

Performance Projection of HPC Applications Using SPEC CFP2006 Benchmarks

Sameh Sharkawi†‡, Don DeSota†, Raj Panda†, Rajeev Indukuru†, Stephen Stevens†,
Valerie Taylor‡ and Xingfu Wu‡

†Systems and Technology Group, IBM, Austin

‡Department of Computer Science, Texas A&M University

Email: {sssharka, desotad, panda, indukuru, sstevens}@us.ibm.com,

{sss1858, taylor, wuxf}@cs.tamu.edu

Abstract

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, enable them to compare the application performance across different existing and future systems, and help HPC users with system procurement and application refinements. In this paper, we present a method for projecting the node level performance of HPC applications using published data of industry standard benchmarks, the SPEC CFP2006, and hardware performance counter data from one base machine. In particular, we project performance of eight HPC applications onto four systems, utilizing processors from different vendors, using data from one base machine, the IBM p575. The projected performance of the eight applications was within 7.2% average difference with respect to measured runtimes for IBM POWER6 systems and standard deviation of 5.3%. For two Intel based systems with different micro-architecture and Instruction Set Architecture (ISA) than the base machine, the average projection difference to measured runtimes was 10.5% with standard deviation of 8.2%.

1. Introduction

Performance projections of HPC applications onto various hardware platforms are important for hardware vendors and the community of HPC users. The projections aid hardware vendors in the design of future systems, enable them to compare the application performance across different existing and future systems, and help HPC users with system procurement

and application refinements. In this paper, we present a scheme to project the node level performance of HPC applications onto different systems using widely available benchmark performance data, in particular SPEC CFP2006[14], and the hardware counter data collected on one base machine, the IBM p575[18]. The main advantage of this scheme is the use of published data about the target machine; the target machine need not be available for experiments for the projections. Further, our method does not involve any simulations, which are often very time-consuming.

The motivation behind the performance prediction work is to facilitate the process of performance projection of HPC applications for systems designers, HPC sales personnel as well as HPC customers and users. System designers need projections because availability of new platforms for HPC applications measurements is limited in most cases, assuming the platforms exist. On the other hand, simulations of such complicated applications on future systems (systems in design phase yet to be built) are extremely time consuming. Assuming the simulators are accurate and available for such systems, scaling an HPC application for a simulator is nearly impossible. Furthermore, sales teams require performance projections for customers planning to purchase HPC systems. It is important to understand how customer's application will perform on a system before it is built. For HPC users, on the other hand, best HPC platform selection is a challenging task. Knowing in advance how applications will execute on different platforms will help select the best platform. Further, the projections can aid in identifying areas for performance refinements.

Figure 1 depicts the high level framework of our scheme. The SPEC CFP2006 benchmarks are executed once on the base machine and the resultant hardware performance counter data is archived for use as needed. In addition, the HPC applications are executed once on

the base machine and the resultant hardware counter data is archived. A tool based on a Genetic Algorithm (GA) [16] 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. This group of benchmarks is given the name “surrogates” for the remainder of the paper. Performance data of the surrogates is then used to project the performance of the application onto a target machine. The performance data for the surrogates on the target machine is obtained from published data from actual execution or simulations of the target system.

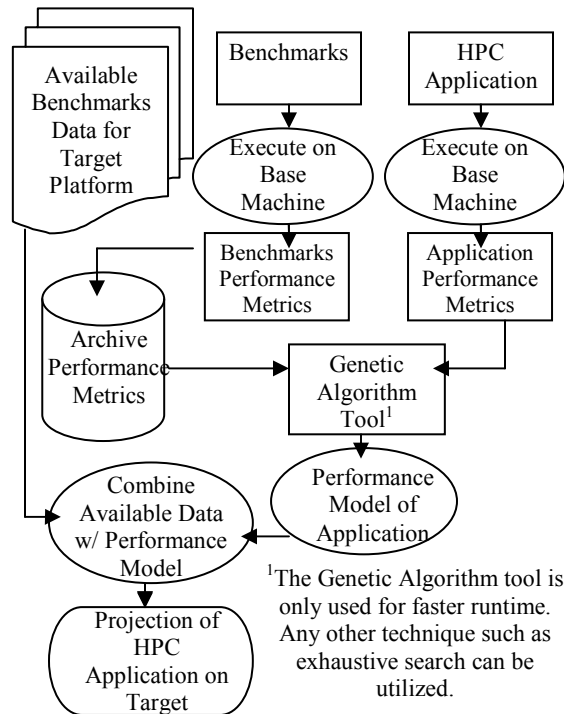


Figure 1. Framework for projection scheme

The proposed method for projection allows one to explore the following issues: (1) new insights that result when one characterizes the behavior of HPC applications based upon a common set of benchmarks using hardware counter data, (2) the robustness of the base machine used to develop the performance model (e.g., is the base machine representative of a number of different systems), and (3) the validity of using published data to project an application performance onto a target systems.

The proposed projection method uses the 17 SPEC CFP2006 benchmarks to project the performance of eight HPC applications onto four systems, two utilizing IBM POWER6 and two utilizing Intel Woodcrest and Clovertown, respectively. The HPC applications

include the following areas: Computational Fluid Dynamics, Molecular Dynamics, Weather Simulation, Seismic Analysis, Crash testing and Quantum Chemistry. The results of the projections onto the IBM POWER6 systems had an average error of 7.2% of measured runtimes with standard deviation of 5.3%. For the Intel machines, the average error was 10.5% with a standard deviation of 8.2%.

The remainder of this paper is organized as follows. Section 2 describes the problem in details. Section 3 describes the details of the performance projection scheme and how the GA tool is used in matching HPC applications to the benchmarks. Section 4 shows the experimental results of our scheme. Section 5 discusses related work followed by the paper summary in Section 6.

2. Background

This work presents a scheme for projecting HPC application node level performance based on the similarity of micro-architecture dependent performance characteristics of SPEC CFP2006 benchmarks as indicated in Figure 1. HPC application performance can be described as a function of computation and communication. In this work we only focus on projecting the compute performance of the HPC application. We express the compute behavior of application/benchmarks using a set of metrics based on hardware performance counters collected on one base machine. Recall that our scheme utilizes a GA tool to generate a performance model of the HPC application as a function of the surrogates SPEC CFP2006 benchmarks. The surrogates are selected based upon the best fit of the metrics. In this section we discuss the details of the base machine and the metrics generated from the hardware counters.

2.1. Base machine

The base machine used for this work is the p575 POWER5+ system. 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 do. To the operating system, each thread is considered a separate logical CPU. 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[1]. The hardware counter data is normalized to the number of instructions for the application or benchmark.

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. Surrogates that behave similarly to the HPC application under different computing conditions are a better representation for the application on different architectures.

The SPEC CFP2006 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.

2.2. Base machine: hardware counters and metrics

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[2]. We use the HPMCOUNT[21] tool on IBM systems to collect our hardware counter data.

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, G_2 -- CPI Stall Cycles [3], G_3 -- Floating Point Instructions, G_4 -- ERAT, SLB and TLB caches miss rates, G_5 – Data Cache Reloads and G_6 – Memory Bandwidth. 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. A detailed description of these groups is provided in Table 1.

2.3. Base machine: relating metrics and metrics' groups to runtime

Relating each metric to the runtime 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, one local to a group and one across all 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.

Table 1. Metrics 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
	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
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
	m _{5,3}	DATA_FROM_LOCAL_MEM_PI Demand d-L1 Reloads from Local Memory per

			Instruction
	m _{5,4}	DATA_FROM_REMOTE_MEM_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)

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 that 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 cycles required for the floating-point multiply-add operation, Y cycles required for other floating-point operations, and Z cycles required for floating-point store operations. Similarly, in G_5 , $c_{5,1}$ (DATA_FROM_L2_PI), $c_{5,2}$ (DATA_FROM_L3_PI), $c_{5,3}$ (DATA_FROM_LOCAL_MEM_PI) and $c_{5,4}$ (DATA_FROM_REMOTE_MEM_PI) have the values of X , Y , Z , K respectively, corresponding to X cycles required to load a cache line from L2, Y cycles required to load a cache line from L3 etc...

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 . H can be defined as follows:

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

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 typical cycles associated with each group relative to the runtime. Since the function F_i in Equation 1 was calculated based on values obtained

from the micro-architecture specification of the base machine, we use F_i to calculate the coefficient a_i for each G_i in Equation 2.

3. Performance projection scheme

The process of performance projection of HPC applications, in this work, entails three steps. The first step involves characterizing/modeling each HPC application by providing ranks 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 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 and modeling on the target machine to be used with the similarity analysis to produce better results.

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 2.3. 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 $a_i G_i$ for this group, the higher the magnitude of $a_i G_i$ the higher the rank of the group G_i .

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_{b_l} using Equation 2 as follows:

$$H_{b_l} = a_1 G_{1_{b_l}} + a_2 G_{2_{b_l}} + a_3 G_{3_{b_l}} + a_4 G_{4_{b_l}} + a_5 G_{5_{b_l}} + a_6 G_{6_{b_l}}$$

$$\text{and } H_{b_l} \propto R_{b_l} \forall l \in B \quad (3)$$

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

$$H_{b_l}' = a_1' G_{1_{b_l}} + a_2' G_{2_{b_l}} + a_3' G_{3_{b_l}} + a_4' G_{4_{b_l}} + a_5' G_{5_{b_l}} + a_6' G_{6_{b_l}}$$

$$\text{and } H_{b_l}' \propto R_{b_l}' \forall l \in B \quad (4)$$

where R_{b_l}' 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_{b_l}' as follows:

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

Since H_{b_l} can be calculated for the base machine, and runtimes R_{b_l} of the base and R_{b_l}' 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 $a_i'G_i$ for this group, the higher the magnitude of $a_i'G_i$ the higher the rank of the group G_i .

3.3. Identifying benchmarks

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 form the set S_{app} which is the set of selected surrogates for an application app . S_{app} is a subset of the set B , the set of SPEC CFP2006 benchmarks, and $S_{app} \subseteq B$. Each member s_k of S_{app} has a weight w_k . The combination of these surrogates called *comb_surrogates* is defined as follows:

$$comb_surrogates = \sum_{k=1}^{|S_{app}|} w_k \times s_k \quad (6)$$

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

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

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 1. 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_{s_k} 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_{s_k}/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 PST modes as mentioned earlier in Section 2.1, the GA tool attempts to minimize the error in metrics for each of the six groups for both SMT and PST 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:

$$if\left(\frac{E_1}{\sum_{q=1}^{M_1} m_{1,q(app)}} < 0.1 \& \frac{E_2}{\sum_{q=1}^{M_2} m_{2,q(app)}} < 0.1 \& \dots \& \frac{E_6}{\sum_{q=1}^{M_6} m_{6,q(app)}} < 0.1\right) \\ return(0) \quad (9)$$

else

$$return\left(\frac{E_1}{\sum_{q=1}^{M_1} m_{1,q(app)}} + \frac{E_2}{\sum_{q=1}^{M_2} m_{2,q(app)}} + \dots + \frac{E_6}{\sum_{q=1}^{M_6} m_{6,q(app)}}\right)$$

The GA tool terminates when the return value of the fitness function is 0; thus, for each of the six groups for both SMT and PST modes the term $\frac{E_i}{\sum_{q=1}^{M_i} m_{i,q(app)}}$ is < 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 3.2. The HPC application characterization/modeling on the target machine using metrics' group ranking in combination with the similarity analysis produce 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 surrogates and their 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.

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

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.

3.4. 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 surrogate, n is the number of surrogates and m the combination size. In this work, we used a string based Genetic Algorithm (GA) tool [16]. Parameters for the genetic algorithm are shown in Table 2. 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 2 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 (2000).

Table 2. Genetic 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

Recall we configure each HPC application to run four tasks each bound to a separate thread in SMT mode using 2 cores on one chip or each bound to a separate core in PST mode using a total of four cores on two chips. The SPEC CFP2006 benchmarks are run in throughput mode using the same configuration.

Since the SPEC benchmarks are serial workloads, the average work done by each task is the same across the four tasks. We assume that the same case applies to the HPC applications and the average work done by each task is similar across tasks. In other words, the metrics average across tasks should be very close to the metrics of each task. In this sense, we use the average metrics across the four tasks for the HPC applications and the SPEC CFP2006 workloads.

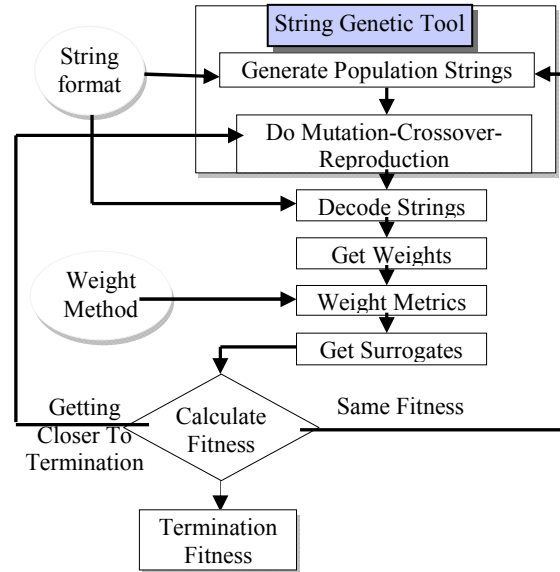


Figure 2. Genetic tool framework

4. Experimental results

We used our method given in Figure 1 to project the performance of the following eight large-scale scientific applications: AMBER[7], CHARMM[8], FLUENT[9], GAMESS[10], LS-DYNA[11], a seismic application that will be referred to as Seismic, STAR-CD[12] and WRF[13]. Table 3 lists the input datasets and the category for the applications we used in our experiments. Table 4 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 4, 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.

In figures 3,4,5 and 6 we show the absolute value of the error in runtime. In this work, we focused on reducing the magnitude of the runtime error. The number of selected benchmarks (surrogates) for our applications was ranging between a minimum of one and a maximum of four surrogates as in Table 5. Typically, the surrogate(s) with the highest weights were from the same scientific area of the HPC application.

Table 3. HPC applications and their datasets

Application	Category	Datasets(s)
AMBER	Molecular Dynamics	Gb-Cox2, Factor IX, Jac
FLUENT	Fluid Dynamics	L1, L2, L3, M1, M2 and M3
LS-DYNA	Crash Simulation	3 Car Crash
STAR-CD	Fluid Dynamics	C-Class
CHARMM	Molecular Dynamics	Alanine Dipeptide
GAMESS	Ab Initio Quantum Chemistry	L-ROTONON, SICCC
Seismic	Seismic	N/A
WRF	Weather	ConUS

Table 4. Base system and other systems used for validation

Machine	Processor	Cores	Freq.	Memory Per core	L2 Cache per core
IBM p575	Out of Order Execution POWER5+	2	1.9 GHz	4GB	1.9 MB shared
IBM JS22[19]	In Order Execution POWER6	2	4.0 GHz	4GB	4 MB
IBM p570[20]	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

Figure 3 shows that our scheme was able to predict the performance of the HPC applications within 5.5% average 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 .

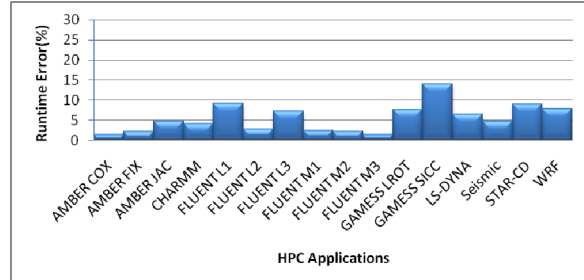


Figure 3. Projection results for POWER6 IBM JS22 system

Figure 4 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 5 and 6.

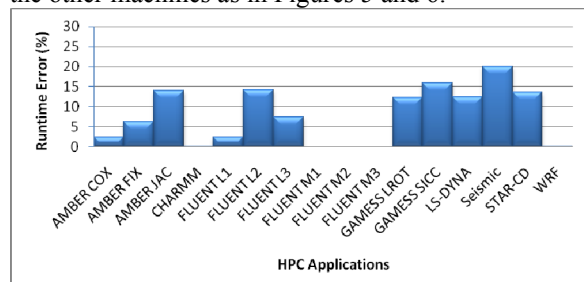


Figure 4. Projection results for POWER6 IBM p570 system

Figure 5 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%.

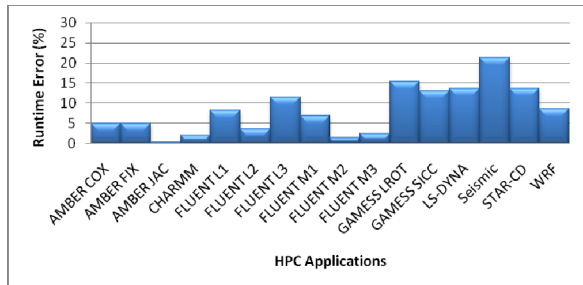


Figure 5. Projection results for Intel Woodcrest IBM x3550 system

Figure 6 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.

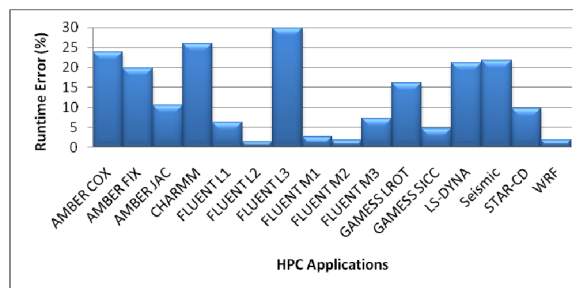


Figure 6. Projection results for Intel Clovertown IBM x3650 system

Table 5 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 5, all surrogates for HPC application are typically from the same area of the HPC application. From a micro-architectural perspective, the combined surrogates 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 5. Surrogates for the eight 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

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.

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 [17] 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 [22] 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 et al. in [4] 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 et al. in [6] 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. The major fundamental difference between [6] and our proposed methodology is that we rank metrics and metrics groups not only based on application properties as in [6] but also based on the properties of the system, such as pipeline stalls and cache hit rates, used to execute the application. This fundamental difference between our work and [6] allows for producing good projection results for each target system since one set of surrogates may not produce the best projections for all target systems due to differences in target systems properties. Also, a perfect match between surrogates and an HPC application is often not achievable, matching on the most important metric groups, however, is more likely to occur.

Tikir et al. in [5] 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.

6. Conclusions

We presented a scheme to project the performance of HPC applications using surrogate workloads from the SPEC CFP2006 benchmark suite and hardware performance counter data. The scheme projected the performance of 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.

SPEC CFP2006 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 CFP2006 and can easily incorporate other benchmark suites such as SPEC CINT2006, SPEC MPI2007, NAS Parallel Benchmarks or others. The choice of SPEC CFP2006 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 CFP2006 suite from one base machine to model the behavior of the HPC applications as a function of the benchmarks of SPEC CFP2006. 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 as in the case of Clovertown. For those applications with the highest ranked group(s) having significantly higher metrics' errors, our scheme indicates that those projections may be inaccurate.

In this work, we focused on projecting the node level performance of HPC applications. As stated earlier application performance consists of computation time and communication time. We have addressed projection of computation time in this paper. Future work will extend this work to inter-node level performance projection. This will involve projection and modeling of the communication component of HPC applications allowing us to project HPC application performance on cluster systems.

7. References

- [1] POWER5 system microarchitecture. <http://www.research.ibm.com/journal/rd/494/sinharoy.html>
- [2] CPI analysis on POWER5, Part 1: Tools for measuring performance. <http://www.ibm.com/developerworks/power/library/pa-cpipower1/>
- [3] CPI analysis on POWER5, Part 2: Introducing the CPI breakdown model <http://www.ibm.com/developerworks/power/library/pa-cpipower2/index.html>
- [4] Aashish Phansalkar, Ajay Joshi and Lizy K. John, "Analysis of Redundancy and Application Balance in the SPEC CPU2006 Benchmark Suite". Proceedings of the 34th annual International Symposium on Computer architecture, 2007, San Diego.
- [5] M. Tikir, L. Carrington, E. Strohmaier and A. Snively, "A Genetic Algorithms Approach to Modeling the Performance of Memory-bound Computations". Proceedings of the International Conference for High Performance Computing, Networking, and Storage, November 2007, Reno.
- [6] K. Hoste, A. Phansalkar, L. Eeckhout, A. Georges, L. John and K. De Bosschere, "Performance Prediction based on Inherent Program Similarity". Proceedings of the 15th International Conference on Parallel Architectures and Compilation Techniques, 2006, Seattle, Washington.
- [7] AMBER 9 Users' Manual. <http://amber.scripps.edu/doc9/amber9.pdf>
- [8] CHARMM, <http://www.charmm.org/>
- [9] FLUENT, http://www.fluent.com/software/fluent/modeling_spec.htm
- [10] GAMESS, <http://www.msg.chem.iastate.edu/gamess/gamess.html>
- [11] LS-DYNA, <http://www.ls-dyna.com/>
- [12] STAR-CD, <http://www.cd-adapco.com/products/STAR-CD/index.html>
- [13] The Weather Research and Forecasting Model, <http://www.wrf-model.org/index.php>
- [14] SPEC, <http://www.spec.org>
- [15] The STREAM Benchmark: Computer Memory Bandwidth, <http://www.streambench.org/>
- [16] J. H. Holland, *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975
- [17] D. Bailey, J. Barton, T. Lasinski, H. Simon, "The NAS parallel benchmarks", *International Journal of Supercomputer Applications*, 1991.
- [18] IBM System p5 575, <http://www-03.ibm.com/systems/p/hardware/highend/575/>
- [19] IBM BladeCenter JS22 Express, <http://www-03.ibm.com/systems/bladecenter/hardware/servers/js22e/index.html>
- [20] IBM System p570, http://www-03.ibm.com/systems/p/hardware/midrange_highend/p570/index.html
- [21] hpmcount command, <http://publib.boulder.ibm.com/inforcenter/pseries/v5r3/index.jsp?topic=/com.ibm.aix.cmds/doc/aixcmds2/hpmcount.htm>
- [22] J.L. Gustafson and R. Todi, "Conventional Benchmarks as a Sample of the Performance Spectrum," *The Journal of Supercomputing*, Vol. 13, pp 321-342, 1999.