Based on the similarity of the application of interest with programs in the benchmark suite, he made a performance prediction of the application of interest. He proposed and evaluated three approaches (normalization, principal components analysis and genetic algorithm) to transform the raw data set of micro-architecture independent characteristics into a benchmark space in which the relative distance is a measure for the relative performance differences. His work was not extended to HPC applications as this paper does. In addition, he used binary instrumentation instead of hardware performance counters collected on several systems to create the data matrix not just one base machine. Also, Hoste's main goal was to predict

machine ranks for applications rather than actual performance of application on certain target machine which makes his accuracy measurements different than ours.

Tikir [33] used genetic algorithms approach to model the performance of memory-bound computations. He proposed a scheme for predicting the performance of HPC applications based on the results of MultiMAPS benchmarks. A Genetic Algorithm approach was used to "learn" bandwidth as a function of cache hit rates per machine with MultiMAPS as the fitness test. His approach differs than what we propose in this paper in many aspects. His scheme works only on memory bound applications while ours can be used on all applications. His approach requires simulating different cache sizes to understand the cache characteristics of the application while we use performance counter measurements with SMT and PST mode. Our approach doesn't require instrumentation of binary code as we depend on hardware performance counters collected by simply executing the binary on a base machine. Also, Tikir approach is tightly coupled to MultiMAPS as the set of benchmarks. Our approach can use any set of benchmarks or several sets of benchmarks.

3.6 Summary

We presented a scheme to project the performance of HPC applications using surrogate workloads from the SPEC CPU2006 benchmark suite and hardware performance counter data. The scheme is very flexible since it doesn't require any instrumentation to the binary code or the source code and only requires execution of the application and the benchmarks on one base machine. Moreover, simulation is not needed

eliminating the long runtimes incurred in simulations. The use of a string based genetic tool reduces the projection scheme runtime significantly.

SPEC CPU2006 being developed as a serial version of real parallel applications covers a large range of HPC applications' space but not the entire range as in the case of Clovertown projections. However, our scheme is not tightly coupled to SPEC CPU2006 and can easily incorporate other benchmark suites such as SPEC MPI2007, NAS Parallel Benchmarks or others. The choice of SPEC CPU2006 was mainly because of its abundant published data and its similarity to real HPC applications.

Our scheme uses hardware performance counter data for the HPC applications and the SPEC CPU2006 suite from one base machine to model the behavior of the HPC applications as a function of the benchmarks of SPEC CPU2006. The model of the HPC application characterizes its behavior based upon a common set of benchmarks using hardware performance counter data. This model gives an insight on the nature of the application, category (Fluid Dynamics, Weather, etc...) and what is the best system it would run on.

Also, in our scheme, we combine the runtimes of the benchmarks on the base machine and on the target with the performance metrics to architecturally characterize each system we are projecting on. This architectural characterization allows for understanding the relation between the behavior of the application and the target architecture. This understanding gives us insight on which metrics are of more significance to the behavior of the application on the target system allowing for better projection results. Furthermore, our scheme has the ability to point out possible inaccurate projections based on the rankings of the metrics groups on the target machine.

For those applications with the highest ranked group(s) having significantly higher metrics' errors, our scheme indicates that those projections may be inaccurate.

4. COMMUNICATION COMPONENT PERFORMANCE PROJECTION

In section 3, we presented our method to project the performance of only the computation component of an HPC application using published data of industry standard benchmarks, the SPEC CPU2006, and hardware performance counter data from one base system. In this section, we extend our method to project the communication performance of HPC applications onto different systems using MPI benchmark data on the different systems as well as a base system, and the communication profile of the application on the base system. In particular, we use the Intel MPI Benchmarks (IMB) [32] as we find it the most comprehensive MPI benchmark suite. The main advantage of our method is the use of MPI profiles of the HPC application instead of MPI traces, which require significant storage and are very complicated, hard to understand and parse. For example, our method requires 12KB storage for the communication profile of the NAS BT benchmark, in contrast to 2.6GB storage for an MPI trace of the communication behavior of the same benchmark for 128 tasks. Further, our method does not involve any simulations, which are often very time-consuming.

Figure 13 depicts the high level framework of our method. The HPC application is executed multiple times on the base machine where each execution utilizes different number of cores C_j for $j \in \{1,..,c\}$ and c is the maximum number of cores the HPC application can utilize. During each execution, we obtain the application's MPI profile for C_j number of cores. The resultant MPI profiles are used to produce an MPI communication model for the HPC application. The model is a function of the number of

cores, the MPI routines (e.g. MPI_Bcast, MPI_Reduce, etc), the mess age sizes, and the number of calls for each routine. The MPI communication model provides for understanding the HPC application scaling of each MPI routine for different C_j. In addition, the performance of IMB benchmarks, is obtained for the base and the target machines for different numbers of cores, C_j. Target system parameters provide the performance of each MPI routine from the IMB benchmark for different message sizes at each C_j on the target system. The application *WaitTime* model on the base machine is generated by analyzing the load imbalance of the application using the IMB data on the base machine and the application MPI profile data on the base machine. In the final step of our method, we combine the application MPI model on the base machine with the *WaitTime* model on the base machine and the IMB parameters for the target machine to project the HPC application MPI communication performance on the target system.

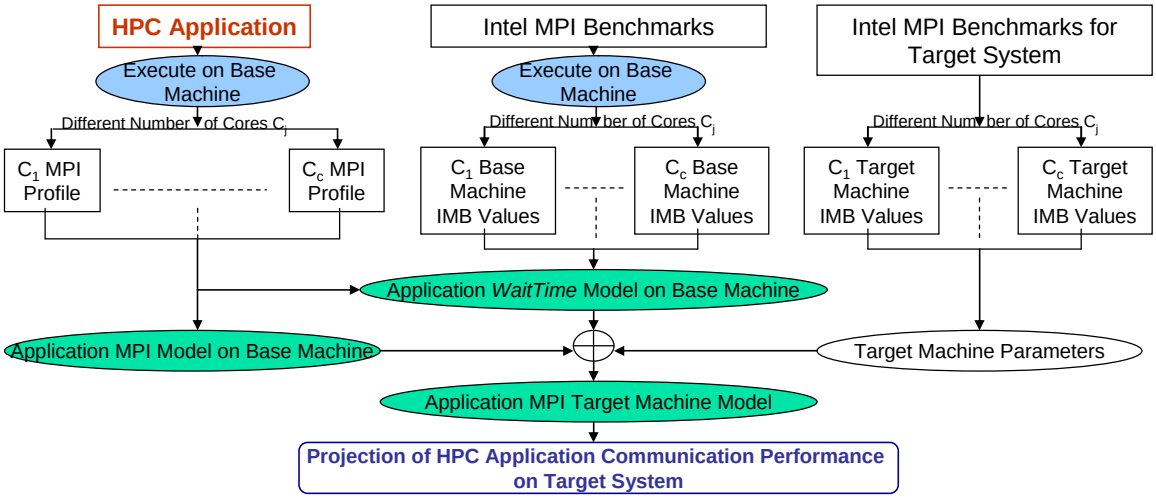

Figure 13: Framework for MPI communication projection scheme

The use of MPI profiles in projecting the communication performance of HPC applications allows for an efficient and fast projection scheme. In contrast to MPI traces, MPI profiles do not have the level of detail found in MPI traces, such as traces obtained by MPIDtrace, which often obscures the high level behavior of the communication component of an HPC application. In addition, obtaining MPI traces is a slow process which often results in massive trace files. These massive trace files are typically too complicated to parse. Also using MPI trace files as input to network simulators is extremely time consuming. On the other hand, MPI profiles are easy and simple to obtain and understand. MPI profiles provide a high level picture of the MPI communication component of an HPC application.

4.1 MPI Profile

An MPI profile is a high level representation of the communication component of an HPC application. This high level representation provides for speed in collection and efficiency in modeling An MPI profile consists of a summary of all MPI routines called by the application, their message sizes and runtime of each call as well as the total runtime of the program. The frequency of each call and its contribution in the overall runtime can be calculated from the profile data. To obtain the MPI profile, one links the HPC application to the profiling library. Once linked, the MPI profile is produced during execution and an output file of size 12KB (for a 128 tasks) is produced. The profile is produced on a per task basis, i.e. each task has its own MPI profile. In this work, we used an MPI profiling library available with the IBM Parallel Environment [61]. The information in the profile is as follows:

1. A summary of all MPI routines the application called and the aggregate timing for each routine. The profiling starts with the MPI_Init call and ends at the MPI_Finalize.
2. Message sizes distribution. The message size distribution breaks down each MPI routine into message sizes for this MPI routine, number of calls for this specific message size and the aggregate time for the calls.
3. The breakdown of total execution time for each task. This breakdown involves the percentage of execution time spent doing computation and the percentage spent in communication. The communication percentage also includes time spent waiting in an MPI_Waitall for example.

4.2 Intel Benchmarks and Target Machine Parameters

The IMB, explained in details in section 2.3.3, is the benchmark suite of choice for most hardware vendors and researchers in measuring the communication and interconnect architecture of a system. It checks many MPI communication patterns and automatically detects clustering, and reports intra-cluster and inter-cluster performance. IMB is targeted at measuring Point to Point MPI communication and Collective MPI communication. Also, IMB measures performance for different message sizes.

In this work, we use IMB benchmarks to measure the performance of the communication and interconnect architecture of both, the base and target, systems. IMB provides a concise set of benchmarks targeted at measuring the most important MPI functions. In addition to the default set of benchmarks included in IMB, we add one extra benchmark, *multi-Sendrecv*. The *multi-Sendrecv* benchmark measures the performance of

the MPI library and the underlying interconnect when multiple successions (one or more) of non-blocking point to point calls (MPI_Isend and MPI_Irecv) are issued followed by an MPI_Waitall as in Figure 14. Note that *multi-Sendrecv* benchmark defined here is different than IMB Multi-Sendrecv. In a blocking point to point call, the control returns back to the user only when the user buffer can be safely used. IMB benchmarks such as PingPong utilize such calls to measure the interconnect network latency and bandwidth. Contrary to non-blocking calls, control returns to user before the buffer can be safely used. Thus, understanding the behavior of a sequence of non-blocking calls followed by an MPI_Waitall requires special handling in order to accurately parameterize the non-blocking calls on the base and target machines.

```
do{
      NUM_ISEND_IRECV++;
      Time_Stamp(entry);
      for(i=0;i< ITERATIONS;i++)
      {

      for(int j=0; j < NUM_ISEND_IRECV; j++)
      {
         MPI_Isend(destination);
         MPI_Irecv(source);
      }
      MPI_Waitall;
      }
      Time_Stamp(exit);
}
while
(!multi-Sendrecv_performance_calculated());
/* Calculate T_LibraryOverhead and T_inFlight */
```

Figure 14: multi-Sendrecv benchmark

In the case of blocking MPI routines, the execution time measured by IMB is the total time from the instance the MPI routine is called until the control returns back to the caller. This measured time includes the time the message takes in the interconnect network (time of flight) to reach the destination in addition to the software library overhead on both the send and receive sides. In the case of the *multi-Sendrecv* benchmark, the measured execution time is the entire time span from the instance of the first MPI_Isend/MPI_Irecv is called until the completion of the MPI_Waitall as indicated in Figure 14. Therefore, we can define this measured time $T_{Transfer}(m_i)$ for an MPI routine $m_i, \forall i \in M$ where M is the set of all MPI routines as:

$$T_{Transfer}(m_i) = T_{LibraryOverhead}(m_i) + xT_{inFlight}(m_i), \forall i \in M \quad (11)$$

$T_{LibraryOverhead}(m_i)$ is the time in the MPI software library from the instance the call to MPI routine m_i is issued until the message reaches the interconnect hardware for transferring. This $T_{LibraryOverhead}(m_i)$ is typically very little compared to the $T_{inFlight}(m_i)$. The second portion of Equation 11, $xT_{inFlight}(m_i)$ represents the $T_{inFlight}(m_i)$ that it takes the message, in the interconnect, for an MPI routine m_i to reach the destination multiplied by x which is the number of messages in flight. For a typical MPI routine such as MPI_Bcast or MPI_Sendrecv we assume x is 1. In the case of *multi-Sendrecv* benchmark, x is the number of MPI_Isend/MPI_Irecv calls issued before the MPI_Waitall. The *multi-Sendrecv* benchmark measures the execution time for different number of successions of MPI_Isend and MPI_Irecv followed by MPI_Waitall. This allows for finding the $T_{LibraryOverhead}(m_i)$ and $T_{inFlight}(m_i)$ in Equation 1. Note that $T_{Transfer}(m_i)$ doesn't include any time MPI_Waitall spends waiting because of load imbalance between tasks especially load imbalance in computation and communication. The time MPI_Waitall spends

waiting because of load imbalance called *WaitTime* is defined later in details in Section 4.3. Note that a similar benchmark to the *multi-Sendrecv* is required for the one sided MPI2 routines which follows the same idea.

Recall that target machine parameters are used to understand the performance of each MPI routine on the target system, the change in performance of these call types with different message sizes and the scaling with different number of cores C_j. IMB benchmarks, as previously mentioned, provide the performance of each MPI routine for different message sizes utilizing different number of cores. Thus, using IMB we can obtain target machine parameters which can be represented by Equation 12.

$$P_{C_j}(m_i, S_k), \forall j \in \{1,...,c\}, i \in M, k \in \{1,...,s\} \quad (12)$$

From Equation 12, a target machine parameter P indicates the performance (in execution time) for an MPI routine m_i and message size S_k at core count C_j where c is the maximum number of cores the HPC application can utilize, M is the set of all MPI point to point and collective routines and s is the maximum message size feasible on the target system. To illustrate, parameter P for an MPI_Bcast, message size 1 and core count 32 would indicate the time it takes to broadcast a message of size 1 byte to 32 cores on the interconnect of the target machine.

Parameter P for the *multi-Sendrecv* benchmark represents performance for MPI tasks on the target system placed in a ring topology. This is similar to IMB *Sendrecv* where the processes form a periodic communication chain. Each process sends to the right and receives from the left neighbor in the chain. Thus, P for *multi-Sendrecv* doesn't consider queuing delays in interconnect due to different communication patterns such as closest neighbors.

Target machine parameters can be obtained through target machine simulators if the system doesn't exist yet or through direct execution on the platform if it exists and available. In the case of future systems, simulating IMB is a much simpler task than simulating the actual HPC application. In case of an existing system, executing IMB on the system may also be simpler than executing the actual HPC application especially if the HPC application is incompatible with the target system OS. In addition, if the target system required is larger than the available system, regression based approaches to extrapolate IMB values for a larger system is simpler than extrapolating performance of the HPC application due to the non-linear behavior of HPC applications.

4.3 Defining *WaitTime*

The communication benchmarks do not entail any computation. Also, all tasks send the same number of messages with the same message sizes. Therefore, there is no load imbalance between tasks and all the performance data illustrated by IMB and the *multi-Sendrecv* benchmarks reflect the performance of the interconnect architecture. However, HPC applications typically involve communication and computation components. In addition, computation and communication may not be balanced among all tasks, i.e. the ratio of computation to communication is not the same among all tasks. This imbalance results in some tasks idly waiting for other tasks to finish before continuing on with their next phase of the compute iteration or timestep. We call this idle waiting *WaitTime*. Therefore, the performance of the MPI communication component of an HPC application represented in elapsed time can be defined as:

$$T_{Elapsed}(m_i) = T_{Transfer}(m_i) + T_{Wait}(m_i), \forall i \in \overline{M} \quad (13)$$

$T_{Transfer}(m_i)$is the transfer time for an MPI routine $m_i, \forall i \in \overline{M}$ where $\overline{M}$ is the set of all MPI point to point and collective routines utilized by the HPC application as defined in Equation 11. Note that the set $\overline{M} \subseteq M$ since $T_{Elapsed}(m_i)$ is specific to an HPC application which may not utilize all MPI routines in set *M*. The $T_{Wait}(m_i)$, on the other hand, is the *WaitTime* elapsed due to load imbalance especially between computation and communication among different tasks for MPI routine m_i. Note that $T_{Transfer}(m_i)$ defined in Equation 11 does not include $T_{Wait}(m_i)$ due to load imbalance. To illustrate, consider the *multi-Sendrecv* benchmark defined in Section 4.2. Recall, in the *multi-Sendrecv* benchmark successions of MPI_Isend/MPI_Irecv are called followed by an MPI_Waitall. Since in the benchmark there are no computations taking place, no imbalance due to computation occurs, and $T_{Transfer}(multi\text{-}Sendrecv)$ corresponds to $T_{LibraryOverhead}(multi\text{-}Sendrecv)$ and $T_{inFlight}(multi\text{-}Sendrecv)$; thus, no waiting time on computation completion occurs, i.e. no $T_{Wait}(multi\text{-}Sendrecv)$. Imbalance between tasks due to network times in the IMB is included in the $T_{Transfer}(m_i)$ component reported by IMB.

4.4 Performance Projection Scheme

Recall that, we introduce a scheme to project the MPI communication performance of HPC applications using MPI profiles obtained on one base machine. As indicated in Figure 13, we obtain MPI profiles for the HPC application for all core counts C_j where $j \in \{1,..,c\}$. These profiles are then used to create the HPC application MPI communication model explained in details in section 4.4.1. The same MPI profiles are used in conjunction with IMB data for the base system to create the *WaitTime* model due to load imbalance explained in section 4.4.2. To understand the performance of each MPI

routine on the target system, the change in performance with different message sizes and the scaling with different number of cores C_j, IMB is used to obtain the target machine parameters. In our projection scheme, we combine the HPC application MPI communication model, the *WaitTime* model and the target system parameters to produce the HPC application MPI performance model on the target system. This model is used to project the performance of the MPI communication component of the HPC application on the target system. In this section, we discuss the details of the MPI profile, IMB benchmarks and how these benchmarks are used to create target machine parameters. We also define the concept of *WaitTime* due to load imbalance in HPC applications.

The process of performance projection of HPC applications, in this work, entails three steps.

1. Characterizing/modeling the MPI communication component of each HPC application. The HPC application MPI communication model is a function of MPI routine, message sizes for these routines and the number of calls for each of these message sizes at each C_j.
2. Modeling the *WaitTime*, defined in section 4.3, due to load imbalance in the HPC application.
3. Combining the MPI communication model, the *WaitTime* model and the target system parameters to produce the HPC application target system MPI communication model which is used to project the communication performance of the HPC application on the target system.

4.4.1 HPC Application MPI Communication Model

The MPI communication Model provides for understanding the HPC application scaling of each MPI routine for different C_j. The target machine parameters detailed in Section 4.2 are target system specific; however, the HPC application MPI communication Model is an HPC application specific model independent of the system the application is executing on. Therefore, we define an MPI communication model *MM* for an HPC application *A* with a specific problem size *D*. The problem size *D* indicates that the dataset size and the algorithm used by an application *A* are constant for all C_j. Furthermore, an application *A* with problem size *D* will utilize a set of MPI routines $\overline{M} \subseteq M$ where *M* is the set of all MPI point to point and collective routines as indicated in Section 4.3.

The *MM* model identifies how each MPI routine scales with different number of cores C_j. For each MPI routine, the message size and number of calls change with changing the number of cores. We define the change in message size for an MPI routine as:

$$S_{m_i C_j} = \alpha_{m_i} \times S_{m_i C_1}, \forall i \in \overline{M}, j \in \{1,...,c\} \tag{14}$$

where α_{m_i} defines the scaling function of message sizes for an MPI routine m_i and $S_{m_i C_1}$ is the message size for an MPI routine m_i at core count C_1 (least possible core count for application *A* with problem size *D*). We can also define the change in number of calls as:

$$N_{m_i C_j} = \beta_{m_i} \times N_{m_i C_1}, \forall i \in \overline{M}, j \in \{1,...,c\} \tag{15}$$

where β_{m_i} defines the scaling function of number of calls for an MPI routine m_i and $N_{m_iC_1}$ is the number of calls for an MPI routine m_i at core count C_1 (least possible core count for application *A* with problem size *D*). Thus *MM* model for an MPI routine m_i can be defined as the ordered pair in Equation 16:

$$MM(m_i)_{C_j} = (S_{m_iC_j}, N_{m_iC_j}), \forall i \in \overline{M}, j \in \{1,...,c\} \tag{16}$$

Using MPI profiles for an HPC application *A* with problem size *D* at different core counts C_j, we can solve for α_{m_i} and β_{m_i} $\forall i \in \overline{M}$. The α and β scaling functions can be as simple as a constant value or a complicated function that depends on multiple variables such as MPI rank and dataset dimensions. To illustrate, the β for an MPI_Bcast can be 1, which means the number of broadcast messages a task sends is constant no matter what the number of cores are. For the same MPI_Bcast, the α can be 2, which means that the message sizes doubles by doubling the number of cores.

4.4.2 *WaitTime* Model

In Section 4.3 we defined the *WaitTime* in an HPC application as the idle time a task spends waiting for other tasks to finish before continuing on with their next phase of the compute iteration or timestep. This *WaitTime* is mainly due to load imbalance between computation and communication among different tasks. On the base machine, *WaitTime* can be modeled accurately using MPI profiles for the HPC application and IMB+*multi-Sendrecv* benchmark data obtained on the base machine. To illustrate, an HPC application MPI profile, as indicated in Section 4.1, includes the MPI message type,

message size for each type, number of calls for each message size and the elapsed time for these calls, i.e. $T_{Elapsed}$ defined in Equation 13 which includes $T_{Transfer}$ and T_{Wait}. The IMB+*multi-Sendrecv* benchmark data will show the transfer time each MPI routine will take to complete, i.e. $T_{Transfer}$ defined in Equation 11. By subtracting the IMB transfer time, $T_{Transfer}$, from the MPI profiles elapsed time, $T_{Elapsed}$, we obtain the T_{Wait} on the base machine. Therefore we can define the *WaitTime* model of an HPC application on the base machine as:

$$T_{Wait_{base}}(m_i)_{C_j} = T_{Elapsed_{base}}(m_i)_{C_j} - T_{Transfer_{base}}(m_i)_{C_j}, \forall i \in \overline{M}, j \in \{1,...,c\} \quad (17)$$

where $\overline{M}$ is the set of all MPI point to point and collective routines utilized by an HPC application *A* and *c* is the maximum core count that *A* can utilize. Recall that $T_{Transfer}$ for point-to-point routines and *multi-Sendrecv* in IMB represents time measured for certain communication patterns as indicated in Section 4.2, e.g. ring pattern for *multi-Sendrecv*. The communication patterns in the application may differ than those in IMB. This difference is a source of error in projection. In this paper we focus on how to utilize existing standard MPI benchmarks such as IMB. It is expected that as the MPI benchmarks evolve to include a larger set of communication patterns our method will have a better approximation to the application.

From Equation 17, since $T_{Elapsed_{base}} \geq T_{Transfer_{base}}$, then the $T_{Wait_{base}} \geq 0$. In the case of a blocking collective MPI routine where all tasks are synchronized, load imbalance is highly reduced and $T_{Wait_{base}}$ approaches 0. On the other hand, in the case of non-blocking MPI routines, load imbalance is higher and $T_{Wait_{base}}$ increases.

4.4.3 HPC Application Target System Communication Model

In this final step of our projection scheme, our goal is to model the communication component of the HPC application on the target system. This model is then used to project the performance of the HPC application. Using Equation 13, we can define the communication of the HPC application on the target system as:

$$T_{Elapsed_{target}}(m_i) = T_{Transfer_{target}}(m_i) + T_{Wait_{target}}(m_i), \forall i \in \overline{M} \quad (18)$$

From Equation 18, identifying the $T_{Transfer}$ value for the target system as well as the T_{Wait} value, one can project the performance of the communication component of the HPC application.

From the definition of $T_{Transfer}$ in Section 4.2, $T_{Transfer}$ on the target system can be obtained by combining the *MM* model for the HPC application with the parameters *P* for the target system. The *MM* model defined in Equation 16 provides for the message size $S_{m_iC_j}$ of an MPI routine m_i at core count C_j. Also *MM* provides the number of calls $N_{m_iC_j}$ for message size $S_{m_iC_j}$. Using $S_{m_iC_j}$ and $N_{m_iC_j}$ for the HPC application from Equation 16 and the target system parameter *P* from Equation 12 we can represent the $T_{Transfer}(m_i)$ on the target system as:

$$T_{Transfer_{target}}(m_i)_{C_j} = P_{C_j}(m_i, S_{m_iC_j}) \times N_{m_iC_j}, \forall i \in \overline{M}, j \in \{1, \ldots, c\} \quad (19)$$

From Equation 19, the $T_{Transfer}(m_i)$ for the target system is the aggregate transfer time for number of calls $N_{m_iC_j}$ to MPI routine m_i, with message size $S_{m_iC_j}$ obtained from *MM* model at core count C_j. Single message time is obtained from the target system parameter *P*.

The remaining portion of the communication time, from Equation 18, on the target system is the $T_{wait}(m_i)$ on the target. To project the *WaitTime* on the target machine, we need to find the scaling factor in *WaitTime* performance from the base to the target system, *SF*. According to *WaitTime* definition in Section 4.3, *WaitTime* is due to load imbalance between computation and communication in an HPC application. Therefore, *WaitTime* scaling factor depends on scaling in computation and communication performance from the base system to the target. *SF* can be defined by Equation 20:

$$SF(m_i)_{C_j} = w_{m_i C_j} \times SF_{comm}(m_i)_{C_j} + w_{compC_j} \times SF_{comp}(comp), \forall i \in \overline{M}, j \in \{1,...,c\} \quad (20)$$

where the weight w_{m_i} is the MPI routine m_i percentage in total elapsed time and the weight w_{comp} is the computation percentage in total elapsed time. w_{m_i} and w_{comp} can both be obtained directly from the MPI profile for the application. $SF_{comm}(m_i)$ is the scaling factor in communication of the MPI routine m_i from base to target and is calculated using the IMB data for the base and target systems. The $SF_{comp}(comp)$, on the other hand, is the scaling factor in computation from base to target and is calculated using our computation projection methodology presented in section 3. Notice that the $SF_{comp}(comp)$ in Equation 20 is not dependent on number of cores C_j as the other components of the equation. Using the *SF* from Equation 20 and $T_{Wait_{base}}$ obtained in Equation 17, we can obtain the projected *WaitTime* on target as in Equation 21:

$$T_{Wait_{target}}(m_i)_{C_j} = SF(m_i)_{C_j} \times T_{Wait_{base}}(m_i)_{C_j}, \forall i \in \overline{M}, j \in \{1,...,c\} \quad (21)$$

Once this step is complete, we complete the projection for the two portions in Equation 18, hence the projection of the communication performance on the target system.

The projection of the *WaitTime* on the target system is dependent on the value for $SF(m_i)$ as indicated in Equation 21. From Equation 20, $SF(m_i)$ calculation depends on four components:

- w_{m_i} which is the percentage in the total elapsed time for MPI routine m_i obtained on the base system. This percentage could change for the target system; hence, w_{m_i} component affects projection accuracy for *WaitTime*.
- w_{comp} which is the percentage of computation in the total elapsed time on the base system. This percentage could change on the target system; hence, w_{comp} component affects projection accuracy for *WaitTime*.
- $SF_{comm}(m_i)$ which is the scaling factor for MPI routine m_i from the base to target. This component of Equation 10 is calculated using the IMB values obtained for the base and target systems. For each MPI routine m_i, IMB reports an average value over all tasks. These averages obscure task placement effect on results. This may slightly affect projection accuracy.
- $SF_{comp}(comp)$ which is the scaling factor for computation from base to target. This factor is obtained using scheme presented in section 3. Thus, accuracy of projecting the computation scaling factor directly affects the accuracy of projecting *WaitTime*.

4.5 Experimental Results

We used the methodology depicted in Figure 13 to project the communication performance of the three NAS Multi-Zone benchmarks BT-MZ, LU-MZ and SP-MZ.

The choice of the Multi-Zone benchmarks was mainly for their OpenMP capabilities since we want to extend this work to encompass Hybrid MPI/OpenMP HPC applications. All NAS-MZ benchmarks were compiled for classes C and D in our validation experiments. Details on benchmarks are provided in Table 8. We used the TAMU Hydra system as our base machine. We projected the performance onto two target systems, an IBM internal POWER6 575 cluster system and an IBM internal BlueGene/P system. Details of the three systems are listed below in Table 9. Throughout our validation process, the IMB benchmarks and NAS-MZ benchmarks are executed using the same MPI library on a system. Also they both follow the same task placement strategy for consistency. On the BlueGene/P system, our experiments were all done using the "Virtual Node" mode where four MPI tasks are utilizing the four cores per node. On the TAMU Hydra and the IBM POWER6 575 systems, the Single Thread (ST) mode was utilized.

Table 8: NAS-MZ benchmarks characteristics on base system for 16-128 tasks

Benchmark	Communication percent (16 tasks – 128 tasks)	*multi-Sendrecv* percent (16 – 128)	Reduce and Bcast percent (16 – 128)
BT-MZ C	3.2 - 59.7	3.17 - 59.1	0.032 - 0.59
LU-MZ C	1.4	1.38	0.014
SP-MZ C	4.8 – 16	4.75 - 15.84	0.048 - 0.016
BT-MZ-D	2.3 - 6.8	2.27 - 6.7	0.023 - 0.068
LU-MZ D	1.2	1.18	0.012
SP-MZ D	4.16 - 6.6	4.1 - 6.5	0.041 - 0.066

Table 9: Base system and the different systems used for validation

Machine	Processor	Total Cores	Cores Per Node	Memory Per Core	Interconnect
TAMU Hydra	POWER5+	832	16	2GB	Federation
IBM POWER6 575 cluster	POWER6	128	32	4GB	InfiniBand
BG/P	PowerPC 450	4096	4	1GB	3D Torus/ Collective Tree

The first step in our validation process is to validate the *MM* model and how accurate it is in modeling the scaling of each MPI routine utilized by the HPC application. Since the *MM* model is an HPC application specific and independent of the underlying system, we validated the *MM* model for the BT-MZ and SP-MZ on the BlueGene/P system (LU-MZ only utilizes 16 cores and doesn't scale). We used four core counts (16, 32, 64 and 128) in order to create the *MM* model. We then validated the *MM* model for 256, 512 and 1024 task count. The accuracy was 100% for the five MPI routines utilized (MPI_Isend, MPI_Irecv, MPI_Waitall, MPI_Reduce and MPI_Bcast).

In the results figures below we show the signed value of the error in communication time; however, as in the computation projection, we focused on reducing the magnitude of the error and all the averages are based on absolute errors. The *WaitTime* for the MPI_Reduce and MPI_Bcast for the three NAS-MZ benchmarks was essentially zero. For the MPI_Isend, MPI_Irecv and MPI_Waitall routines, the *WaitTime* was a major component of their communication time. The results here indicate the

aggregate error for both the $T_{Transfer}$ and T_{Wait} components i.e. overall communication error for each MPI routine. Note that the MPI_Isend, MPI_Irecv and MPI_Waitall are equivalent to our *multi-Sendrecv* benchmark with $x = 1$. In all our experiments, MPI_Isend, MPI_Irecv and MPI_Waitall in the NAS-MZ benchmarks are represented as *multi-Sendrecv* with $x = 1$.

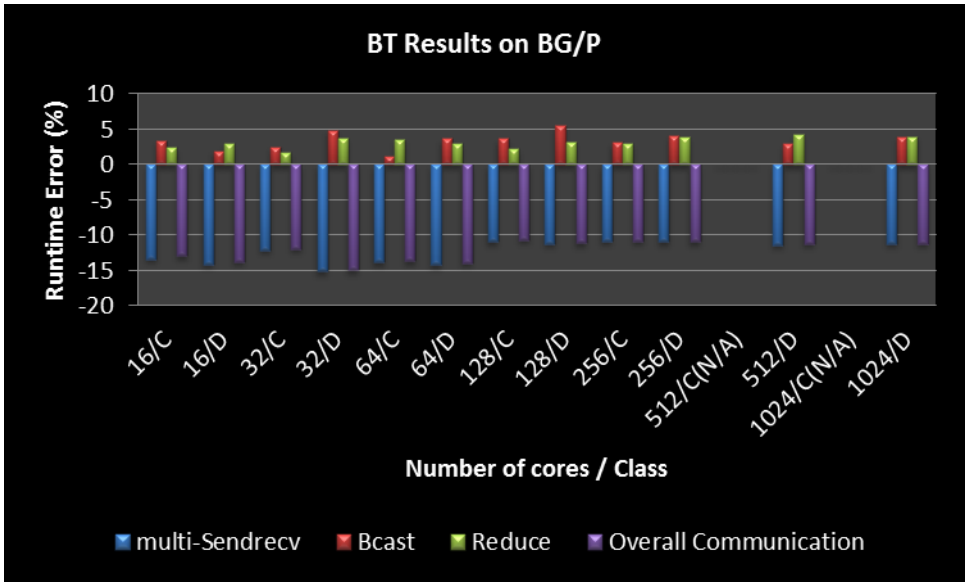

Figure 15: BT results on BG/P

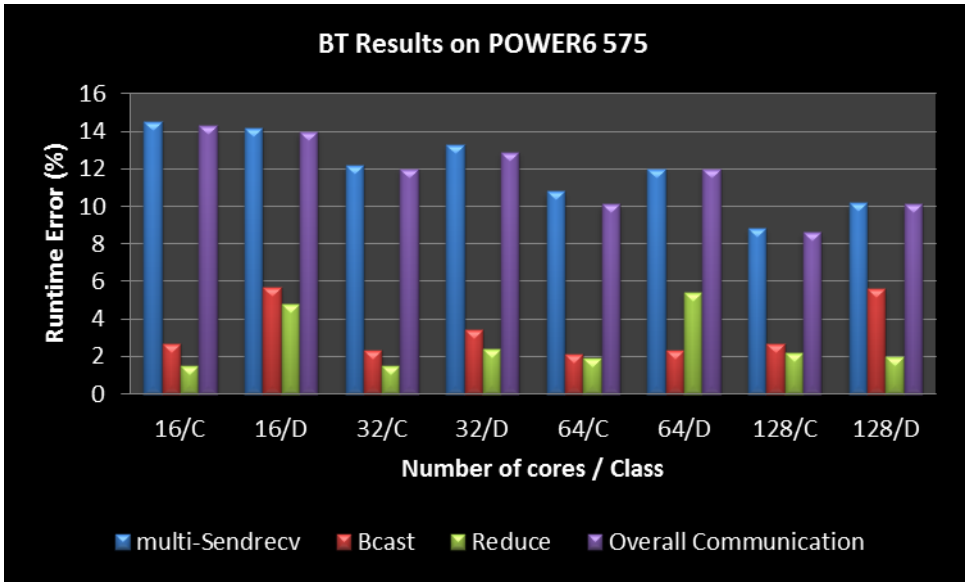

Figure 16: BT results on POWER6 575

Figures 15 and 16 show the results of our communication projection scheme for the BT-MZ benchmark. The results indicate that our scheme accurately projected the communication performance for the BT-MZ with an average error of 12.22% and standard deviation of 1.48% on the BG/P system and 11.74% average error with standard deviation of 2.01% on the POWER6 575 cluster system. An obvious trend on both systems is that the magnitude of the error is less for larger number of cores. This is due to the fact that *WaitTime* component in the communication decreases as more time is being spent in the interconnect ($T_{inFlight}$). Therefore, T_{Wait} becomes of less significance in the overall projected elapsed communication time. The projection error for BT-MZ on BG/P system is mostly attributed to the $SF_{comp}(comp)$ component of Equation 20 where the projection error for the computation component on BG/P was ranging between 7%-10%. On the POWER6 575 system, on the other hand, the projection error for *WaitTime* is due to the change in ratio between computation and communication from the base to the

POWER6 575 system (w_{comp} and w_{m_i} components in Equation 20). Also, on the BG/P system, the computation projection error was obviously the main factor impacting the communication projection error. As indicated in figure 15, computation projection error being a negative value forced the communication error to also be of a negative value. This trend on BG/P system is consistent for the three NAS benchmarks.

In Figure 17 we show the projection results for the LU-MZ benchmark. As indicated in Figure 17, the projected communication time for the collectives that exhibit almost no *WaitTime* is highly accurate. On the other hand, the *multi-Sendrecv* with a large *WaitTime* component has a less accurate projection. The projection error for *WaitTime* component in the *multi-Sendrecv* ranges between 14% and 15% for LU-MZ. On both BG/P and POWER6 575 systems, this is attributed to the computation projection error of 14% and 12% respectively affecting the $SF_{comp}(comp)$ component in Equation 20.

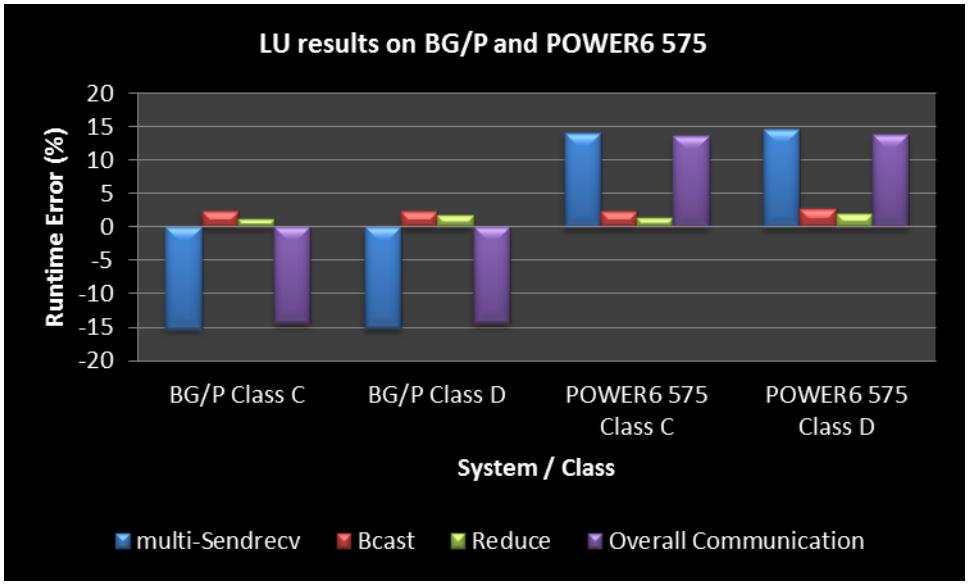

Figure 17: LU results on BG/P and POWER6 575

Figures 18 and 19 show the results of our communication projection scheme for the SP-MZ benchmark. The results indicate that our scheme accurately projected the communication performance for the SP-MZ with an average error of 11.89% and standard deviation of 2.13% on the BG/P system and 11.92% average error with standard deviation of 2.48% on the POWER6 575 system. A similar trend to BT-MZ where the magnitude of error is less for larger number of cores can be noticed here. The projection error for *WaitTime* on BG/P system utilizing 16 and 32 cores is ranging between 14% and 15% percent. This is mostly attributed to the $SF_{comp}(comp)$ component of Equation 20 where the projection error for the computation component on BG/P was ranging between 9%-12% for SP-MZ. On the POWER6 575 system, on the other hand, the *WaitTime* projection error ranging between 12%-14% for 16, 32 and 64 cores is due to the change in ratio between computation and communication from the base to the POWER6 575 system (w_{comp} and w_{m_i} components in Equation 20).

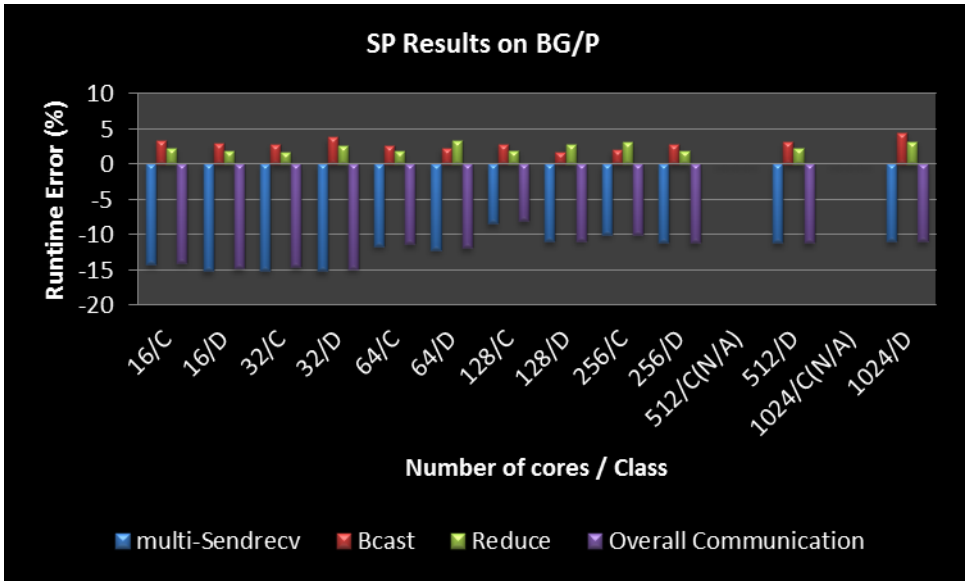

Figure 18: SP results on BG/P

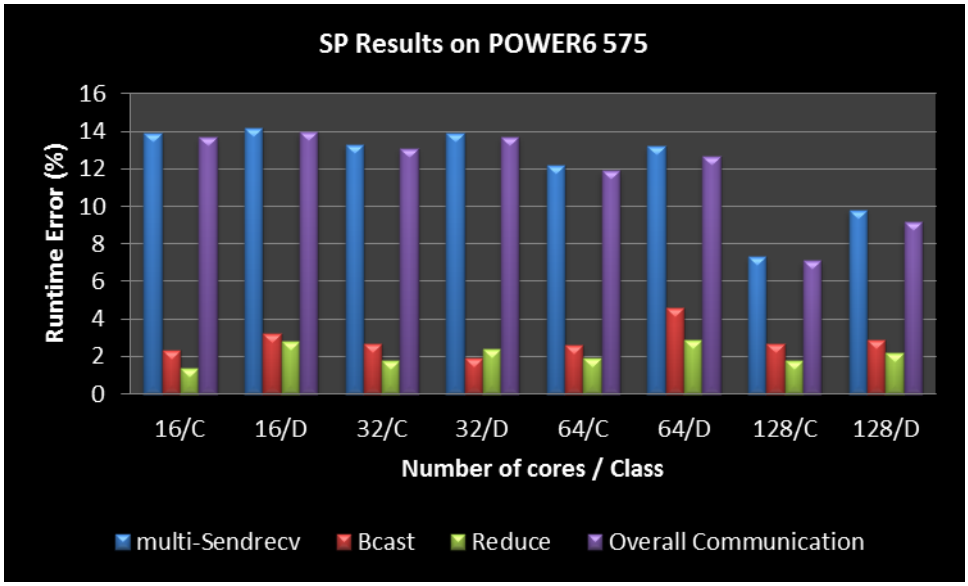

Figure 19: SP results on POWER6 575

Overall, our projection scheme projected with high accuracy using MPI profiles of HPC applications obtained on one base system and IMB benchmark numbers obtained on the target system. From the previous figures one can summarize the results as follows:

1. Projection accuracy for MPI collective routines that have a small *WaitTime* component (almost negligible) is higher than projection accuracy for *multi-Sendrecv* with a large *WaitTime* component.
2. Accuracy of projecting the *multi-Sendrecv* was higher for larger number of tasks. This trend is obvious in SP-MZ and BT-MZ for both classes C and D. This trend is due to the decrease in the *WaitTime* component in *multi-Sendrecv*, i.e. T_{wait} decreases when utilizing larger number of tasks because more time is being spent in the interconnect ($T_{inFlight}$).

Although the average projection error for BlueGene/P is very close to average projection error on IBM POWER6 575, the average error on IBM POWER6 575 was consistently

lower, especially for the *WaitTime* component. This trend is due to the higher similarity between the base system and IBM POWER6 575 interconnect architecture (Federation and Infiniband) and processor architecture (POWER5+ and POWER6), resulting in better computation projection.

4.6 Related Work

There is extensive research dealing with the performance prediction of HPC applications. One approach is to build an analytical model for the application on the target platform using one of the known modeling techniques such as LogGP [27] or LogP [28]. The main advantages of this work over the LogP and LogGP models can be summarized in the following points. First, LogP and LogGP models ignore the network topology and the routing algorithm. In current systems, network topologies have complicated and hierarchical designs, which have significant effect on communication performance. In our scheme, network topology effects are reflected in IMB values. Second, in our prediction methodology, support for collective communication acceleration in the hardware on the target system is captured by IMB; however, LogP and LogGP models assume that a processor will only do Send/Recv. Finally, we model *WaitTime* defined in section 4.3 which is typically due to load imbalance between computation and communication among different tasks. The LogP and LogGP don't model such *WaitTime*.

Another HPC performance modeling approach entails combining a performance profile of an application on a well-known HPC architecture, and the machine characteristics of an emerging architecture to project an application's performance on the

emerging architecture [36], [40], [62]. Error rates were in the range of 0.03% up to 24.70%. Our work primarily differs in the use of MPI profiles to characterize the application communication performance rather than MPI traces. Further, with respect to target machine, we use the machine characteristics obtained from the benchmark data for the projection.

The PHANTOM [36] tool uses deterministic replay techniques to execute any process of a parallel application on a single node of the target system at real speed, hence measuring computation performance. PHANTOM also integrates this replay technique with a trace-driven network simulator, SIM-MPI, to predict communication performance. PHANTOM performance prediction error was 2.22%, 3.95% and 2.29% for BT, LU and SP of the NAS MPI benchmarks respectively. The SIM-MPI simulation overhead was 132%, 420% and 171% of actual execution time for these three benchmarks. In contrast, our MPI profile based methodology used in this work has a maximum overhead of 0.05% of actual execution time.

In [40], Snavely et al introduced a framework for performance modeling and prediction. In the framework, an application signature is created (single processor signature through MetaSim and communication signature through MPIDTrace). Then a machine profile is created (MAPS profile of memory and PMB profile of interconnect). Finally, the machine profile is convoluted with application signature to predict its performance. Projecting the MPI communication performance relies on an MPI trace and the Dimemas simulator [41], [42], [43]. Similar to PHANTOM, network traces and simulation have a significant overhead when compared to our profile based scheme.

Also in [62], Kerbyson et al. introduced the PACE framework where CHIPS application model is convoluted with hardware model of the target system to provide application performance projection. The workload definitions have an Application Layer and Parallel Template Layer. This powerful approach requires significant performance analysis effort.

Clement and Quinn in [31] proposed modeling an application as a function of compiler effects, memory effects, communication overhead and floating point trends. Their work focused on projecting parallel speedup of an application rather than its performance on different systems.

Prakash and Bagrodia in [22] introduced MPI-SIM which simulates the MPI communication library. MPI-SIM uses a detailed contention model. Also Wilmarth et al. in [23] introduced POSE which uses a detailed network contention model to simulate the communication library. Simulators although highly accurate, they are slow and time consuming.

4.7 Summary

We presented a method to project the performance of the MPI communication component of HPC applications using MPI profiles obtained on one base machine. The use of MPI profiles instead of MPI traces allows for efficient and fast projection. The use of MPI traces may yield more accurate results; however, projecting using MPI traces can be impossible in many cases for large scale production applications due to the huge sizes of the traces and the long time it takes to simulate these traces for target systems.

The *MM* model for an HPC application *A* assumes a constant dataset size *D* and a constant algorithm for all cores C_j. The *MM* model identifies how each MPI routine scales with different number of cores C_j. If the dataset size *D* changes or the algorithm is altered, a new *MM* model is required.

In our scheme, we model the time a task spends waiting on other tasks to finish as *WaitTime*. This is specifically important for MPI asynchronous calls in HPC applications that exhibit load imbalance between tasks. We calculate the scaling factor for the *WaitTime* from the base to the target system using the scaling of performance for individual MPI routines as well as the scaling in computation performance. The ratio of computation to communication on the base system, which may differ than the ratio on the target, as well as the computation scaling factor are the three parameters that affect the accuracy of projecting *WaitTime*; thus, the accuracy of projecting the performance of MPI routines with little *WaitTime* component, synchronous routines, is higher than projecting MPI routines with higher *WaitTime* since they don't depend on these three parameters.

5. COMBINED COMPUTATION AND COMMUNICATION PERFORMANCE PROJECTION

In section 3, we presented our method to project the performance of only the computation component of an HPC application using published data of industry standard benchmarks, the SPEC CPU2006, and hardware performance counter data from one base system. In section 4, we extended our method to project the communication performance of HPC applications onto different systems using MPI benchmark data on the different systems as well as a base system, and the communication profile of the application on the base system. In this section, we present a scheme to combine the communication projection with the computation projection to project the entire HPC application performance on a target system. In particular, we identify the communication strong scaling factor of the HPC application as well as the computation strong scaling factor. We then apply the scaling factor to the projected performance of each component and combine both scaled components.

Figure 20 depicts the high level framework of our scheme for combining the communication and computation projections. The HPC application is executed multiple times on the base machine where each execution utilizes different number of cores C_j for $j \in \{1,..,c\}$ and c is the maximum number of cores the HPC application can utilize. During each execution, we obtain the application's MPI profile for C_j number of cores. Note that this step is already completed in the MPI communication projection scheme and need not be repeated here. The resultant MPI profiles are used to produce the Application MPI Model *MM* defined in section 4.3.1 and Equation 16. These MPI profiles are also used to

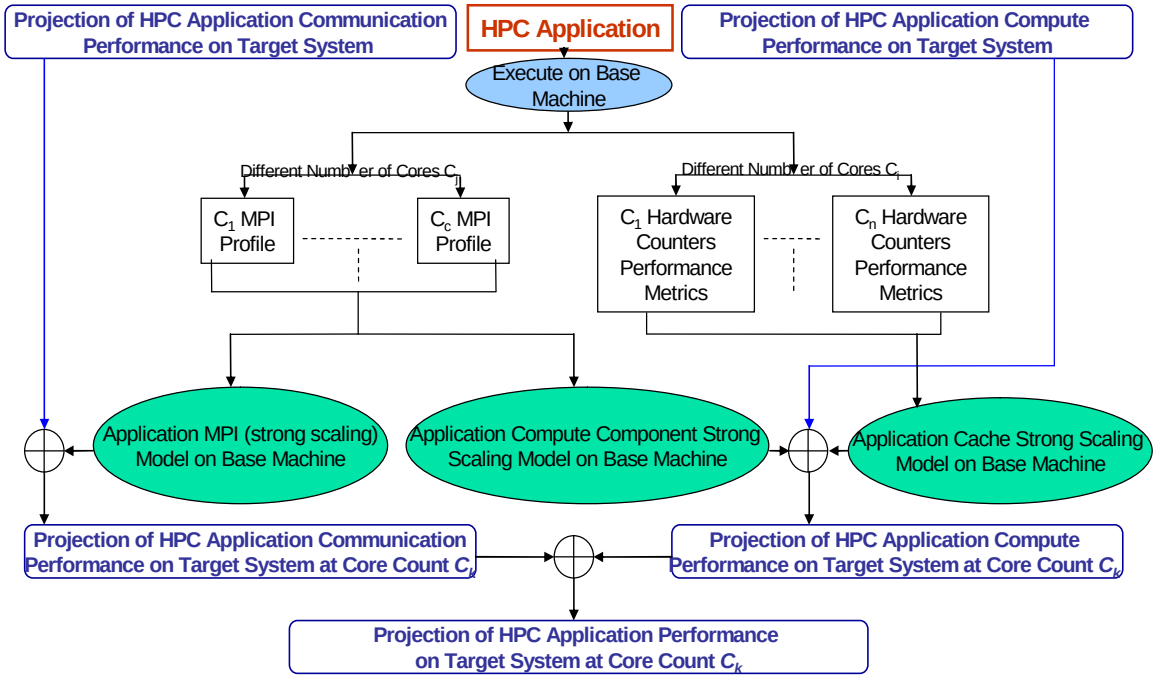

Figure 20: Framework for combining computation and communication projection

produce the Application **C**ompute **C**omponent Strong **S**caling **M**odel (*CCSM*) on the base machine. *CCSM* identifies how the compute component scales with increasing number of cores. Furthermore, we collect hardware performance counter metrics for the HPC application at different processor counts C_i for $i \in \{1,...,n\}$ and $n \leq c$. Using hardware performance counter metrics, we develop the Application **C**ache Strong **S**caling **M**odel (*CSM*) on the base machine. The *CSM* model allows for identifying the number of cores at which the application cache footprint may be contained in a lower level cache. Once these models are developed, projecting the entire application performance is achieved in three steps. One, the *CSM* model and the *CCSM* model are combined with the compute component projection to produce the compute component performance projection at the required number of cores C_k. Two, the *MM* model is combined with the MPI communication projection to produce the communication component performance

projection at the required number of cores C_k. Finally, we add the projected performance for the compute component at core count C_k to the projected performance of the communication component at core count C_k to produce the entire application projected performance at core count C_k where C_k is the required core count on the target system.

5.1 Cache Scaling Model

Cache Strong Scaling Model (*CSM*) allows one to identify the number of cores at which the application cache footprint may be contained in a lower level cache. For example, an HPC application utilizing four cores could be using the L3, L2 and L1 caches; however, the same application when utilizing 1000+ cores, only L2 and L1 may be needed. In such case, hyper scaling in performance may occur when using more than 1000 cores. Since there is a significant difference between latency of L3 and L2, the application performance benefits from never having to access the L3 cache and hyper scaling occurs.

Using *CSM* model, one can identify the number of cores at which the application performance exhibits hyper scaling. In our projection scheme, once the *CSM* model identifies the number of cores C_h at which the application experience hyper scaling, a new computation projection is required at C_h. This implies that the compute projection scheme explained in section 3 will be repeated for the HPC application at C_h where hardware performance counter metrics collected at C_h will reflect the new cache footprint and its effect on application performance on the processing core. The need to repeat the compute performance projection at C_h is due to the fact that the change in the cache

footprint affects several hardware performance metrics such as memory bandwidth, CPI stack breakdown and data from different memory levels.

Calculating C_h at which the application experiences hyper scaling due to significant change in cache footprint follows directly from the values of metrics $m_{5,1}$, $m_{5,2}$, $m_{5,3}$ and $m_{5,4}$ in metric group G_5 in Table 4 for different processor counts C_i for $i \in \{1,...,n\}$ and $n \leq c$. Typically, $n = 4$ suffices to calculate C_h for an application by extrapolating on the values of the metrics to identify C_h. For example, using $m_{5,2}$ (DATA_FROM_L3) one can identify the C_h where all the data will be contained in L2 for an HPC application by calculating C_h where $m_{5,2}$ value will be 0. This is done by extrapolating on the decreasing values of $m_{5,2}$ when increasing number of cores.

5.2 Compute Component Strong Scaling Model

Compute Component Strong Scaling Model allows one to calculate the scaling factor γ for the compute component of the HPC application. The MPI profiles for the HPC application at different task counts C_j for $j \in \{1,..,c\}$ contains information about the computation elapsed time at C_j. Using curve fitting techniques, the scaling factor γ can be directly calculated.

5.3 Combined Communication and Computation Performance Projection Scheme

As indicated in Figure 20, the process of combining the communication projection with the computation projection to produce the entire HPC application projections entails three steps:

1. Combining the projected performance of the MPI communication component of the HPC application with the MPI communication model *MM*. Recall that the *MM* model identifies how each MPI routine scales with different number of cores C_j. For each MPI routine, the message size and number of calls change with changing the number of cores. Also recall that the MPI communication model *MM* is for an HPC application *A* with a specific problem size *D*. The problem size *D* indicates that the dataset size and the algorithm used by an application *A* are constant for all C_j. By combining the projected performance of the MPI communication component of the HPC application with the *MM* model, one can produce the projected performance of the MPI communication for the required task count C_k. As indicated previously, this step follows directly from section 4.4.3. Thus, we can define the projected performance of the MPI communication at the required task count C_k using Equation 18 as follows:

$$T_{Elapsed_{target}}(m_i)_{C_k} = T_{Transfer_{target}}(m_i)_{C_k} + T_{Wait_{target}}(m_i)_{C_k}, \forall i \in \overline{M} \quad (22)$$

2. Combining the projected performance of the compute component of the HPC application with the *CCSM* and the *CSM* models. Recall that the *CCSM* model identifies how the compute component scales with increasing number of cores. By combining the *CCSM* model with the projected performance of the compute component as in Equation 10, one can define the projected compute component performance at required task count C_k as follows:

$$P_{app_{C_k}} = \gamma \sum_{k=1}^{|S_{app}|} (w_k P_k)_{C_k} \quad (23)$$

where γ is the scaling factor for the compute component identified by the *CCSM* model as indicated previously. The *CSM* model is used in this step to identify the point where γ will not be applicable as hyper scaling of the application may occur due to significant changes in cache footprint.

3. In the final step of the projection process, we add the projected performance of the two components of the HPC application. The result of this addition is the projected performance of the entire HPC application.

5.4 Experimental Results

We used the methodology depicted in Figure 20 to project the performance of some of the workloads that were used in validating the computation component as well as the three NAS Multi-Zone benchmarks BT-MZ, LU-MZ and SP-MZ. The HPC applications used for validating the combining scheme are AMBER (GB_COX, JAC, Factor_IX), GAMESS (SICCC, L-Rotenon) and WRF (CONUS). The remaining workloads from the computation validation, Fluent, StarCD, CHARMM, Siesmic and LS-Dyna were not used due to licensing issues. Please refer to Tables 3 and 8 for details of the applications. We projected the performance onto three target systems, an IBM internal POWER6 575 cluster system, an IBM internal Intel Westmere (Xeon X5670) cluster system and an IBM internal BlueGene/P system. The choice of the three target systems allows for validating on different processor architectures and different interconnect architecture. Details of the three target systems and the base system are provided below in Table 10. Throughout our validation process, the IMB benchmarks and

NAS-MZ benchmarks are executed using the same MPI library on a system. Also they both follow the same task placement strategy for consistency. On the BlueGene/P system, our experiments were all done using the "Virtual Node" mode where four MPI tasks are utilizing the four cores per node. On the TAMU Hydra and the IBM POWER6 575 systems, the Single Thread (ST) mode was utilized.

Table 10: Base system and different systems used for validation

Machine	Processor	Total Cores	Cores Per Node	Memory Per Core	Interconnect
TAMU Hydra	POWER5+	832	16	2GB	Federation
IBM POWER6 575 cluster	POWER6	128	32	4GB	InfiniBand
BG/P	PowerPC 450	4096	4	1GB	3D Torus/ Collective Tree
IBM X5670	Intel Xeon X5670	768	12	2GB	Idataplex

Table 11 shows the characteristics of the HPC workloads used in the validation process as well as the characteristics of the NAS Benchmarks. In all the following results graphs we divided the communication as Point-to-Point Blocking (P2P-B), Point-to-Point Non-Blocking (P2P-NB) and Collectives. The details of these calls and their types are explained in more details in Table 8 for the applications on the base system at 128 tasks for elaboration.

Table 11: HPC workloads characteristics on base system for 128 tasks

HPC Workload	Comm. Percent	Collective Percent	Dominating Collective	P2P-NB Percent	Dominating P2P-NB
BT-MZ-C	59.7%	0.59%	MPI_Bcast and MPI_Reduce	59.1%	*multi-Sendrecv*
LU-MZ-C	1.4%	0.014%	MPI_Bcast and MPI_Reduce	1.38%	*multi-Sendrecv*
SP-MZ-C	16%	0.016%	MPI_Bcast and MPI_Reduce	15.84%	*multi-Sendrecv*
BT-MZ-D	6.8%	0.068%	MPI_Bcast and MPI_Reduce	6.7%	*multi-Sendrecv*
LU-MZ-D	1.2	0.012%	MPI_Bcast and MPI_Reduce	1.18%	*multi-Sendrecv*
SP-MZ-D	6.6%	0.066%	MPI_Bcast and MPI_Reduce	6.5%	*multi-Sendrecv*
AMBER-GB_COX2	67.2%	67.1%	MPI_Reduce_scatter and MPI_Bcast	0%	N/A
AMBER-Factor_IX	53.7%	41.87%	MPI_Bcasr and MPI_Allreduce	11.28%	*multi-Sendrecv*
AMBER-JAC	41.4%	34.28%	MPI_Bcast and MPI_Allreduce	7.1%	*multi-Sendrecv*

Table 11: continued

HPC Workload	Comm. Percent	Collective Percent	Dominating Collective	P2P-NB Percent	Dominating P2P-NB
GAMESS-SICCC	5.78%	5.02%	MPI_Bcast and MPI_Allreduce	0.69%	*multi-Sendrecv*
GAMESS-LROT	3.08%	2.99%	MPI_Bcast and MPI_Allreduce	0.09%	*multi-Sendrecv*
WRF	18.7%	13.2%	MPI_Bcast	4.37%	*Multi-Sendrecv*

Figures 21-23 show the results for the BT-MZ benchmark on the three target systems. Overall, the average errors were 10.53%, 9.32% and 13.61% on the BG/P, POWER6 575 cluster and the Intel Westmere cluster systems respectively. The maximum error didn't exceed the 15% on any of the systems. An apparent trend on all systems is that the computation projections for the Class D workloads were more accurate, i.e. less projection error. This trend is due to the fact that the Class D has a longer execution time than Class C allowing for collecting more accurate hardware performance counters data. This trend continues in the SP-MZ workloads as well. Another point worth mentioning is that for the BT-MZ, SP-MZ and LU-MZ benchmarks, the computation projection accuracy determines the entire projection accuracy even in the cases where communication is the dominating component. Recall that *WaitTime* projection in the communication component, which is the dominating factor in communication for the three NAS benchmarks highly depends on the computation projection.

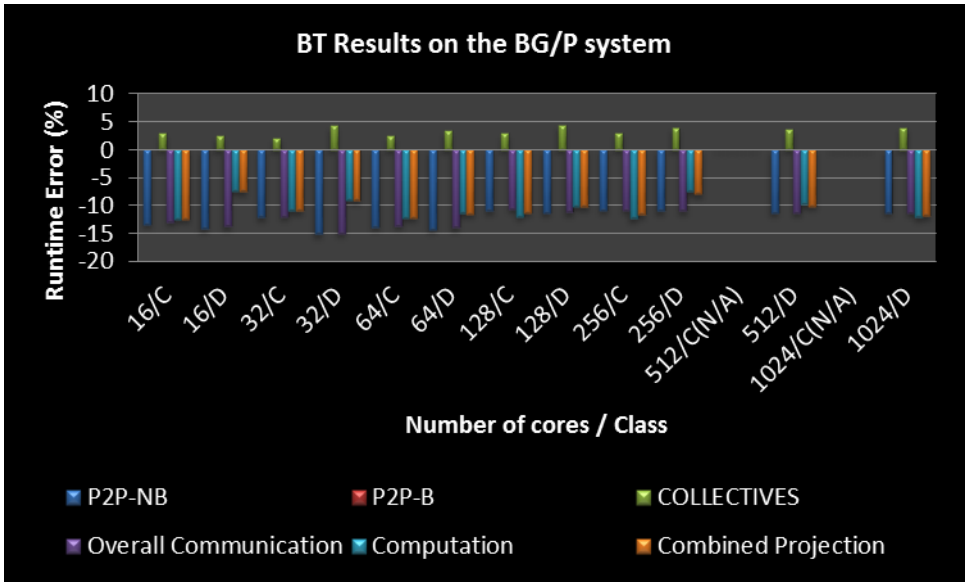

Figure 21: BT results on the BG/P system

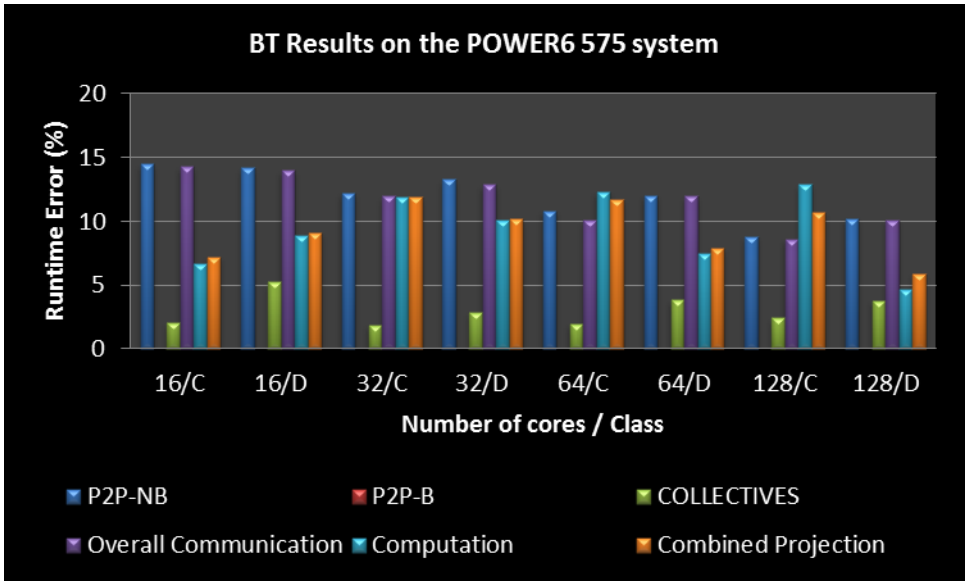

Figure 22: BT results on the POWER6 575 system

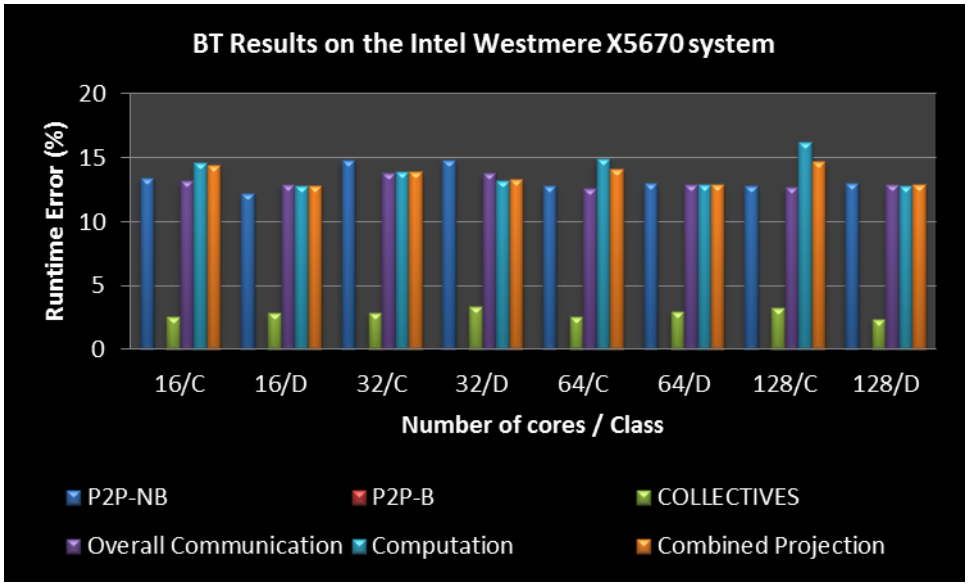

Figure 23: BT results on the Intel Westmere X5670 system

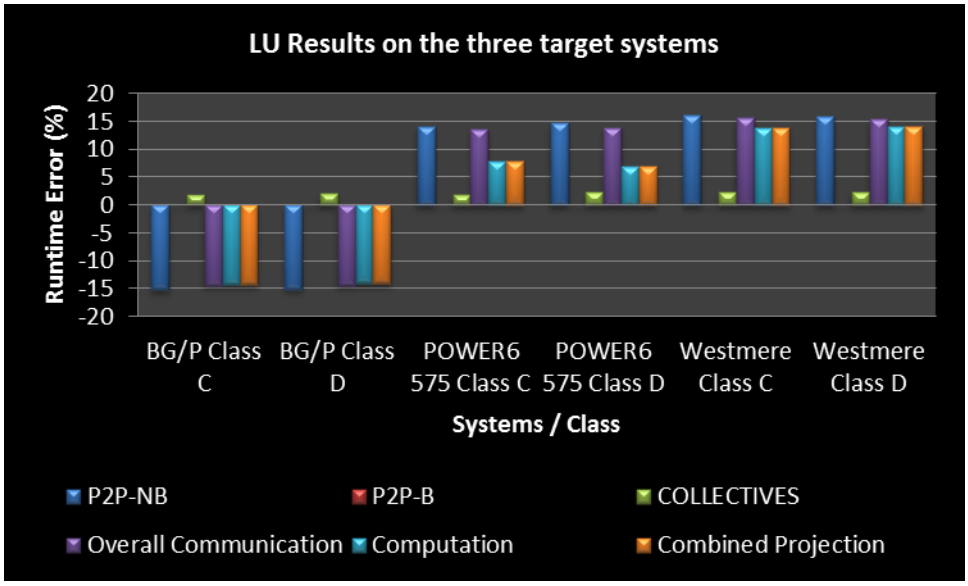

Figure 24: LU results on the three target systems

Figure 24 shows the LU-MZ benchmark results of the projection methodology on the three target systems. As indicated earlier from the BT results, Class D has better computation projection results than Class C which drives the entire projection error down.

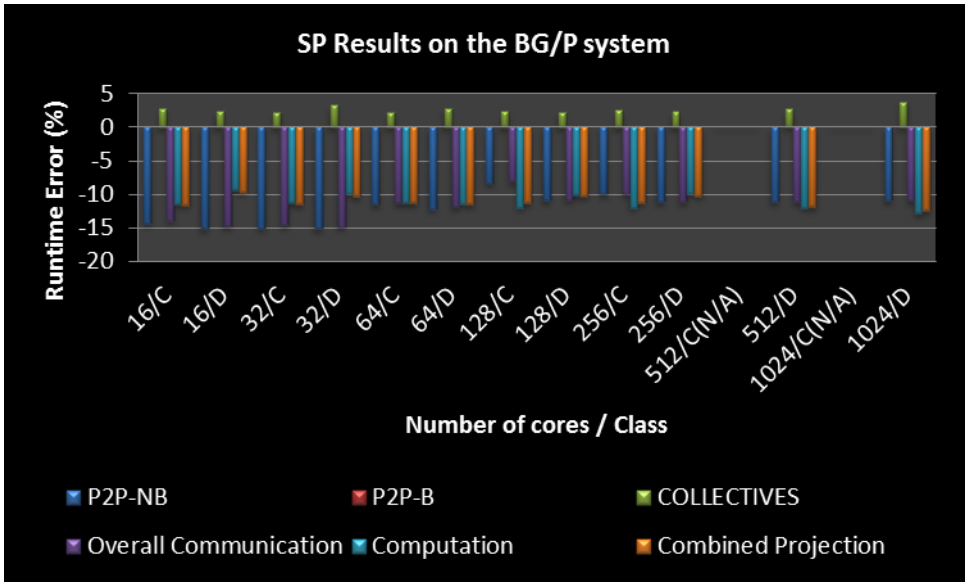

Figure 25: SP results on the BG/P system

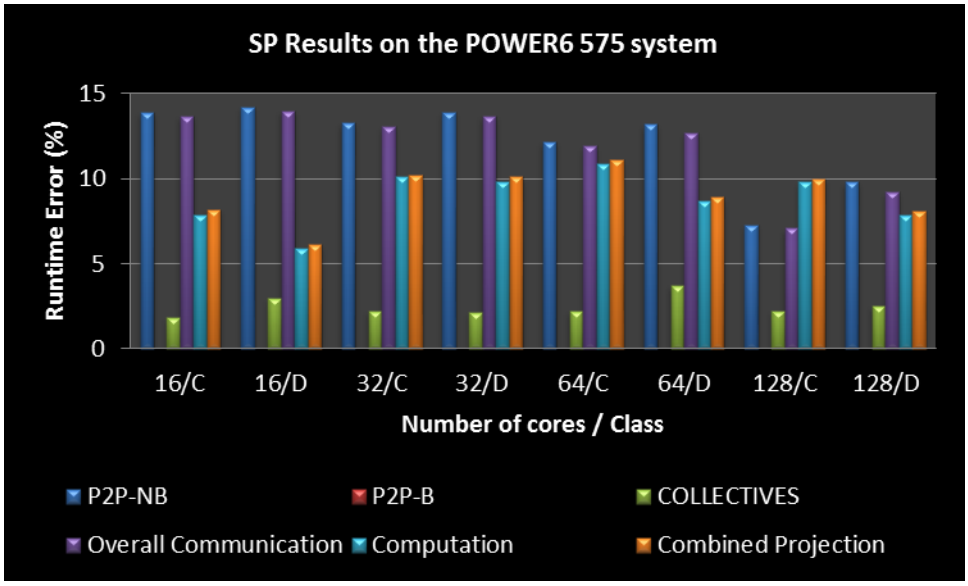

Figure 26: SP results on the POWER6 575 cluster system

Figures 25-27 show the results for the SP-MZ benchmark on the three target systems. The average projection errors were 11.06%, 9.08% and 13.54% on BG/P, POWER6 575 and Intel Westmere systems respectively. As indicated earlier in the BT-MZ case, computation projection is typically more accurate for the Class D case. Also,

the POWER6 575 projections were more accurate than the BG/P and the Westmere system where the Westmere was typically the system with less accurate projections. The fact that the POWER6 system uses a POWER ISA allows for better matching of performance metrics resulting in a set of surrogates with closer behavior to the HPC workload.

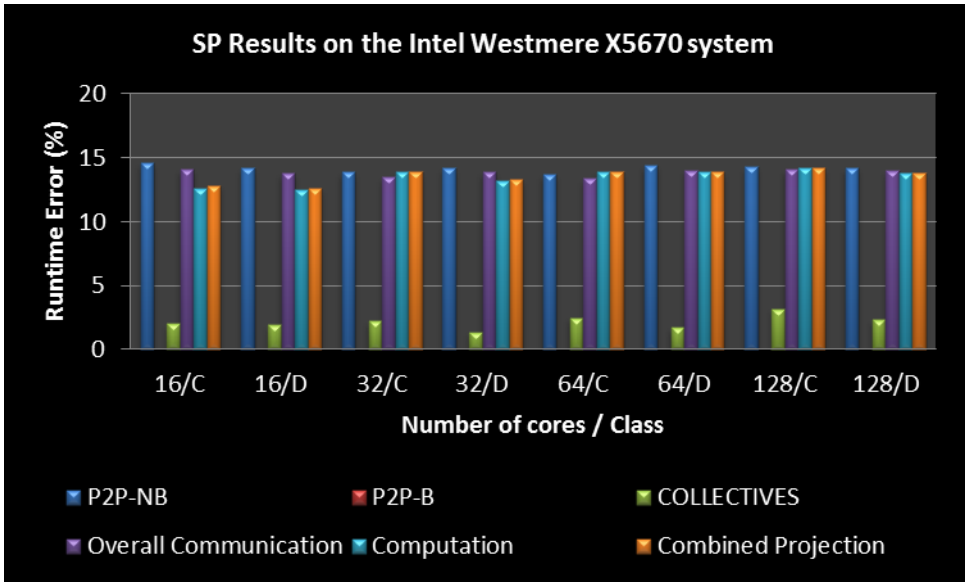

Figure 27: SP results on the Intel Westmere X5670 system

Figures 28-30 show the results for the AMBER Gb_COX2 workload. The average errors were 12.17%, 11.21% and 13.55% for the BG/P, POWER6 575 and Intel Westmere systems respectively. In the case of the BG/P system, it is noticeable that the computation projection accuracy decreases with increasing number of cores, computation projection is about 17% for the 1024 tasks. This behavior is due to the fact that the AMBER Gb_COX2 workload has a small dataset that doesn't scale very well and the scaling factor γ used in Equation 23 doesn't reflect the exact scaling of the workload for larger number of tasks. Nevertheless, the overall projection error is still below 10%, 8.9%

to be exact for the 1024 tasks. In the case of 1024 tasks, the communication component is the dominating factor in performance. Since the dominating MPI routines in the communication are the collectives calls as indicated in Table 11, and collectives have almost no *WaitTime* component, i.e. independent of computation, the communication projections have high accuracy. Thus, the overall application projection accuracy is less than 10%.

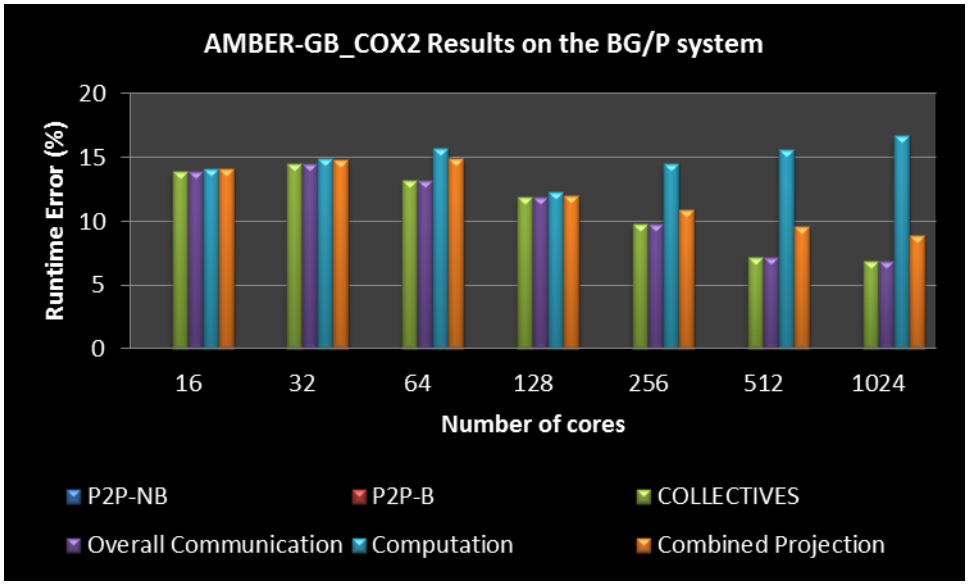

Figure 28: AMBER-GB_COX2 results on the BG/P system

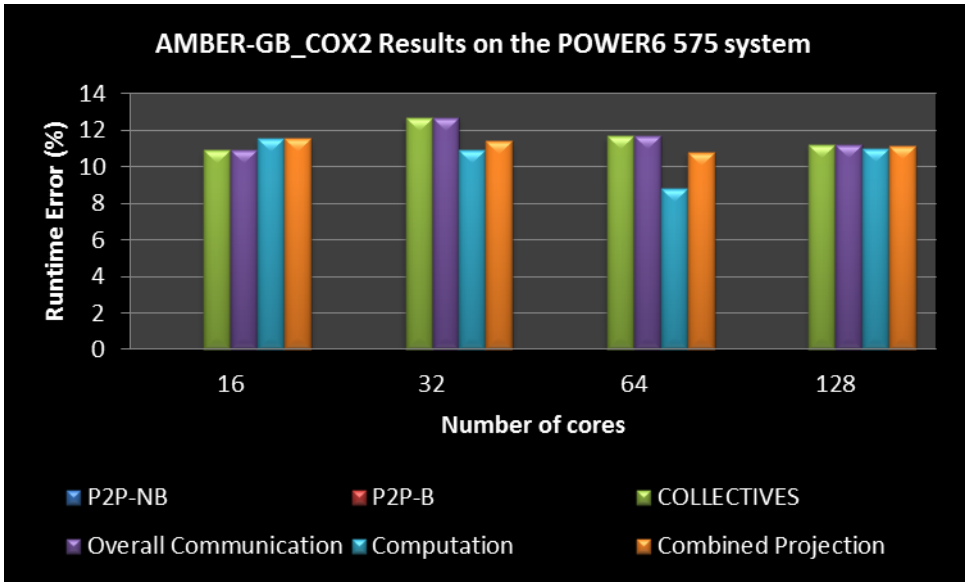

Figure 29: AMBER-GB_COX2 results on the POWER6 575 system

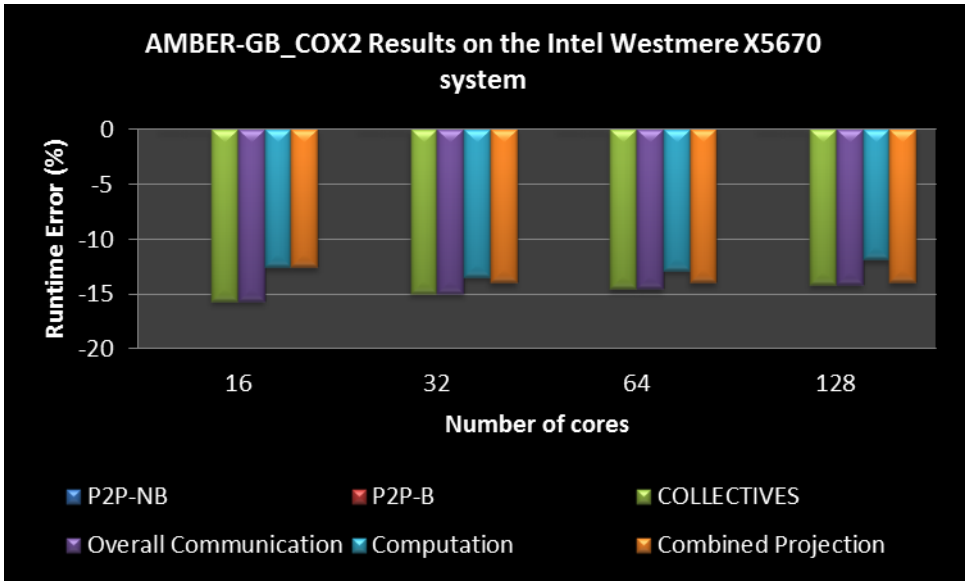

Figure 30: AMBER-GB_COX2 results on the Intel Westmere X5670 system

Figures 31-36 show the results for the AMBER-Factor_IX and AMBER_JAC respectively. Both workloads exhibit similar behavior and their projection results follow the same trends. The average errors for the Factor_IX workload are 10.42%, 7.07% and 10.15% for the BG/P, POWER6 575 and Intel Westmere systems respectively. As for the

JAC workload, the average errors are 11.78%, 9.45% and 12.65% for the BG/P, POWER6 575 and Intel Westmere systems respectively. For both workloads, the communication component performance is dominated by the collective routines MPI_Bcast and MPI_Allreduce. Since collective calls projection is independent from the projection of computation, the low computation projection error is worsened by the higher projection error of the communication; however, the overall error is still much lower than 15%. In these two workloads, we notice the same trend of higher computation projection error for the larger number of tasks on BG/P as in the Gb_COX2 workload.

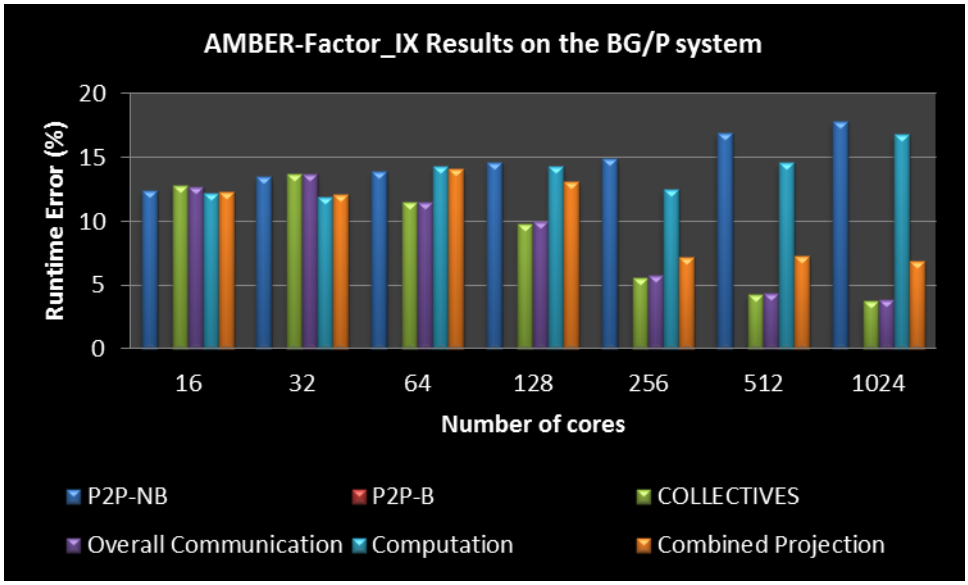

Figure 31: AMBER-Factor_IX results on the BG/P system

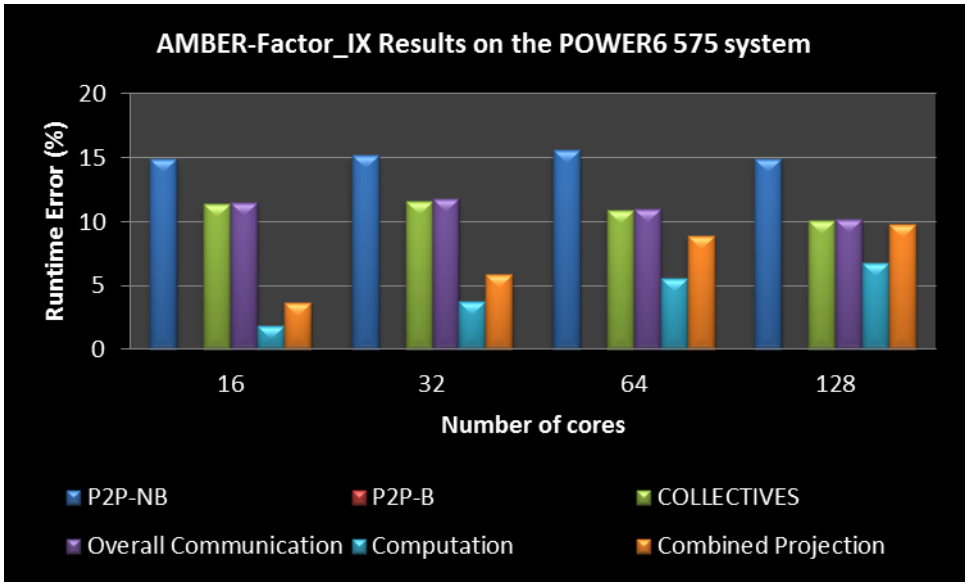

Figure 32: AMBER-Factor_IX results on the POWER6 575 system

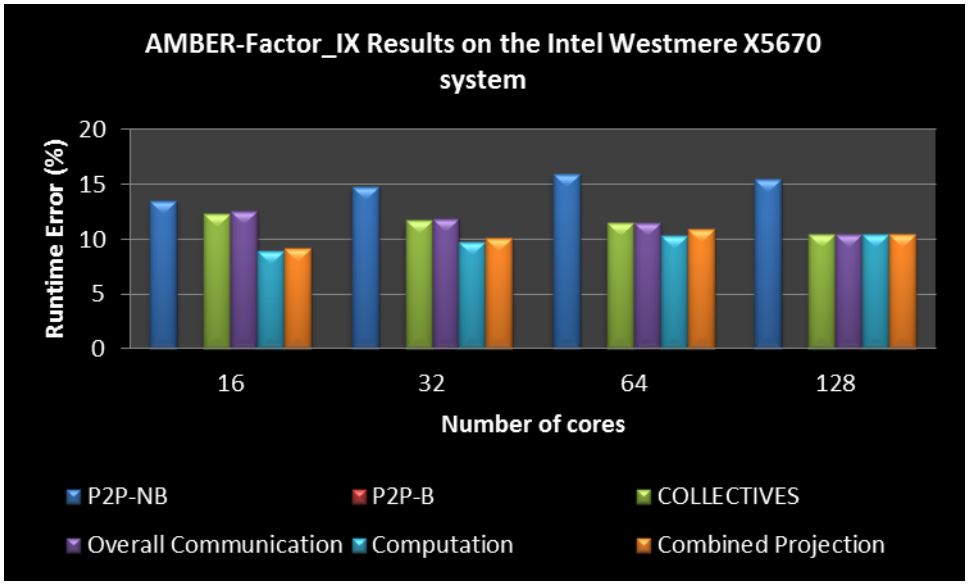

Figure 33: AMBER-Factor_IX results on the Intel Westmere X5670 system

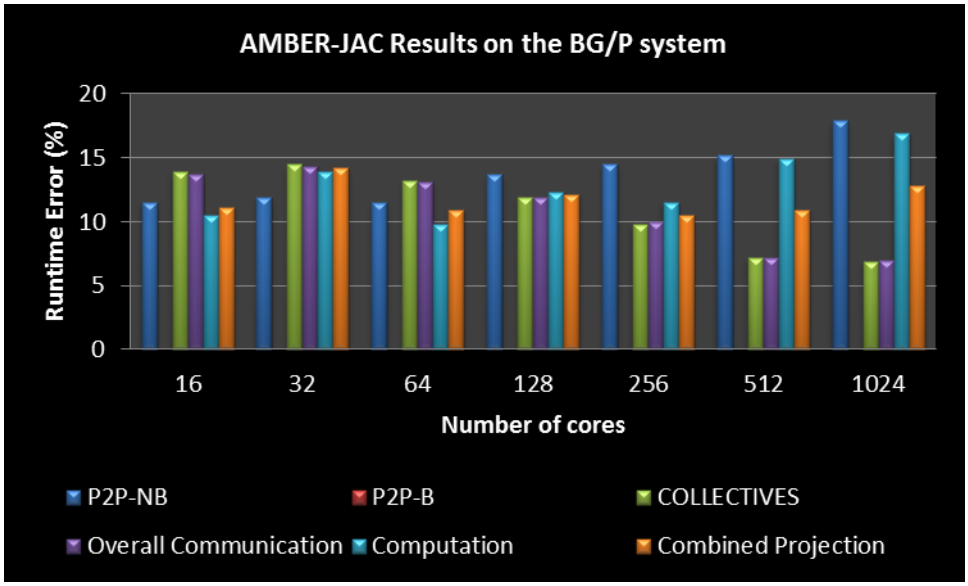

Figure 34: AMBER-JAC results on the BG/P system

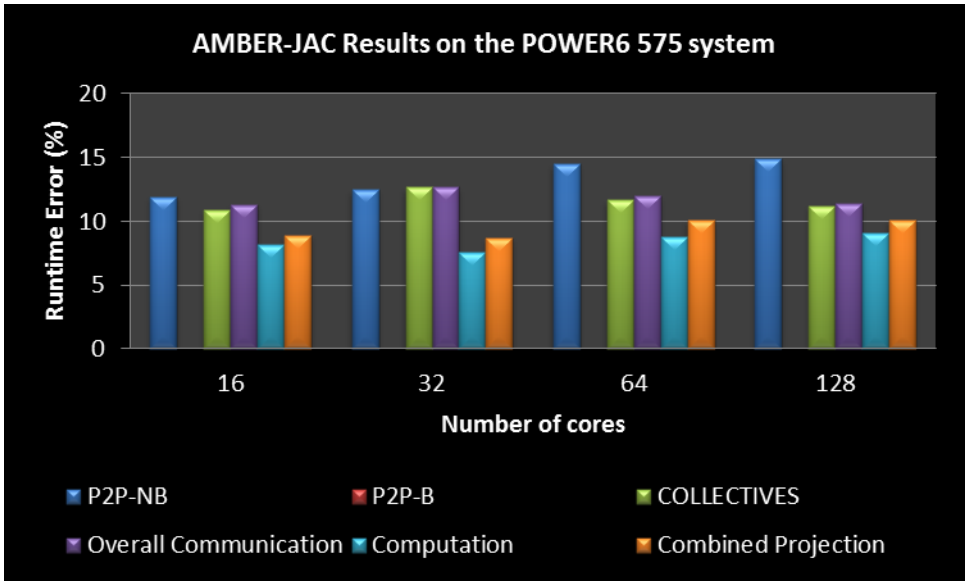

Figure 35: AMBER-JAC results on the POWER6 575 system

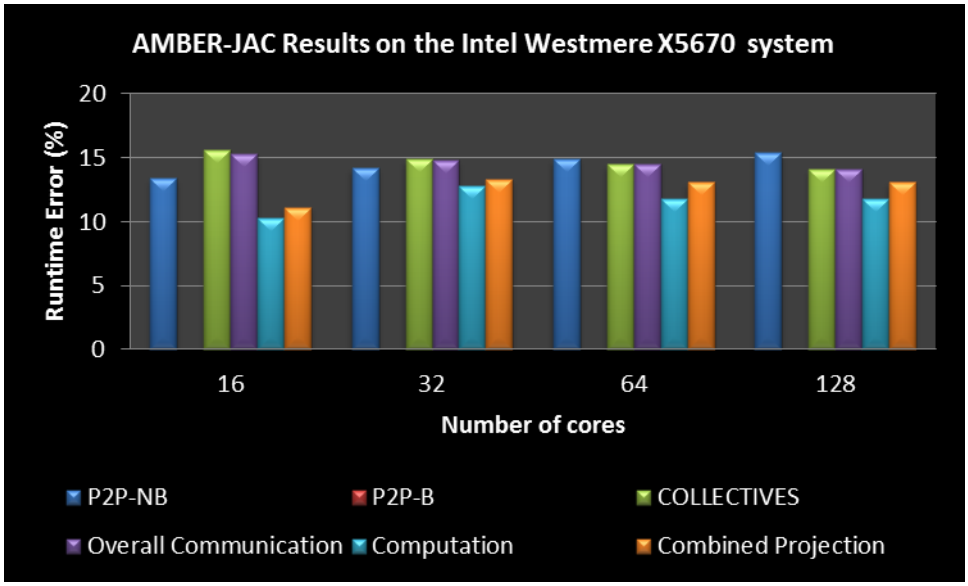

Figure 36: AMBER-JAC results on the Intel Westmere X5670 system

Figures 37-42 show the results for the two GAMESS workloads. The average projection error for both workloads are 12.11%, 8.74% and 13.9% for the BG/P, POWER6 575 and the Intel Westmere systems. As it is clear from the averages, the POWER6 system has the lowest errors. The errors on POWER6 are lower since the computation component is the more significant factor in the performance and POWER6 has the most accurate computation projection. As it is clear from the results, the collective and P2P-B communication projection results have minor effect on the overall projection accuracy due to their minor significance in the overall performance of the applications.

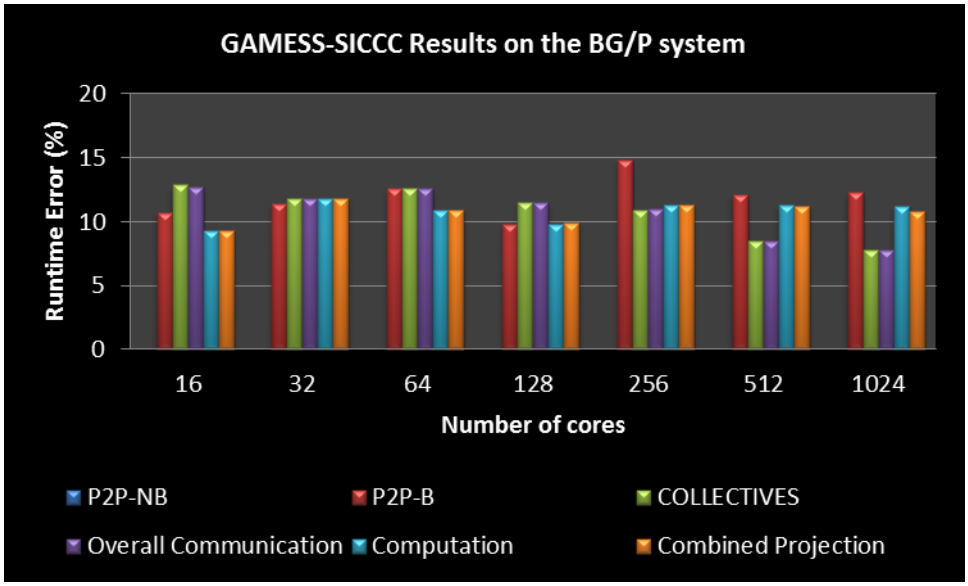

Figure 37: GAMESS-SICCC results on the BG/P system

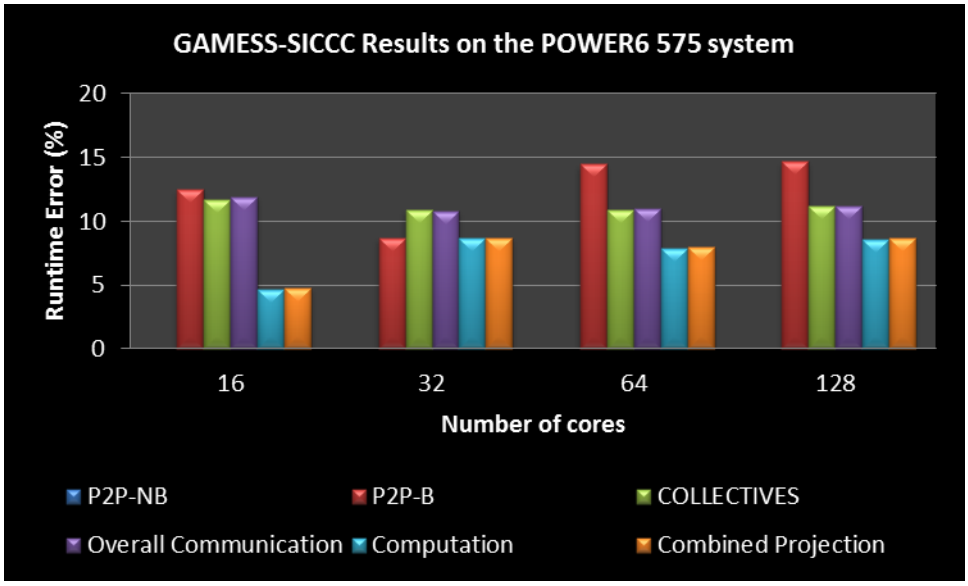

Figure 38: GAMESS-SICCC results on the POWER6 575 system

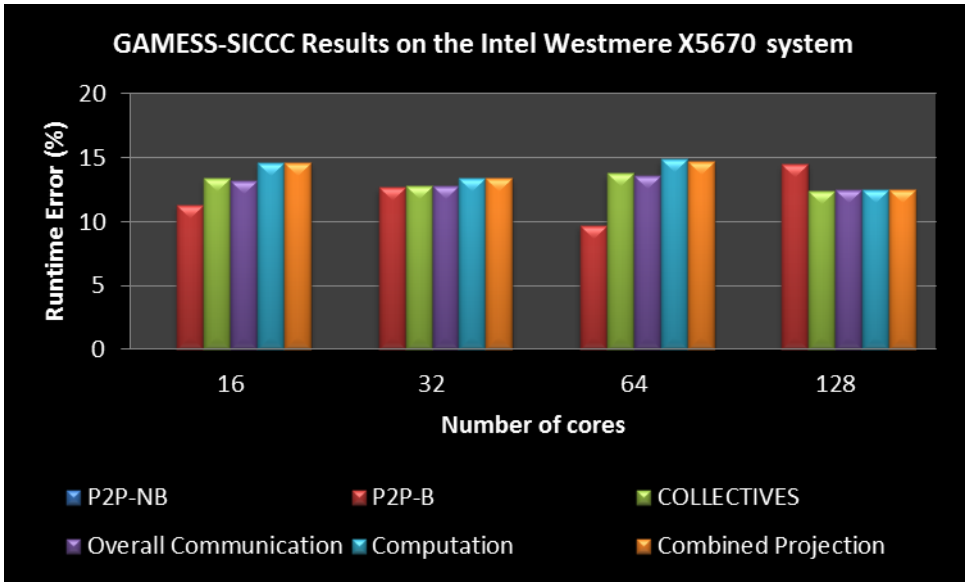

Figure 39: GAMESS-SICCC results on the Intel Westmere X5670 system

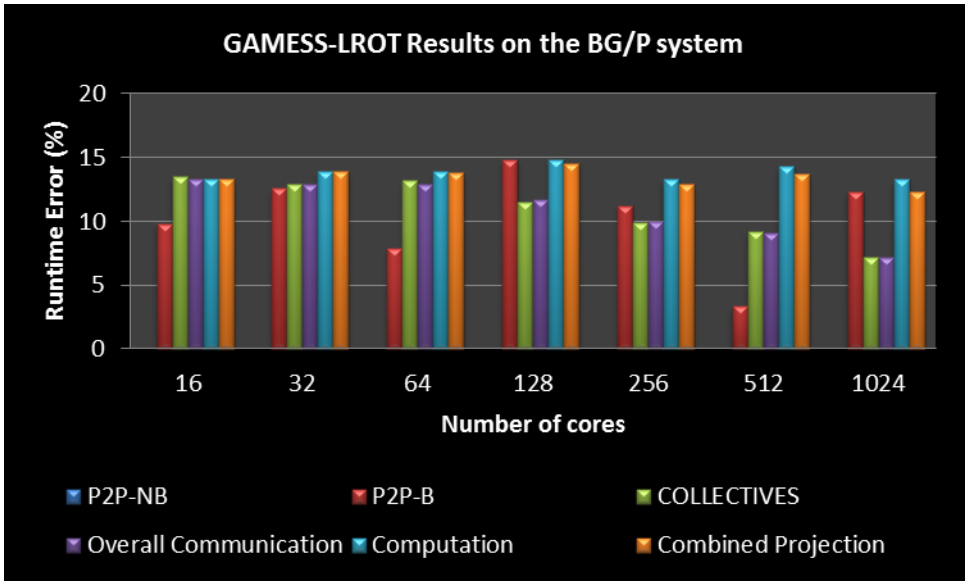

Figure 40: GAMESS-LROT results on the BG/P system

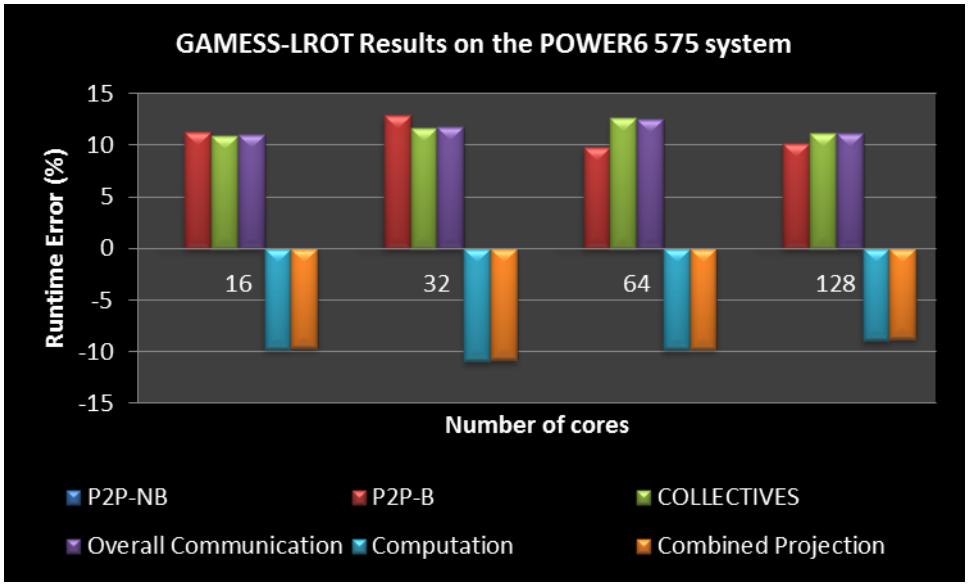

Figure 41: GAMESS-LROT results on the POWER6 575 system

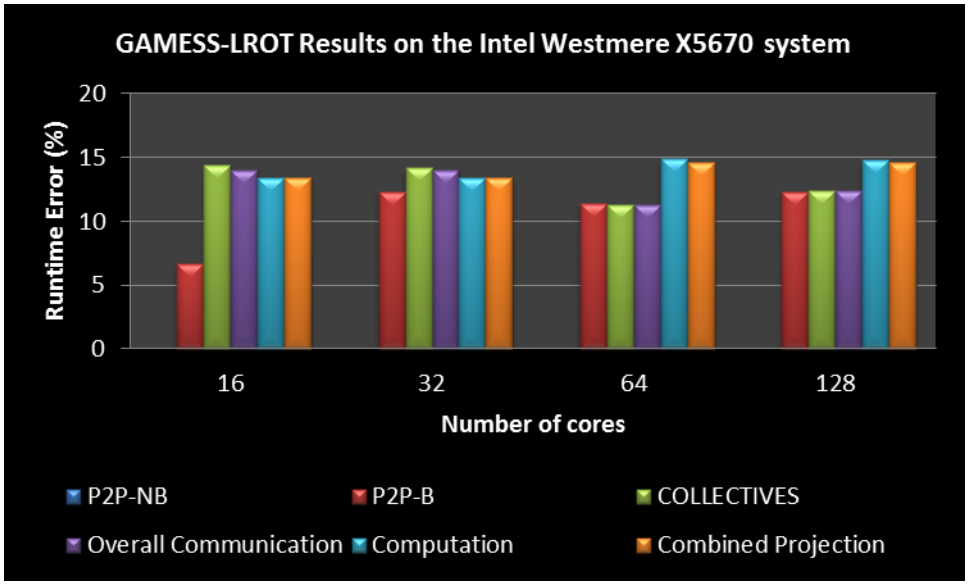

Figure 42: GAMESS-LROT results on the Intel Westmere X5670 system

Figures 43-45 show the results for the WRF application. The average projection errors are 10.67%, 10.12% and 13.45% for the BG/P, POWER6 575 and the Intel Westmere systems respectively. There is a major point to notice in WRF projection data. The scaling factor γ on the BG/P machine is more accurate in the WRF case than in any

other workload. This is due to the fact that the CON-US dataset scales well on larger number of tasks. The better scaling for WRF also contributed to better projections results on the three systems with very small standard deviation.

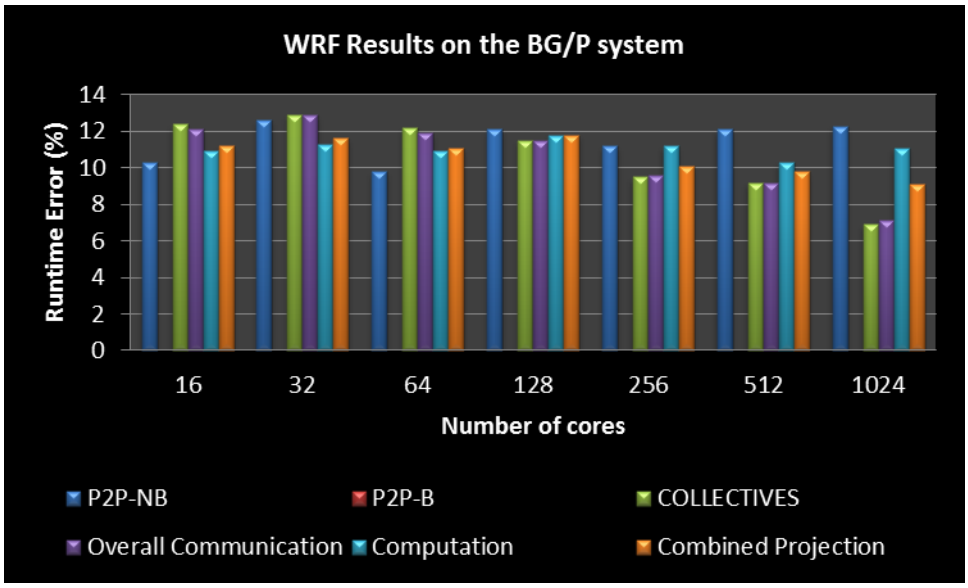

Figure 43: WRF results on the BG/P system

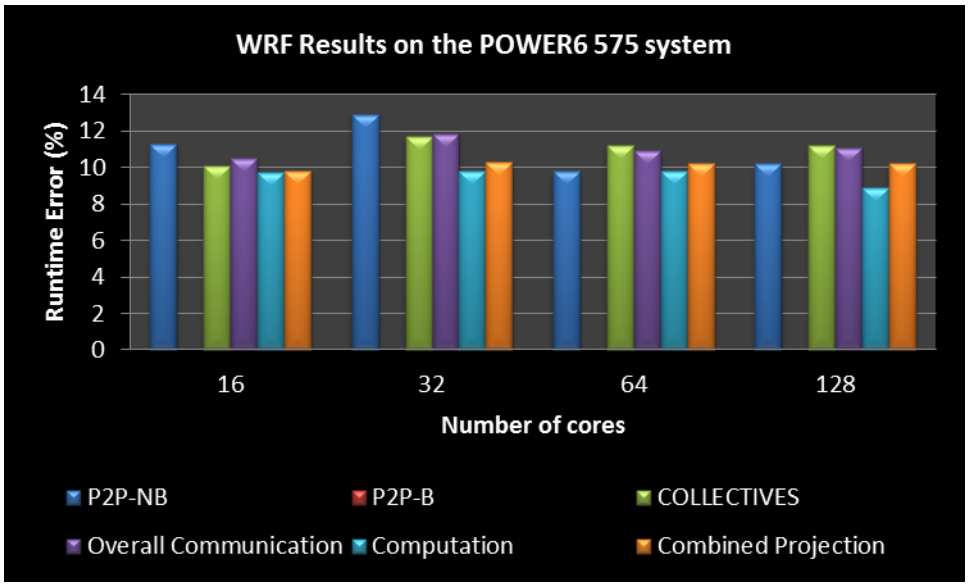

Figure 44: WRF results on the POWER6 575 system

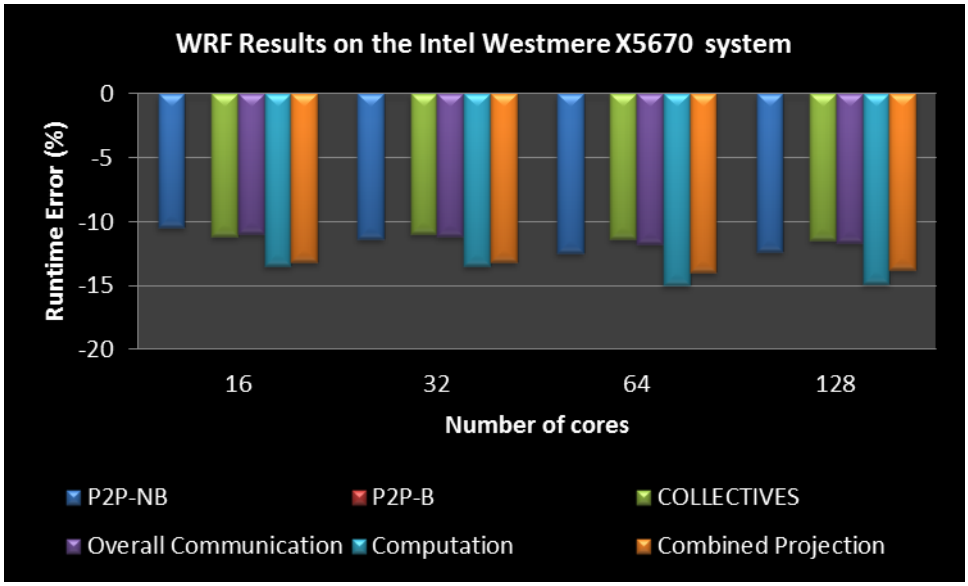

Figure 45: WRF results on the Intel Westmere X5670 system

Overall, our projection methodology projected the performance of the HPC workloads with high accuracy and efficiency. The average projected error on the BG/P system was 11.36% with standard deviation of 1.51%. Also the average projected error on the POWER6 575 cluster system was 9.21% with standard deviation of 1.31%. Since the POWER6 system has a similar ISA to the POWER5+ base system, all the workloads, with the computation component as the dominating factor in performance, have less projection error than on other systems. Finally, the average projected error on the Intel Westmere X5670 is 13.09% with standard deviation 0.74%. Since the Intel Westmere has the most different ISA and micro-architecture from the base processor, the projections for the compute component and the communication component that has a significant *WaitTime* have a higher error than other systems.

In the cases where the γ scaling factor was not accurate, the computation projection at a higher task count was not accurate; however, this happened in the case where the dataset size was relatively small and the workload didn't scale very well with

increasing number of tasks. On the other hand, when the application scales well with the increasing number of tasks, the γ scaling factor was accurate. Also, in the cases where the application has a very short execution time, the performance counter data may be less accurate. In such cases, we recommend that one collects the hardware counter data one group at a time, i.e. no performance counter groups multiplexing.

5.5 Related Work

Current state-of-art HPC performance modeling techniques primarily rely on combining a performance profile of an application on a well-known HPC architecture, and the machine characteristics of an emerging architecture to project an application's performance on the emerging architecture [63]. Existing profiling and tracing tools on well-known architectures are typically used to collect the necessary performance data by executing applications and benchmarks on available systems.

In our approach of surrogate based performance projection an HPC application is modeled as a combination of benchmarks. These benchmarks may be executed on the target system if the system exists or simulated if the system is still in the design phase. This technique provides for the accuracy of cycle-accurate simulation and the speed and ease of performance modeling. In this work, we model the compute component separately from the communication component. The compute component is modeled after the SPEC CPU benchmarks while the communication is modeled after the IMB benchmarks for MPI communication.

Several researches have been done on using surrogate workloads to predict application performance. SPEC benchmarks suite was often proposed as the benchmarks

of choice due to the abundance of published data but it was not used for HPC applications. NAS Parallel [59] benchmarks, on the other hand, were used more often with HPC applications due to their parallel nature. Also curve fitting on runtimes is extensively used in industry to project performance using surrogate workloads.

An approach to similar to the approach introduced in this work is the PHANTOM tool [36]. The PHANTOM tool uses deterministic replay techniques to execute any process of a parallel application on a single node of the target system at real speed, hence measuring computation performance. This assumes that a single node of the target system is available which may not always be the case. PHANTOM also integrates this replay technique with a trace-driven network simulator, SIM-MPI, to predict communication performance. Thus, PHANTOM simulates only the communication component while replacing computation blocks with their actual execution time to speed up simulation time. PHANTOM performance prediction error was 2.22%, 3.95% and 2.29% for BT, LU and SP of the NAS MPI benchmarks respectively. The SIM-MPI simulation overhead was 132%, 420% and 171% of actual execution time for these three benchmarks. In contrast, our MPI profile based methodology used in this work has a maximum overhead of 0.05% of actual execution time.

WARPP simulator introduced in [37] also uses benchmarks to acquire target machine performance specific characteristics. WARPP prediction framework entails four steps: (1) model construction which is achieved by hand-coded simulation script programming that requires significant work by the user, (2) machine benchmarking using a reliable MPI benchmarking utility, a filesystem I/O benchmark and an instrumented version of the application, (3) the post-execution analysis of machine benchmarking

results to produce simulator inputs and finally (4) simulation. Although simulation in the last step proved to be significantly efficient and accurate in [37], step (1) requires significant manual source code analysis and instrumentation by the user. A similar approach to the one used in WARPP was also introduced in [38].

In [5], [6], [40], Snavely et al introduced a framework for performance modeling and prediction. In the framework, an application signature is created (single processor signature through MetaSim and communication signature through MPIDTrace). Then a machine profile is created (MAPS profile of memory and PMB profile of interconnect). Finally, the machine profile is convoluted with application signature to predict its performance. Projecting the MPI communication performance relies on an MPI trace and the Dimemas simulator [41]-[43]. These network traces and simulation have a significant overhead when compared to our profile based scheme.

Also in [62], Kerbyson et al. introduced the PACE framework where CHIPS application model is convoluted with hardware model of the target system to provide application performance projection. The workload definitions have an Application Layer and Parallel Template Layer. This powerful approach requires significant performance analysis effort.

5.6 Summary

In this section we presented a scheme for combining the computation and communication projection. In our scheme, the projected computation performance is combined with the HPC application scaling factor γ to produce the application projection at the required task count C_k on the target system. Similarly, the MM model that defines

the MPI communication scaling for the HPC application is combined with the MPI communication projection to produce the application projection at the required task count C_k on the target system. Once these two steps are completed, the projected performance of the two components are added together to produce the HPC application performance projection.

The Cache Scaling Model *CSM* is used to identify the point at which the HPC application cache behavior significantly impacts the application scaling. *CSM* model identifies at which task count the application data can be contained in a lower cache level causing hyper scaling in performance of the application.

6. SUMMARY AND FUTURE WORK

6.1 Summary

Using the framework provided in Figure 20, it is our goal to provide for an accurate and fast performance projection scheme for HPC applications on future systems. As indicated in section 2 and explained in details in sections 3, 4 and 5 our scheme involves deep analysis of computation and communication behavior of HPC applications on target systems. For the computation component, we developed a scheme to project the performance of HPC applications using surrogate workloads from the SPEC CPU2006 benchmark suite and hardware performance counter data. The scheme projected the performance of eight HPC applications on 2 IBM POWER6 systems with 7.2% average error and standard deviation of 5.3%. The results on systems with different ISAs than POWER are in the range of 8.3% and 12.8% for Intel Woodcrest and Intel Clovertown respectively which indicates that the base machine is a representative of a number of different systems. More importantly, the scheme is very flexible since it doesn't require any instrumentation to the binary code or the source code and only requires execution of the application and the benchmarks on one base machine. Moreover, simulation is not needed eliminating the long runtimes incurred in simulations. The use of a string based genetic tool reduces the projection scheme runtime significantly.

As for the communication component, we presented a method to project the performance of the MPI communication component of HPC applications using MPI profiles obtained on one base machine, TAMU Hydra utilizing a Federation interconnect, and IMB benchmark data obtained for the target system. The projected communication elapsed times for the three NAS MultiZone benchmarks were with 12.84% average error

on the BlueGene/P system utilizing a 3D Torus and collective tree interconnect. The average error on an IBM POWER6 575 system utilizing an InfiniBand interconnect was 12.81% with standard deviation of 1.13%.

The use of MPI profiles instead of MPI traces allows for efficient and fast projection. The use of MPI traces may yield more accurate results; however, projecting using MPI traces can be impossible in many cases for large scale production applications due to the huge sizes of the traces and the long time it takes to simulate these traces for target systems.

The *MM* model for an HPC application *A* assumes a constant dataset size *D* and a constant algorithm for all cores C_j. The *MM* model identifies how each MPI routine scales with different number of cores C_j. If the dataset size *D* changes or the algorithm is altered, a new *MM* model is required.

In our scheme, we model the time a task spends waiting on other tasks to finish as *WaitTime*. This is specifically important for MPI asynchronous calls in HPC applications that exhibit load imbalance between tasks. We calculate the scaling factor for the *WaitTime* from the base to the target system using the scaling of performance for individual MPI routines as well as the scaling in computation performance. The ratio of computation to communication on the base system, which may differ than the ratio on the target, as well as the computation scaling factor are the three parameters that affect the accuracy of projecting *WaitTime*; thus, the accuracy of projecting the performance of MPI routines with little *WaitTime* component, synchronous routines, is higher than projecting MPI routines with higher *WaitTime* since they don't depend on these three parameters.

In combining the computation and communication components projection to achieve a full application performance projection we introduced the methodology depicted in Figure 20 to combine the computation projection with the communication projection. We projected on three different systems utilizing a variety of processor architectures as well as different interconnect architectures. Our methodology accurately projected the performance of the HPC workloads with an average error of 11.22% and standard deviation of 1.18% on the three systems for the twelve workloads. Our methodology proved to be quite efficient as it doesn't require any simulation or massive traces.

6.2 Future Work

6.2.1 OpenMP Communication Projection

(a) OpenMP Profile: Typically an OpenMP profile contains the program general information (Header), region overview, program callgraph, flat region profile, callgraph region profiles, overhead analysis report and performance properties report [64]. In this work, we are mostly interested in the overhead analysis report. An overhead analysis report gives a detailed overview of overhead caused by the different OpenMP directives and this overhead contribution in the overall runtime. The overhead analysis can be visualized as a 2D matrix where the columns correspond to the type of overhead, i.e. thread management overhead, synchronization overhead, limited parallelism overhead or imbalance overhead, and the rows are the different directives such as Parallel, Parallel_Loop, Barrier etc. The overhead analysis profile shows the total overhead runtime for the different directives and their counts. In essence, we just need to capture

the different directives and their counts on the base machine. Thus, the OpenMP profile data provides for enough information.

(b) OpenMP uBenchmarks: The OpenMP uBenchmarks are [65] intended to measure the overheads of synchronization, loop scheduling and array operations in the OpenMP runtime library. The first part of the synchronization benchmark measures the overhead incurred by the following directives, all of which contain barrier synchronization:

- *PARALLEL*: Defines a parallel region, which is code that will be executed by multiple threads in parallel.
- *DO/for*: Non parallel DO/for loops.
- *PARALLEL DO/parallel for*: Parallel DO/for loops where work inside the DO/for is divided among threads.
- *BARRIER*: Synchronizes all threads in a team; all threads pause at the barrier, until all threads execute the barrier.
- *SINGLE*: Specifies that a section of code should be executed on a single thread, not necessarily the master thread.
- *WORKSHARE and PARALLEL WORKSHARE (for FORTRAN)*: divides the execution of the enclosed structured block into separate units of work, each of which is executed only once

The overhead is defined as follows: if T_s is the sequential time for a section of code, and T_p the time for the parallel version of this on p processors, then the overhead is given by $O_p = T_p - T_{s/p}$.

(c) OpenMP Proposed Projection Methodology: As in Figure 46, the OpenMP communication performance projection scheme entails 4 steps:

1. Execute HPC application on a base machine to collect OpenMP Profile for different task counts
    - Different task counts to understand application scaling
2. Decompose OpenMP profile into categories of directives for each task count as in Figure 46
3. Obtain OpenMP uBenchmarks data on target system for different task counts
    - Different task counts to capture the OpenMP overhead scaling
4. Map application OpenMP directives to uBenchmarks directives on target system and calculate communication runtime for different task counts

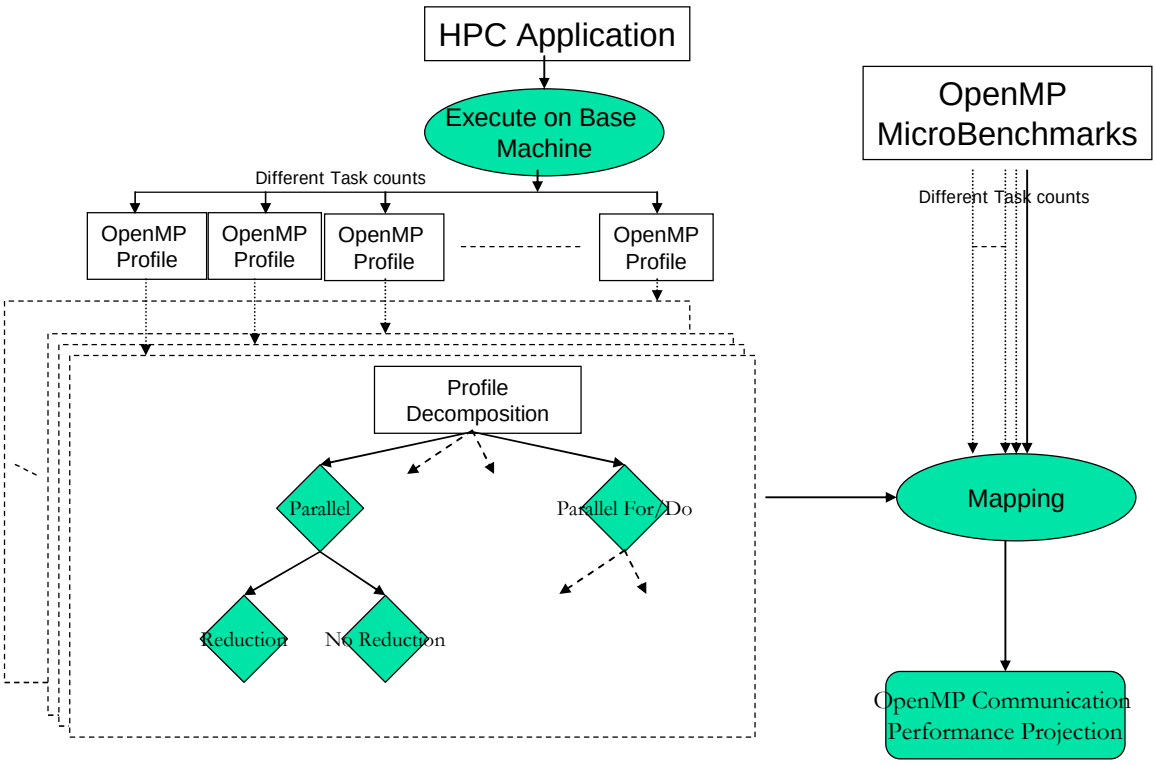

Figure 46: OpenMP communication performance projection framework

(d) Challenges: In the OpenMP proposed projection methodology, we face one major challenge. This challenge is similar to that of the MPI which is the separation of computation from communication. In fact, it is harder in the case of OpenMP than MPI since the communications in OpenMP is just a memory read and write operation and there is no explicit communication calls. To illustrate, in an MPI application, the separation of computation and communication can be achieved by turning off performance monitoring at MPI function entry and turning them on again at function exit; in openMP, this is hard to achieve since there is no explicit function entry and exit.

6.2.2 Hybrid Applications

Hybrid applications are composed of computation, MPI communications and OpenMP communications. In our approach, each of these components is projected separately. There may be, however, some overlapping communication time between OpenMP and MPI. To account for this overlap, we calculate an overlapping factor α as in Equation 24 on the base machine. The overlapping factor is then applied on the target machine.

$$\alpha = \frac{TotalExecutionTime}{ComputationRuntime + MPICommunicationRuntime + OpenMPOverhead} \quad (14)$$

The overall Hybrid application projection methodology can be summarized as follows:

1. Calculate overlapping factor between OpenMP and MPI on base machine
2. Project MPI communication overhead on target machine
3. Project OpenMP communication overhead on target machine

4. Combine the projection of both communication schemes and apply the overlapping factor

6.2.3 Overall Proposed Projection Framework

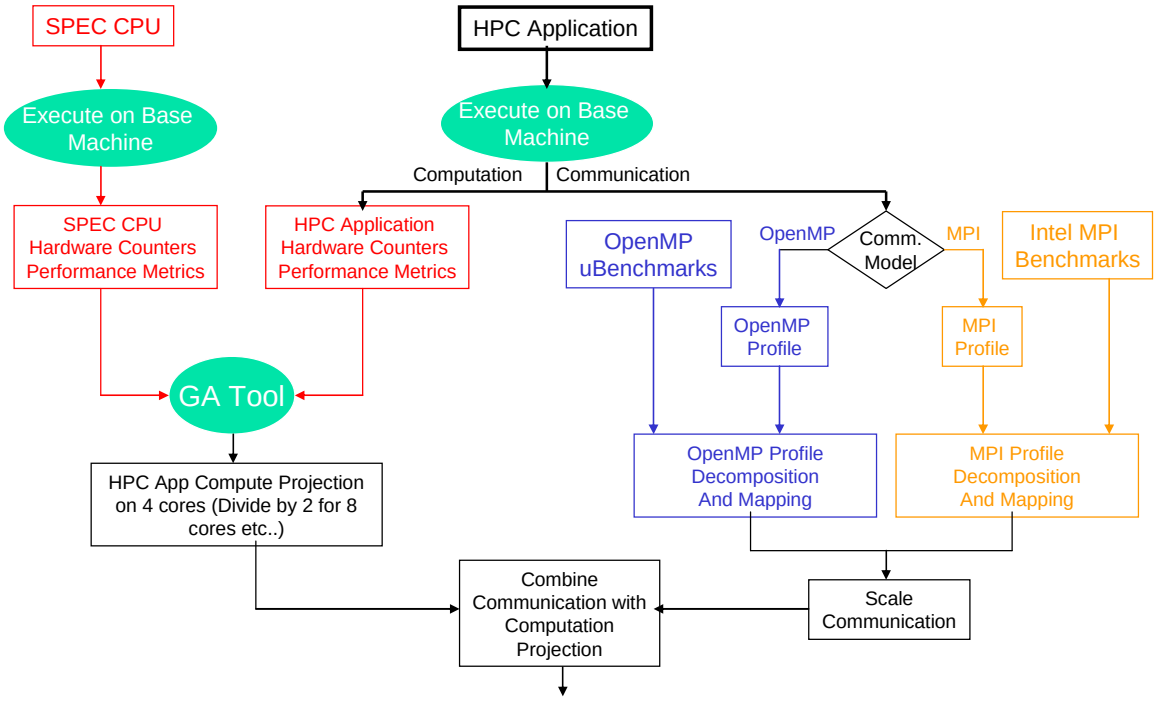

Figure 47: Proposed projection scheme framework

As indicated in Figure 47, we project computation and communication separately. This separation allows for the use of already available benchmark data without requiring any further execution for the surrogates. The SPEC CPU suite provides the surrogates for the computation component. The Intel MPI Benchmarks (IMB) or the OpenMP MicroBenchmarks provide the surrogates for the communication component. SPEC CPU, IMB or uBenchmarks data are available either online for target machine processors, or

through hardware vendors for communication interconnects. Also, the separation of computation from communication allows for maximum projection scalability.

6.2.4 Predicting Projection Error Value

Predicting the value of the projection error is very beneficial for both the HPC vendors and users. In fact, predicting the value of the projection error is a very challenging task; however, there are several possible means that could be utilized in predicting the value of error. One way is to find the relationship between the difference in metrics values of the surrogates and the application and the percent error. To illustrate, if the performance of application A on target machine X requires that metrics group 5 be the highest ranked group, then we do the following:

1 - Identify the percent difference between the metrics values of the application and the surrogate for group 5.

2 – Find the relationship between the percent difference of the metrics values for each metric group for target machine X and application A and the error percent.

3 – Apply the previous relationship to find the error percent.

REFERENCES

[1] L. Carrington, M. Laurenzano, A. Snavely, R. L. Campbell, L. P. Davis, "How Well Can Simple Metrics Represent the Performance of HPC Applications?" in *Proc. Supercomputing*, Seattle, WA, Nov. 2005.

[2] L. T. Yang, X. Ma, F. Mueller, "Cross-Platform Performance Prediction of Parallel Applications Using Partial Execution," in *Proc. Supercomputing*, Seattle, WA, Nov. 2005.

[3] D. Kerbyson, H. Alme, A. Hoisie, F. Petrini, H. Wasserman, M. Gittings, "Predictive performance and scalability modeling of a large-scale application," in *Proc. Supercomputing*, Denver, CO, Nov. 2001.

[4] J. Mellor-Crummey, R. Fowler, G. Marin, N. Tallent, "HPCView: A tool for top-down analysis of node performance," *The Journal of Supercomputing*, vol. 23, pp. 81-101, Aug. 2002.

[5] A. Snavely, N. Wolter, L. Carrington, "Modeling Application Performance by Convolving Machine Signatures with Application Profiles," in *IEEE 4th Annual Workshop on Workload Characterization*, Austin, TX, Nov. 2001.

[6] L. Carrington, N. Wolter, A. Snavely, "A Framework for Application Performance Prediction to Enable Scalability Understanding," in *Scaling to New Heights Workshop*, Pittsburgh, PA, May 2002.

[7] J. Mellor-Crummey, R. Fowler, G. Marin, "HPCView: A tool for top-down analysis of node performance," in *Proc. 2nd Los Alamos Computer Science Institute Annual Symposium*, Santa Fe, NM, Oct. 2001.

[8] G. Marin, J. Mellor-Crummey, "Application insight through performance modeling," in *Proc. IEEE 26th Int. Performance, Computing, and Communications Conf.*, New Orleans, LA, Dec. 2007.

[9] R.T. Mills, V. Sripathi, G. Mahinthakumar, G.E. Hammond, P.C. Lichtner, B.F. Smith, "Experiences and Challenges Scaling PFLOTRAN, a PETSc-based Code for Subsurface Reactive Flow Simulations, Towards the Petascale on Cray XT Systems," in *Proc. The Cray User Group Meeting*, Atlanta, GA, May 2009.

[10] X. Wu, V. Taylor, C. Lively, S. Sharkawi, "Performance Analysis and Optimization of Parallel Scientific Applications on CMP Clusters," *Scalable Computing: Practice and Experience*, vol. 10, no. 1, pp. 188–195, Mar. 2009.

[11] SPEC, "Standard Performance Evaluation Corporation," *SPEC* May 2010 [Online]. Available: http://www.spec.org

[12] M. Kuhnemann, T. Rauber, G. Runger, "A Source Code Analyzer for Performance Prediction," in *Proc. 18th Int. Parallel and Distributed Processing Symposium*, Santa Fe, NM, May 2004.

[13] L. DeRose, D. A. Reed, "Pablo: A Multi-language, Architecture-Independent Performance Analysis System," in *Proc. Int. Conf. on Parallel Processing*, Aizu-Wakamatsu, Fukushima, Japan, Sep. 1999.

[14] L. DeRose, Y. Zhang, D. A. Reed, "SvPablo: A Multi-Language Performance Analysis System," in *Proc. 10th Int. Conf. on Computer Performance Evaluation*, Palma de Mallorca, Spain, Jun. 1998.

[15] B.P. Miller, M.D. Callaghan, J.M. Cargille, J.K. Hollingsworth, R.B. Irvin, K.L. Karavanic, K. Kunchithapadam, T. Newhall, "The Paradyn Parallel Performance Measurement Tool," *IEEE Computer*, vol. 28, no. 11, pp. 37-46, Nov. 1994.

[16] HPCToolkit, "HPCToolkit Home", *HPCToolkit* Jan. 2011 [Online]. Available: http://www.hpctoolkit.org

[17] B.P. Miller, J.K. Hollingsworth, M.D. Callaghan, "The Paradyn Parallel Performance Tools and PVM," *Environments and Tools for Parallel Scientific Computing*, J.J. Dongarra and B. Tourancheau (Eds.), SIAM Press, SIAM, 1994.

[18] J.K. Hollingsworth, B.P. Miller, J. Cargille, "Dynamic Program Instrumentation for Scalable Performance Tools," in *Proc. Scalable High-performance Computing Conf.*, Knoxville, TN, USA, May 1994.

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

[20] R.F. Cmelik, D. Keppel, "Shade: A Fast Instruction Set Simulator for Execution Profiling," in *Proc. ACM SIGMETRICS Conf. on the Measurement and Modeling of Computer Systems*, Nashville, TN, Jun. 1994.

[21] S. Farfeleder, A. Krall, N. Horspool, "Ultra fast cycleaccurate compiled emulation of inorder pipelined architectures," *EUROMI-CRO Journal of Systems Architecture*, vol. 53, no.8, pp. 501-510, Aug. 2007.

[22] S. Prakash, R. L. Bagrodia, "MPI-SIM: using parallel simulation to evaluate MPI programs," in *Proc. 30th Conf. on Winter simulation*, Washington DC, Dec. 1998.

[23] T.L.Wilmarth, G. Zheng, E.J. Bohm, Y. Mehta, N. Choudhury, P. Jagadishprasad, L.V. Kale, "Performance prediction using simulation of large-scale interconnection networks in POSE," in *Workshop on Principles of Advanced and Distributed Simulation*, Monterey, CA, Jun. 2005.

[24] G. Zheng, G. Kakulapati, L.V. Kale, "BigSim: A parallel simulator for performance prediction of extremely large parallel machines," in *Proc.18th Int. Parallel and Distributed Processing Symposium*, Santa Fe, NM, May 2004.

[25] R. Susukita, H. Ando, M. Aoyagi, H. Honda, Y. Inadomi, K. Inoue, S. Ishizuki, Y. Kimura, H. Komatsu, M. Kurokawa, K.J. Murakami, H. Shibamura, S. Yamamura, Y. Yu, "Performance prediction of large-scale parallell system and application using macro-level simulation," in *Proc. Supercomputing*, Austin, TX, Nov. 2008.

[26] E. Grobelny, D. Bueno, I. Troxel, A.D. George, J.S. Vetter, "FASE: A Framework for Scalable Performance Prediction of HPC Systems and Applications," *Simulation*, vol. 83, no.10, pp. 721-745, Oct. 2007.

[27] A. Alexandrov, M.F. Ionescu, K.E. Schauser, C. Scheiman, "LogGP: incorporating long messages into the LogP model for parallel computation," *Journal of Parallel and Distributed Computing*, vol. 44, no. 1, pp. 71–79, Aug. 1997.

[28] D.E. Culler, R.M. Karp, D.A. Patterson, A. Sahay, K.E. Schauser, E. Santos, R. Subramonian, T. von Eicken, "LogP: towards a realistic model of parallel computation," in *Proc. 4th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, San Diego, CA, Feb. 1993.

[29] R. Blasko, "Hierarchical Performance Prediction for Parallel Programs," in *Proc. Int. Symposium and Workshop on Systems Engineering of Computer Based Systems*, Tucson, AZ, Mar. 1995.

[30] V. Taylor, X, Wu, X, Li, J, Geisler, Z. Lan, M. Hereld, I.R. Judson, R. Stevens, "Prophesy: Automating the Modeling Process," in *3rd Annual Int. Workshop on Active Middleware Services*, San Francisco, CA, Aug. 2001.

[31] M. Clement, M.J. Quinn, "Analytical Performance Prediction on Multicomputers," in *Proc. Supercomputing*, Portland, OR, Nov. 1993.

[32] Intel Corporation, "Intel MPI Benchmarks, Users Guide and Methodology Description (Version 3.1)," *Intel* May 2009 [Online]. Available: http://www3.intel.com/cd/software/products/asmona/eng/219848.htm

[33] M. Tikir, L. Carrington, E. Strohmaier, A. Snavely, "A Genetic Algorithms Approach to Modeling the Performance of Memory-bound Computations," in *Proc. Int. Conf. for High Performance Computing, Networking, and Storage*, Reno, NV, Nov. 2007.

[34] K. Hoste, A. Phansalkar, L. Eeckhout, A. Georges, L. John, K. De Bosschere, "Performance Prediction based on Inherent Program Similarity," in *Proc. 15th Int. Conf. on Parallel Architectures and Compilation Techniques,* Seattle, WA, Oct. 2006.

[35] J.L. Gustafson, R. Todi, "Conventional Benchmarks as a Sample of the Performance Spectrum," *The Journal of Supercomputing*, vol. 13, pp. 321-342, Jan. 1999.

[36] J. Zhai, W. Chen, W. Zheng, "PHANTOM: Predicting performance of parallel applications on large-scale parallel machines using a single node," in *Proc. 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, Bangalore, India, Feb. 2010.

[37] S.D. Hammond, G.R. Mudalige, J.A Smith, S.A. Jarvis, J.A. Herdman, A. Vadgama, "WARPP - A Toolkit for Simulating High Performance Parallel Codes," in *Proc. 2nd Int. Conf. on Simulation Tools and Techniques*, Rome, Italy, Mar. 2009.

[38] D.A. Grove, "Performance Modelling of Message-Passing Parallel Programs," *PhD Thesis*, Department of Computer Science, University of Adelaide, Roseworthy, SA Australia, Jan. 2003.

[39] R. F. Van der Wijngaart, H. Jin, "NAS parallel benchmarks, multi-zone versions," *NAS Technical Report NAS-03-010*, NASA Ames Research Center, Moffett Field, CA, 2003.

[40] A. Snavely, L. Carrington, N. Wolter, J. Labarta, R. M. Badia, A. Purkayastha, "A Framework for Performance Modeling and Prediction," in *Proc. Supercomputing*, Denver, CO, Nov. 2002.

[41] S. Girona, J. Labarta, R.M. Badia, "Validation of Dimemas communication model for MPI collective operations," in *Proc. EuroPVM/MPI*, Balatonfüred, Lake Balaton, Hungary, Sep. 2000.

[42] R.M. Badia, J. Labarta, J. Giménez, F. Escalé, "Dimemas: Predicting MPI applications behaviour in Grid environments," in *Workshop on Grid Applications and Programming Tools*, Seattle, WA, Jun. 2003.

[43] ERDC, "Dimemas," *DoD Supercomputing Resource Center*, May 2009 [Online]. Available: http://www.erdc.hpc.mil/hardSoft/Software/dimemas

[44] Intel Corporation, "Intel 5520 Chipset and Intel 5500 Chipset Datasheet," *Intel* Mar. 2009 [Online]. Available: http://www.intel.com/assets/pdf/datasheet/32132-8.pdf

[45] S. Gunther, R. Singhal, “Next Generation Intel Microarchitecture (Nehalem) Family: Architectural Insights and Power Management,” in *Intel Developer Forum*, San Francisco, CA, Aug. 2008.

[46] R. Singhal, “Inside Intel Next Generation Nehalem Microarchitecture,” in *Intel Developer Forum*, San Francisco, CA, Aug. 2008.

[47] IBM, “POWER5 system microarchitecture,” *IBM Journal of Research and Development* May 2007 [Online]. Available: http://www.research.ibm.com/-journal/rd/494/sinharoy.html

[48] IBM, “CPI analysis on POWER5, Part 1: Tools for measuring performance,” *developerWorks* May 2007 [Online]. Available: http://www.ibm.com/developerworks/power/library/pa-cpipower1/index.html

[49] IBM, “CPI analysis on POWER5, Part 2: Introducing the CPI breakdown model,” *developerWorks* May 2007 [Online]. Available: http://www.ibm.com/developerworks/power/library/pa-cpipower2/index.html

[50] IBM, “hpmcount command,” *AIX Documentation* May 2007 [Online]. Available: http://publib.boulder.ibm.com/infocenter/pseries/v5r3/index.jsp?topic=/com.ibm.aix.cmds/doc/aixcmds2/hpmcount.htm

[51] The Scripps Research Institute, “AMBER 9 Users’ Manual,” *AMBER* group May 2007 [Online]. Available: http://amber.scripps.edu/doc9/amber9.pdf

[52] CHARMM, “CHARMM Introduction,” *CHARMM* May 2007 [Online]. Available: http://www.charmm.org/

[53] ANSYS, “ANSYS FLUENT Flow Modeling Simulation Software,” *ansys* May 2007 [Online]. Available:http://www.ansys.com/Products/Simulation+Technology/Fluid+Dynamics/ANSYS+FLUENT

[54] Ames Laboratory Iowa State University, “The General Atomic and Molecular Electronic Structure System (GAMESS),” *Iowa State University* May 2007 [Online]. Available: http://www.msg.chem.iastate.edu/gamess/gamess.html

[55] LS-DYNA, "A combined Implicit/Explicit solver,” *LS-DYNA* May 2007 [Online]. Available: http://www.ls-dyna.com/

[56] CD-adapco, ”STAR-CD,” *cd-adapco* May 2007 [Online]. Available: http://www.cd-adapco.com/products/STAR-CD/index.html

[57] WRF, “The Weather Research and Forecasting Model,” *WRF* May 2007 [Online]. Available: http://www.wrf-model.org/index.php

[58] J. H. Holland, “Adaptation in Natural and Artificial Systems,” University of Michigan Press, Michigan, 1975.

[59] D. Bailey, J. Barton, T. Lasinski, H. Simon, “The NAS parallel benchmarks,” *Int. Journal of Supercomputer Applications*, vol. 5, no. 3, pp. 63-73, 1991.

[60] A. Phansalkar, A. Joshi, L.K. John, “Analysis of Redundancy and Application Balance in the SPEC CPU2006 Benchmark Suite,” in *Proc. 34th Annual Int. Symposium on Computer Architecture*, San Diego, CA, Jun. 2007.

[61] IBM, “IBM Parallel Environment,” *Parallel Environment (PE) Documentation* May 2009 [Online]. Available: http://www-03.ibm.com/systems/software-/parallel/index.html

[62] D.J. Kerbyson, E. Papaefstathiou, J.S. Harper, S.C. Perry, G.R. Nudd, “Is Predictive Tracing Too late for HPC Users?” *High Performance Computing*, R.J. Allan, A. Simpson, and D.A. Nicole (Eds.), Plenum Press, New York and London, 1999.

[63] N. Bhatia, S.R. Alam, J.S. Vetter, “Performance Modeling of Emerging HPC Architectures,” in *Proc. HPCMP Users Group Conf.*, Denver, CO, Jun. 2006.

[64] University of Tennessee Knoxville, “OpenMP Profiler,” *Department of Computer Science* May 2009 [Online]. Available: http://www.cs.utk.edu/~karl/ompp.html

[65] EPCC, “OpenMP MicroBenchmarks V2.0 Page,” *OpenMP MicroBenchmarks* May 2009 [Online]. Available: http://www2.epcc.ed.ac.uk/computing/research_-activities/openmpbench/openmp_index.html

VITA

NAME:	Sameh Sh Shawky Sharkawi
ADDRESS:	301 Harvey R. Bright Building, College Station, TX 77843-3112
EMAIL ADDRESS:	sss1858@cs.tamu.edu
EDUCATION:	B.S., Computer Science, The American University in Cairo, 2002
	M.S., Computer Science, Texas A&M University, 2006
	Ph.D., Computer Science, Texas A&M University, 2011