Noname manuscript No.
(will be inserted by the editor)

Power-Aware Predictive Models of Hybrid (MPI/OpenMP) Scientific Applications on Multicore Systems

Charles Lively · Xingfu Wu · Valerie Taylor · Shirley Moore · Hung-Ching Chang · Chun-Yi Su · Kirk Cameron

Received: date / Accepted: date

Abstract Predictive models enable a better understanding of the performance characteristics of applications on multicore systems. Previous work has utilized performance counters in a system-centered approach to model power consumption for the system, CPU, and memory components. Often, these approaches use the same group of counters across different applications. In contrast, we develop application-centric models (based upon performance counters) for the runtime and power consumption of the system, CPU, and memory components. Our work analyzes four Hybrid (MPI/OpenMP) applications: the NAS Parallel Multizone Benchmarks (BT-MZ, SP-MZ, LU-MZ) and a Gyrokinetic Toroidal Code, GTC. Our models show that cache utilization (L1/L2), branch instructions, TLB data misses, and system resource stalls affect the performance of each application and performance component differently. We show that the L2 total cache hits counter affects performance across all applications. The models are validated for the system and component power measurements with an error rate less than 3%.

Keywords Modeling · Performance Prediction · Power Prediction · Multicore Systems · Performance Analysis

C. Lively, Xingfu Wu, Valerie Taylor
Department of Computer Science,Texas A&M University
E-mail: clively,wuxf,taylor@cse.tamu.edu

Shirley Moore
EECS, University of Tennessee-Knoxville
E-mail: shirley@eecs.utk.edu

Hung-Ching Chang, Chun-Yi Su, Kirk Cameron
Department of Computer Science, Virginia Tech
E-mail: kirk.w.cameron@gmail.com

1 Introduction

The current trend in high-performance computing places a great focus on minimizing the power consumption of scientific applications on multicore systems without increasing runtime performance [7–9,18,20,21,24]. Performance models can be used to provide insight into the application's performance characteristics that significantly impact the runtime and power consumption. As HPC multicore systems become more complex it is important to understand the relationships between performance and power consumption and the characteristics of scientific applications that influence the various levels of performance. In this paper we develop application-centric models based upon performance counters for modeling the runtime and power consumption of the system, CPU, and memory (which are the major components of the overall power consumption). We use these models to explore common and different characteristics about the applications that impact runtime and power consumption.

The hierarchical organization of multicore systems lends itself well to the Hybrid (MPI/OpenMP) programming model. The Hybrid programming model exploits the intra-node parallelization available in multicore chips via OpenMP and the scaling to large number of cores using inter-node parallelism via MPI. Utilizing OpenMP within a node reduces communication overhead and makes use of the memory bandwidth available within a node. Utilizing MPI between nodes matches the system organization with distributed memory across nodes. In this paper we focus on Hybrid applications to investigate performance models. In particular, we utilize the following four Hybrid applications: the NAS Parallel Multizone Benchmark suite, which consists of

BT-MZ, SP-MZ, LU-MZ (strong scaling) [12], and a Gyrokinetic Toroidal Code, GTC (weak scaling).

The use of performance counters to predict power consumption has been explored in previous work [2–4, 6,13,15,22]. Often this work identified a set of common performance counters to be used across all of the applications considered. Further, some of the work uses the performance counters to identify methods for reducing power consumption. Typical methods for reducing power include power scaling techniques, such as dynamic voltage and frequency scaling (DVFS) [8], and dynamic concurrency throttling (DCT) [4]. DVFS and DCT seek to reduce the power consumption of applications by reducing frequency during low activity periods or determining the most efficient execution configuration dynamically during execution, respectively. These methods exploit imbalances in the application and reduce power consumption based on the performance of the current systems. In contrast, our method is focused on an application-centric view of the models developed and thereby understanding the unique characteristics of each application that impact both runtime and power consumption. We seek to explore which application characteristics (via performance counters) affect performance in order to gain a better understanding of how the application can be modified to improve performance with respect to runtimes, and power consumption of the system, CPU, and memory.

In this paper, we explore the following issues in regards to modeling the runtime and power consumption of Hybrid scientific applications on multicore systems:

a) What is the accuracy of application-centric models in estimating performance (runtime and power consumption for system, CPU, and memory components?

b) Which combination of performance counters can be used to model the Hybrid application in terms of runtime, system power, CPU power, and memory power?

c) What are the application characteristics that affect runtime and power consumption in Hybrid scientific applications?

d) What characteristics of Hybrid applications can be optimized to improve performance on multicore systems?

In this work, we use MuMI (Multicore application Modeling Infrastructure) [17], which facilitates systematic measurement, modeling, and prediction of performance, power consumption and performance-power tradeoffs for multicore systems. The MuMI framework is an integration of existing frameworks: Prophesy [25], PowerPack [23], and PAPI [19]. We use the SystemG power-aware cluster to conduct our experiments. Table 1 provides an overview of the overall performance specifications of SystemG. SystemG, available at Virginia Tech, is a 22.8 TFLOPS research platform that utilizes 325 Mac Pro computer nodes. This system is the largest power-aware research system and with each node containing more than 30 thermal sensors and more than 30 power sensors. Each node in the system contains two quad-core 2.8GHz Intel Xeon Processors 5400 series processor. Each node is configured with 8GB RAM which is configured with eight 1GB 1066MHz DDR3 ECC DIMMs .

Table 1 Configuration of SystemG

Configuration of SystemG	
Mac Pro Model Number	MA970LL/A
Total Cores	2,592
Total Nodes	324
Cores/Socket	4
Cores/Node	8
CPU Type	Intel Xeon 2.8Ghz Quad-Core
Memory/Node	8GB
L1 Inst/D-Cache per core	32-kB/32-kB
L2 Cache/Chip	12MB
Interconnect	QDR Infiniband 40Gb/s

Our models have an average error of less than 3% for runtime, system power, CPU power and memory power and are validated with actual component power measurements. Our models highlight that L1 cache utilization (reads/writes/accesses/hits/misses), L2 cache utilization (accesses/hits), branch instructions, TLB data misses, and various system resource stalls affect performance differently for each application and performance component. Additionally, our models show that L2 total cache hits per cycle affects performance greatly across all applications and all performance components.

The remainder of this paper is organized as follows. Section 2 presents an overview of our modeling methodology. Section 3 presents the validation and results of our modeling methodology. Section 4 presents related work followed by the paper summary in Section 5.

2 Model Development Method

Our models are developed using five configuration points for each application to constitute a training set for predicting performance. Three of the configuration points are used for predicting intra-node performance and two are used for predicting inter-node performance. Scaling processor performance for each MPI x OpenMP configuration is represented as (M x N), where M is the number of MPI processes or nodes used, and N is the number of OpenMP threads utilized per node. One MPI process is utilized per node in all of our experiments. A configuration of 11x8 means that 11 MPI processes (11 nodes with 1 MPI process per node) with

8 OpenMP threads per node were used for a total of 88 processors. The training set is used to predict the larger performance components. In this sense, an applications training set would consist of an intra-node set using performance of 1 thread (1x1), 2 threads (1x2), and 3 threads (1x3) within a node. The training set for inter-node performance would use two data points consisting of performance using the configurations for 1x8 (1 MPI/8 OpenMP threads) or 8 processes and 2x8 (2 MPI/8 OpenMP threads) or 16 processes. The 5 training points were used to predict performance for up to 16 larger configuration points. We developed predictive models for the following Hybrid (MPI/OpenMP) configuration points: 1x4, 1x5, 1x6, 1x7, 3x8, 4x8, 5x8, 6x8, 7x8, 8x8, 9x8, 10x8, 11x8, 12x8, 13x8, 14x8, 15x8, 16x8.

During each execution we capture 40 performance counter events utilizing the performance application programming interface (PAPI) [19] and the perfmon performance library. All performance counter events are normalized using the total cycles of execution to create performance event rates for each counter. Curve fitting is used to extrapolate the smaller counter rates to the counter rates for the larger configurations. The 40 performance counters are analyzed for each application using a performance-tuned supervised principal component analysis method, which is a derivative of the supervised principal components analysis method [1]. In particular, the following algorithm was used to identify the performance counter events needed to build the predictive models for each application:

1. Compute the Spearman's rank correlation for each performance counter event rate for each performance component (runtime, system power, CPU power, and memory power).
2. Establish a threshold, β_{ai}, to be used and eliminate any counters below the threshold.
3. Compute a multivariate linear regression model based upon the remaining performance counter event rates.
4. Establish a new threshold, β_{bi}, and eliminate performance counters and ensure that the regression coefficients are not greater than the selected threshold in terms of magnitude.
5. Compute the principal components of the reduced performance counter event rates.
6. Use the performance counter event rates with the highest principal component coefficient vectors to build a multivariate linear regression model to predict the respective performance metric.

This process is repeated for each application and for each performance component.

Table 2 provides a sample of the β_{ai} and β_{bi} values used in our work for modeling runtime. The β_{ai} were used to identify the performance counters with the strongest correlation to each application and performance component. Different values for β_{ai} and β_{bi} were used for system power, CPU power, and memory power component modeling.

Table 2 Correlation θ values for Runtime

β	BT-MZ	SP-MZ	LU-MZ	GTC
β_{ai}	0.60	0.50	0.60	0.60
β_{bi}	0.100	3.50	6.000	1.250

Table 3 Overview of Normalized Performance Counters

Counter	Description
PAPI_TOT_INS	Total instructions completed
PAPI_TLB_ DM	TLB misses
PAPI_L1_TCA	L1 cache total accesses
PAPI_L1_ICA	L1 instruction cache accesses
PAPI_L1_TCM	L1 total cache misses
PAPI_L1_DCM	L1 data cache misses
PAPI_L2_TCH	L2 total cache hits
PAPI_L2_TCA	L2 total cache accesses
PAPI_L2_ICM	L2 instruction cache misses
PAPI_BR_INS	Branch instructions completed
PAPI_RES_STL	System stalls on any resource
Cache_FLD_per_instruction	L1 writes/reads/hits/misses
LD_ST_stall_ per_cycle	Load/stores stalls per cycle

We are able to utilize a small subset of the original 40 performance counters (presented in Table 3). Throughout our experiments the performance counter events collected are per cycle event rates and are normalized based on the total cycles of execution for each application. In addition, in our work several restrictions were placed on our derived predictive models to ensure that they were representative of realistic performance scenarios. In several cases, utilizing some counters resulted in negative regression coefficients for those counters. For example, a runtime predictive model that utilizes PAPI_TOT_INS (normalized per cycle) should not have a negative regression coefficient. This would mean that as the number instructions per cycle increases the runtime would decrease. Theoretically this could provide an accurate model for predicting runtime but it is a blackbox approach, which is not indicative of a realistic performance scenario. Based on our modeling methodology (which combines correlation, principal component analysis, and regression), we assume a causal relationship between the derived performance counter event rates and each performance component.

These performance counters represent a smaller subset of the total number of performance counters used to measure the application performance on SystemG.

This subset provides for a representation of the counters needed for the application-centered predictive models.

$$y = \beta_0 + \beta_1 * r_1 + \beta_2 * r_2 .....\beta_n * r_n \quad (1)$$

Each multivariate linear regression model is constructed for each performance component (execution time, system power, CPU power, and memory power) for each application. In Equation 1, y is the component predictor used to represent the value for runtime, system power, CPU power, or memory power. The intercept is β_0 and each β_n represents the regression coefficient for hardware counter r_n.

3 Experimental Results: Model Development and Validation

In this section, we provide the details of our experimental results. This section is organized as follows. Section 3.1 provides an overview of the Hybrid MPI/OpenMP applications utilized in our experiments. Section 3.2 provides an overview of the predictive model validation discussing commonalities across all applications for each performance component (runtime, system power, CPU power, and memory power. Section 3.3 provides an overview of modeling validation for each application and discusses application characteristics that affect each performance component.

3.1 Benchmarks and Scientific Applications

We model the runtime, system , CPU, and memory power consumption of Hybrid scientific applications. We utilize the MultiZone NAS Parallel Benchmark Suite (NPB-MZ). NPB-MZ contains three benchmarks (LU-MZ, SP-MZ, and BT-MZ), using a main loop to exchange values during MPI communication and an OpenMP phase within the loop.

The Block Tri-diagonal algorithm (BT-MZ) contains (16x16) x-zones x y-zones and has uneven mesh tilings. BT-MZ represents realistic performance case for exploring the discretization meshes in parallel computing. The Scalar Penta-diagonal algorithm (SP-MZ) contains (16x16) x-zones x y-zones and is representative of a balanced workload in the suite. The Lower-Upper symmetric Gauss-Seidel algorithm (LU-MZ) contains (4x4) x-zones x y-zones and the coarse-grain parallelism of LU-MZ is limited to 16. Therefore, at most 16 MPI processes can be used in executing LU-MZ. The problem sizes for all NPB-MZ benchmarks are strong scaling using class C, utilizing 800MB of memory.

The Gyrokinetic Toroidal code (GTC) is a 3D particle-in-cell application developed at the Princeton Plasma Physics Laboratory to study turbulent transport in magnetic fusion. GTC is a flagship SciDAC fusion microturbulence code written in Fortran90, MPI and OpenMP. There are 7 major functions: load, field, smooth, poisson, charge, shift and pusher in the code. Charge, pusher and shift dominate most of the runtime. Note that GTC is executed in weak scaling to keep a constant workload per processor as the number of processors increase using 100 particles per cell and 100 time steps.

In our experiments with NPB-MZ we test inter-node performance from 1 OpenMP thread to 8 OpenMP threads for a single node. NPB-MZ performance was predicted from up to 11x8 (88 processors) and up to 16x8 (128 processors) for GTC.

3.2 Predictive Model Validation

In this section, we present the normalized (per cycle) performance counters derived for each application and for each performance component (runtime, system power, CPU power, and memory power). Figure 1 shows an overview of the counters derived from our performance-tuned supervised principal components method. The figure shows each performance counter event rate used and the corresponding regression coefficient, β_1. The regression coefficients utilized in this work are represented up to 5 significant digits based on the precision of the performance counter measurements obtained using PAPI. These counters provided for the best performance models according to their correlation to each performance component and application, principal component coefficients, and restrictions placed on our detailed multivariate linear regression models.

In this work, we placed restrictions on our models to ensure that the models reflected realistic performance trends for each application. Different performance counter events are utilized to represent each application and modeling of each component. The reason for this method is to ensure that the application trends are correctly represented through appropriate counters that strongly represent the performance characteristics of each application.

3.2.1 Runtime Performance Overview

Our method highlights that the per cycle event rates from PAPI_TOT_INS and PAPI_L2_TCH affect runtime performance across all four applications. For SP-MZ, LU-MZ, and GTC, optimizations made to increase the number of L2 cache hits per cycle could potentially improve performance. BT-MZ and LU-MZ models indicate that improvements made to L1 cache utilization (through Cache_FLD_per_instruction) should improve

	Time		System Power		CPU Power		Memory Power	
BT-MZ	Cache_FLD	-1.611	PAPI_L2_TCH	-1.6769	PAPI_L1_TCM	3.5432	PAPI_L1_TCA	0.0763
	PAPI_TOT_INS	0.0967	PAPI_L2_TCA	1.5967	PAPI_L2_TCH	-3.9389	PAPI_L1_DCM	4.0496
	PAPI_L2_TCH	0.2992	PAPI_RES_STL	0.0803	PAPI_RES_STL	0.3967	PAPI_L2_TCH	-1.9443
	PAPI_L2_TCA	1.2152					PAPI_L2_TCA	2.1806
SP-MZ	PAPI_TOT_INS	0.1818	PAPI_L1_ICA	0.355	LD_ST_stall	0.1917	Cache_FLD	0.4563
	PAPI_L1_TCA	0.0744	PAPI_L2_TCH	-1.3452	PAPI_L1_TCM	1.5008	LD_ST_stall	0.0192
	PAPI_L2_TCH	-1.2834	PAPI_L1_TCM	0.9911	PAPI_L2_TCH	-1.6914	PAPI_L2_TCH	-3.5895
	PAPI_L1_TCM	1.1761					PAPI_L2_TCA	3.1151
LU-MZ	Cache_FLD	-0.0006	LD_ST_stall	0.0166	LD_ST_stall	0.0869	PAPI_L1_TCA	0.27923
	PAPI_TOT_INS	0.0011	PAPI_L2_TCH	-0.9886	PAPI_L2_TCH	-8.0003	PAPI_L2_TCH	-3.9574
	PAPI_TLB_DM	3.9085	PAPI_L2_TCA	1.0411	PAPI_L2_TCA	7.9137	PAPI_RES_STL	-0.29141
	PAPI_L2_TCH	-0.0591	PAPI_RES_STL	0.025				
GTC	PAPI_TOT_INS	0.0006	PAPI_RES_STL	1.5689	PAPI_RES_STL	0.9261	PAPI_TOT_IN	0.169617
	PAPI_L2_TCH	-1.8976	PAPI_L2_TCH	-3.2505	PAPI_TOT_IN	0.2663	PAPI_L2_TCH	-2.881
	PAPI_L2_TCA	1.9351	PAPI_L1_TCA	1.6916	PAPI_L1_TCA	0.0816	PAPI_L2_ICM	2.7119
	PAPI_BR_INS	-0.0381			PAPI_L2_TCH	-1.2640		

Fig. 1 Overview of Performance Counters Used for Predictive Models with the Corresponding Regression Coefficients

runtime, while increasing the number of L2 cache accesses per cycle could increase runtime.

3.2.2 System Power Performance Overview

BT-MZ, LU-MZ, and GTC use the event rates from PAPI_L2_TCH and PAPI_RES_ STL for predicting system power. For all applications, increasing the L2 cache hits per cycle can aid in reducing the system power consumption of each application. BT-MZ, LU-MZ and GTC utilize MPI/OpenMP with an increased MPI communication component. To accurately model their system power consumption the number of system stalls on any resource must be used to provide an accurate model. Since SP-MZ has a smaller variation in the communication message sizes and utilizes less than 160KB size messages, PAPI_RES_STL is not needed.

3.2.3 CPU Power Performance Overview

Improvements made to L1 and L2 cache utilization can potentially result in lower CPU power consumption across all applications. Improvements made to increase the number of L2 cache hits per cycle can potentially reduce the application's CPU power. Each application needs improvements in different aspects with regards to L1 cache performance. The models for both BT-MZ and SP-MZ demonstrate that improvements made to reduce the number of L1 cache misses can aid in decreasing the CPU power consumption. LD_ST_stall_per _cycle in SP-MZ and LU-MZ causes CPU power consumption to increase.

3.2.4 Memory Power Prediction Overview

PAPI_L2_TCH is used in all applications in predicting the memory power consumption. Importantly, improving the number of L2 cache hits per cycle will aid in reducing the memory power. Better utilization of the L2 cache will result in a decrease in the workload of the memory component. For all of the NPB-MZ applications, L1 cache performance affects the memory power consumption. For BT-MZ and LU-MZ, PAPI_L1_TCA affects memory performance by indicating that as the number of L1 cache accesses per cycle increases memory power consumption will also increase. SP-MZ utilizes Cache_FLD_per_cycle to capture activity with regards to L1 cache reads, writes, hits, and misses.

3.3 Application-Centric Analysis

In this section, we identify the characteristics of each application that affect the predictive models in terms of runtime, system power, CPU power, and memory power. The goal is to outline the differences that each application exhibits with respect to the other Hybrid applications utilized in this work.

3.3.1 BT-MZ

BT-MZ has an average error of less than 3%. Figure 2 provides an overall depiction of the errors for each modeling component for BT-MZ. Four performance counters from our performance-tuned principal component analysis method were derived to model runtime. Improving the L1 cache utilization of the application on SystemG has the potential to reduce the runtime of the

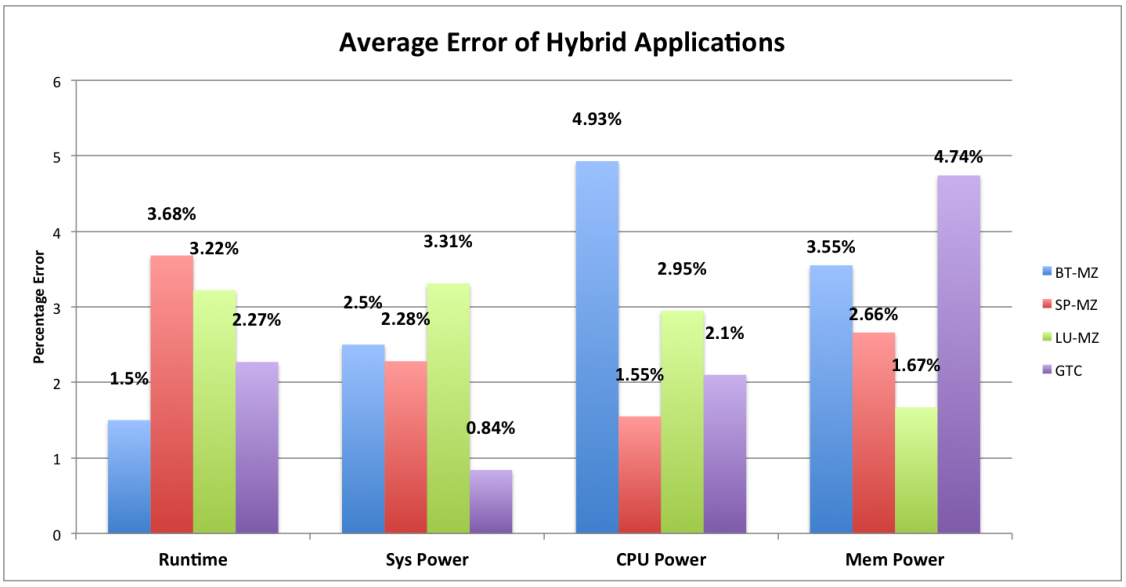

Fig. 2 Average Error of Predictive Models for Four Hybrid Scientific Applications

application. The Cache_FLD_per_instruction event rate has the largest regression coefficient in this model in terms of magnitude (-1.611). Cache utilization for BT-MZ places a larger emphasis on L1 utilization as activity in the L2 cache is limited for the class C benchmark. The BT-MZ model indicates that improvements to L1 cache performance may improve runtime.

Our system power model shows that increasing the L2 cache hits per cycle provides for a strong optimization method for reducing the system power consumption. The PAPI_L2_TCH has the largest regression coefficient in terms of magnitude (-1.6769) for system power. Our model shows that increasing the number of cache hits per cycle can possibly reduce system power.

The CPU model shows that L2 cache hits per cycle has the largest regression coefficient in terms of magnitude (-3.9389). Improvements made to increase the L2 cache hits per cycle can potentially lower the power consumption of memory. PAPI_RES_STL has a large number of reported counts per cycle and the regression coefficient is not large in comparison to the other counter event rates. The product of the regression coefficient for PAPI_RES_STL and the reported event count per cycle contribute greatly to the CPU power consumption.

Prediction for memory power consumption has an average error of less than 3.6%. The predictive model shows that the PAPI_L1_DCM event has the largest regression coefficient. Improvements made to the utilization of the L2 cache are likely to affect the power consumption of the memory as it has a negative regression coefficients for PAPI_L2_TCH.

3.3.2 SP-MZ

The SP-MZ benchmark has an average error of less than 3% for all modeling components. Insight provided from the modeling of the runtime shows that optimizations made to improve L2 total cache hits can be used to reduce the runtime of the application. Consequently, poor utilization of the L1 cache with regards to total cache misses per cycle may have a larger affect on the runtime as the PAPI_L1_TCM event rate carries a larger regression coefficient (1.1761). The system power model utilizes three counter events and indicates that improvements made to L2 cache utilization by improving total cache hits can decrease system power consumption. The regression coefficient for PAPI_L2_TCH has the largest magnitude (-1.3452) for the system power consumption model. Poor utilization of the L1 cache results in an increase in power consumption as demonstrated through the model from the L1 instruction cache access counter and L1 total cache misses counter event.

The CPU power model has an average error of less than 1.6% for SP-MZ. The CPU regression model is based upon LD_ST_stall_per_cycle, PAPI_L1_TCM, and PAPI_L2_TCH. The model shows that improving the total L2 cache hits provides for the best method for reducing the CPU power consumption of the application. The PAPI_L2_TCH has the largest regression coefficient, -1.6914, and serves as the largest contributor to the power model.

The memory model highlights the large affect that L2 cache performance has on memory power consumption. Specifically, the L2 total cache hits per cycle largely

affect memory power consumption. Our model shows that increasing the number of L2 total cache hits per cycle could help to significantly reduce the memory power consumption. Improving the L2 cache performance of the application will reduce memory activity, which will lower the power consumption. Our predictive models for SP-MZ show that improving L2 cache performance provides a strong opportunity for improving performance in terms of runtime and component power consumption. For each component, the PAPI_L2_TCH event had the highest regression coefficient.

3.3.3 LU-MZ

Modeling of the LU-MZ benchmark has an average error of less than 3.3%. Insight provided from the runtime model indicates that improvements made to reduce TLB misses per cycle and increase L2 total cache hits per cycle can possibly reduce the runtime. On average, the TLB data misses per cycle in LU-MZ are 5 times higher than the TLB data misses per cycle in BT-MZ and SP-MZ. In addition, the average number of L2 total cache hits per cycle and the regression coefficient is the second largest contributor to the runtime model.

For LU-MZ, our model for system power consumption utilizes four performance counters and shows that improvements made to utilization of the L2 cache by reducing the total cache misses can decrease system power consumption. The largest contributor to the model is from PAPI_TOT_INS event.

The CPU power model is based upon three performance counters and indicates that improving the total L2 cache hits provides for the best method for reducing the CPU power consumption of the application. The PAPI_L2_TCH has the largest regression coefficient, -0.9886. Reducing the total number of L2 cache accesses will also reduce the power consumption of the CPU as the PAPI_L2_TCA event rate is the largest contributor to the CPU power consumption. Modeling the CPU power consumption for LU-MZ utilizes three performance counters. The CPU power model shows the strong correlation between L2 cache activity and CPU power consumption. PAPI_L2_TCH, and PAPI_L2_TCA have the largest regression coefficients at -8.0003 and 7.9137. Improvements made to the L2 total cache hits will result in reduction in the CPU power consumption for the LU-MZ application. The memory power model shows that activity on the L1 and L2 cache can be used to measure and predict power consumption for the memory on the system. Specifically, the L2 total cache hits counter has the largest regression coefficient, -3.9574. L1 total cache accesses is the strongest contribution to the memory model because of the large number of L1 cache accesses that occur per cycle.

3.3.4 Gyrokinetic Toroidal Code (GTC)

The GTC application has an average error of less than 2.8% for all performance component models. To model the runtime for GTC we utilize four performance counters, indicated in Figure 1. Insight provided from the modeling of the runtime shows that optimizations made to improve L2 cache performance will play a significant roll in reducing the runtime of this application. The value from the product of L2 total cache hits per cycle and the regression coefficient is the second largest contributor to the runtime model.

Modeling of system power for GTC indicates L1 cache activity affects system power greatly. The event rate captured from PAPI_L1_TCA, L1 total cache accesses, is the largest contributor to the total system power. The second largest contributor to the system power model is PAPI_RES_STL. Reducing the number of stalls on any system resource can be used to reduce performance and system power consumption. PAPI_L2_TCH has the largest regression coefficient in terms of magnitude, -3.25050068, however it contributes the least to the system power model.

Memory power is predicted using PAPI_TOT_INS, PAPI_L2_TCH, and PAPI_L2_ICM. The model shows that the instructions per cycle (through PAPI_TOT_INS) correlates strongly with memory power consumption for GTC. PAPI_L2_TCH can be increased to reduce memory power consumption as it has the largest regression coefficient in terms of magnitude, -2.8805.

4 Related Work

In [3] performance counters are used to provide power measurements of the complete system. The average error was less than 9%. This work provided a system-centered approach to modeling based on the correlation of performance counter events to applications.

In [20] a surrogate estimation model using performance counters is presented on an Intel Core i7 system to estimate for CPU, Memory, and the total system power for OpenMP benchmarks up to 8 threads. The median error was 5.32% on the system. In [20] various Intel Core i7 specific counters that were representative of the system features were utilized. For example, this work used counters that represented the number of unhalted cycles in the CPU and retired instructions for building the CPU power model. The robust regression model was built based on the spearman correlation. In contrast, our work seeks to understand how

the application characteristics, such as how L1 and L2 cache utilization, affect the power consumption by using application-centric performance counters available in PAPI and perfmon. In [22] power estimations using counters are presented with median errors of 5.63%.

Our work differs from previous works using performance counters in that we seek to identify areas within each Hybrid application that should be optimized to improve performance. We determine which counters most influence the performance of each application. We also apply our methodology to Hybrid (MPI/OpenMP) applications to exploit the hierarchical nature of multicore systems. In addition, we identify performance counters that can provide detailed measurements of the hierarchical characteristics of the system with regards to utilization of the L1 and L2 cache.

5 Conclusions

In this paper, we present predictive performance models to analyze the performance characteristics of Hybrid scientific applications in terms of runtime, system power, CPU power, and memory power. The predictive models are able to determine the performance characteristics that affect each performance component. Most importantly, our method identifies the different performance counter measurement that are needed to accurately predict application performance and provide insight to improve performance for each application. Our models make use of the Multicore Application Modeling Infrastructure, MuMI, which utilizes Prophesy, PowerPack, and PAPI to provide systematic measurement, and modeling of power consumption and performance-power tradeoffs on multicore systems. Our predictive models are 97% accurate across four Hybrid scientific applications for up to 128 processors and can be used to obtain insight into improving applications for better performance on multicore systems.

Future work will use our models to predict performance for different problems sizes and frequency settings. We will also test our modeling insights to improve performance by reducing runtime and decreasing power consumption. We will apply our methodology to different applications and parallel programming models.

Acknowledgements This work is supported by NSF grants CNS-0911023, CNS-0910899, CNS-0910784, CNS-0905187. The authors would like to acknowledge Stephane Ethier from Princeton Plasma Physics Laboratory for providing the GTC code.

References

1. E. Bair, et. al. , Prediction by Supervised Principal Components, Journal of the American Statistical Ass., 2006.
2. F. Bellosa, The Benefits of Event-Driven Energy Accounting in Power-Sensitive Systems. ACM SIGOPS Euro. Workshop, September 2000.
3. W. Lloyd Bircher, Lizy K. John, Complete System Power Estimation: A Trickle-Down Approach Based on Performance Events, In Proc. of ISPASS'2007. pp.158 168.
4. W. Lloyd Bircher, et. al., Runtime Identification of Microprocessor Energy Saving Opportunities, In Proc. of the Int. Sym. on Low Power Elec. and Design, August 2005.
5. Zhenwei Cao, David R. Easterling, Layne T. Watson, Dong Lia; Kirk W. Cameron; Wu-Chun Feng,Power saving experiments for large-scale global optimisation, In the International Journal of Parallel, Emergent and Distributed Systems, Volume 25, Issue 5 October 2010 , pages 381 - 400.
6. M. Curtis-Maury, et al., Online Power-Performance Adaptation of Multithreaded Programs using Hardware Event-Based Prediction (ICS06), 2006.
7. M. Curtis-Maury, et. al., Prediction-Based Power-Performance Adaptation of Multithreaded Scientific Codes, (TPDS), Vol.19, No.10, 2008.
8. V. Freeh, et. al., Just-in-time Dynamic Voltage Scaling: Exploiting inter-node slack to save energy in MPI Programs, Journal of Parallel and Dist. Comp. Vol. 68 Issue 9, Sept. 2008.
9. V. Freeh, Feng Pan, D. Lowenthal, and N. Kappiah, Using Multiple Energy Gears in MPI Programs on a Power-Scalable Cluster, PPOPP05, 2005.
10. C-H. Hsu, and W-C Feng, A Power-Aware Run-Time System for High-Performance Computing, IEEE/ACM Supercomputing 2005 (SC05), November 2005.
11. R. Ge, et.al., PowerPack: Energy Profiling and Analysis of High-Performance Systems and Applications, IEEE Trans. Parallel Distrib. Syst. 21(5): 658-671, 2010.
12. H. Jin et. al., Performance Characteristics of the Multi-Zone NAS Parallel Benchmarks, (IPDPS), 2004.
13. K. Lee and Kevin Skadron, Using Performance Counters for Runtime Temperature Sensing in High Performance Processors, (HPPAC05), April 2005.
14. D. Li, et. al., Hybrid MPI/OpenMP Power-Aware Computing, IPDPS2010, Atlanta, Georgia, April 2010.
15. T. Li, et. al., Run-Time Modeling and Estimation of Operating System Power Consumption, Sigmetrics2003, 2003.
16. C. Lively, V. Taylor, et. al., A Methodology for Developing High Fidelity Communication Models on Multicore Systems, SBAC-PAD 2008: 55-62.
17. Multicore application Modeling Infrastructure (MuMI) project, http://www.mumi-tool.org.
18. P. M. Kogge, ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems, Univ. of Notre Dame, CSE Dept. Tech. Report TR-2008-13, Sept. 28, 2008.
19. Performance Application Programming Interface, papi, http://icl.cs.utk.edu/papi/
20. M. Lim, A. Porterfield, and R. Fowler, SoftPower: fine-grain power estimations using performance counters, (HPDC10), New York, NY, 2010.
21. B. Rountree, et. al. , Adagio: making DVS practical for complex HPC Applications, (ICS09), NY, 2009.
22. K. Singh, M. Bhadhauria, and S. A. McKee, Real Time Power Estimation and Thread Scheduling via Performance Counters, Proc. of Workshop on Design, Architecture, and Simulation of Chip Multi-Processors, November 2008.

23. S. Song, et. al., Energy Profiling and Analysis of the HPC Challenge Benchmarks, Int. Journal of High Perf. Computing Applications, Vol. 23, No. 3, (2009).
24. S. Song, et. al., Iso-energy-efficiency: An approach to power-constrained parallel computation, (IPDPS), 2011.
25. V. Taylor, X. Wu, and R.Stevens, Prophesy: An Infrastructure for Performance Analysis and Modeling System of Parallel and Grid Applications, ACM SIGMETRICS Perf. Evaluation Review, Vol. 30, Issue 4, March 2003.
26. X. Wu, V. Taylor, et. al., Performance Analysis, Modeling and Prediction of a Parallel Multiblock Lattice Boltzmann Application Using Prophesy System, ICCC06, 2006.
27. X. Wu, V. Taylor, C. Lively et al, Performance Analysis and Optimization of Parallel Scientific Applications on CMP Clusters, Scalable Computing: Practice and Experience, Vol. 10, No. 1, 2009.
28. X. Wu and V. Taylor, Performance Characteristics of Hybrid MPI/OpenMP Implementations of NAS Parallel Benchmarks SP and BT on Large-scale Multicore Supercomputers, ACM SIGMETRICS Perf Evaluation Review, Vol. 38, Issue 4, 2011.