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

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

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

Received: date / Accepted: date

Abstract In this paper, we present the Energy-Aware Modeling and Optimization Methodology (E-AMOM) framework, which develops models of runtime and power consumption based upon performance counters and uses these models to identify energy-based optimizations for scientific applications. E-AMOM utilizes predictive models to employ run-time Dynamic Voltage and Frequency Scaling (DVFS) and Dynamic Concurrency Throttling (DCT) to reduce power consumption of the scientific applications, and uses cache optimizations to further reduce runtime and energy consumption of the applications. The models and optimization are done at the level of the kernels that comprise the application. Our models resulted in an average error rate of at most 6.79% for Hybrid MPI/OpenMP and MPI implementations of six scientific applications. With respect to optimizations, we were able to reduce the energy consumption by up to 21%, with a reduction in runtime by up to 14.15%, and a reduction in power consumption by up to 12.50%.

Keywords Performance modeling, Energy Consumption, Power Consumption, MPI, Hybrid MPI/OpenMP, Power prediction, performance optimization.

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

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

Shirley Moore
Dept. of Computer Science,University of Texas at El Paso
E-mail: svmoore@utep.edu

Dan Terpstra
Innovative Computing Lab., University of Tennessee
E-mail: terpstra@icl.utk.edu

1 Introduction

Currently, an important research topic in high-performance computing is that of reducing the power consumption of scientific applications on high-end parallel systems (e.g., teraflop and petaflop systems) without significant increases in runtime performance [2,3,5–7,9–11,13–16, 18–20,23]. Performance models can be used to provide insight into an applications performance characteristics that significantly impact runtime and power consumption. As HPC systems become more complex, it is important to understand the relationships between performance and power consumption and the characteristics of scientific applications. In this paper, we present E-AMOM, an Energy-Aware Modeling and Optimization Methodology for developing performance and power models and reducing energy. In particular, E-AMOM develops application-centric models of the runtime, CPU power, system power, and memory power as a function of performance counters. The models are used to identify ways to reduce energy.

E-AMOM is useful to HPC users in the following ways:

* To obtain the necessary application performance characteristics at the kernel level to determine application bottlenecks on a given system with regard to execution time and power consumptions for the system, CPU, and memory components.
* To improve performance of the application at the kernel level with regard to applying DVFS [13] and DCT [5] to reduce power consumption and making algorithmic changes to improve energy consumption.
* To provide performance predictions (about execution time and power requirements) for scheduling

methods used with systems having a fixed power budget.

The preliminary foundation of this work was presented in [18]; the preliminary work, however, did not include any optimization and the models were at the level of the application, not the kernel. Further, frequency and input sizes were not included in the earlier models. This paper builds upon our previous work to make the following major contributions:

1) Using the performance-tuned principal component analysis (PCA) method, we develop accurate performance models of Hybrid (MPI/OpenMP) and MPI implementations of six scientific applications at the kernel level. Our models are able to accurately predict runtime and power consumptions of the system, CPU, and memory components across different numbers of cores, frequency settings, concurrency settings, and application inputs. For the Hybrid (MPI/OpenMP) and MPI implementations of six scientific applications, the average error rate was at most 6.79%.
2) The models are used to determine appropriate frequency and concurrency settings for application kernels to reduce power consumption. The kernel models are also used to improve runtime through loop blocking and loop unrolling.
3) Our combined optimization strategy, developed in E-AMOM, is able to reduce energy consumption of Hybrid and MPI scientific applications by up to 21%, with a reduction in runtime by up to 14.15%, and a reduction in power consumption by up to 12.50% on multicore systems.

The remainder of this paper is organized in the following manner: In Section 2, we give pertinent background related to the E-AMOM framework. Section 3 discusses the details of the modeling component of E-AMOM. Section 4 provides the methodology for optimization of scientific applications using E-AMOM. Section 5 presents detailed experimental results obtained for the modeling and optimization experiments. Section 6 discusses some related work, and Section 7 concludes the work and discusses some future work.

2 Background

E-AMOM is the modeling and optimization component of MuMMI (Multiple Metrics Modeling Infrastructure) [27]. MuMMI facilitates systematic measurement, modeling, and prediction of performance, power consumption and performance-power trade-offs for multicore systems. Figure 1 provides an overview of the

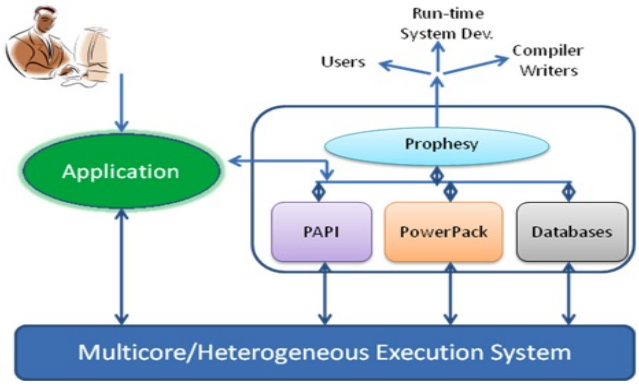

Fig. 1 Multiple Metrics Modeling Infrastructure (MuMMI)

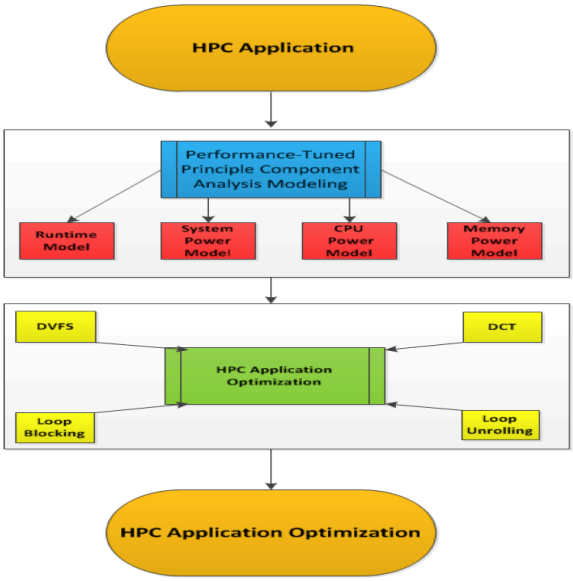

Fig. 2 E-AMOM Framework

MuMMI framework, which builds upon three existing frameworks: Prophesy[25], PowerPack[9] and PAPI [21] E-AMOM utilizes Prophesy's instrumentation framework [27], PowerPack for collecting power profiles, and PAPI for collecting performance counter data.

Figure 2 provides the high level view of E-AMOM, which makes use of a performance-tuned principal component analysis method to develop the models for runtime, system power, CPU power, and memory power. The models are developed for the kernels that comprise the application. Hence, these models identify the optimization strategies to be used at the kernel level. We focus on four optimization strategies, two for power consumption: DVFS and DCT, and two for execution time: loop blocking and loop unrolling.

Dynamic Voltage and Frequency Scaling (DVFS) [13] is a technique that is used to reduce the voltage and frequency of a CPU in order reduce power consumption. Applying DVFS is especially beneficial during a

period where communication slack time appears during parallel execution due to load imbalance between task communications. For simplicity, we assume that all cores that execute during the application phase will run at the same frequency.

Dynamic Concurrency Throttling (DCT) [5] is a technique that can be used to reduce the number of threads used to execute an application. Applying DCT is especially beneficial during OpenMP performance phases that do not benefit from using the maximum number of OpenMP threads per node.

Much of the computation involved in the kernels of HPC applications occurs within nested loops. Therefore, loop optimization is fundamentally important for such applications. It is for this reason that E-AMOM utilizes loop blocking and loop unrolling with the optimization component.

3 Modeling Methodology

In this section we present the performance-tuned principal component analysis method, which is used for the modeling component of E-AMOM. During an application execution we capture 40 performance counters utilizing PAPI and the perfmon performance library. For the given execution, all of the performance counters are normalized by the total cycles of execution to create performance event rates for each counter. In addition, we restrict the models to have non-negative regression coefficients to ensure that the models represent actual performance scenarios. Negative coefficients indicate a reduction in power with some system activity, which is not representative of reality; all system activity requires some power. A multivariate linear regression model is constructed for each performance component (execution time, system power, CPU power, and memory power), for each kernel in an application.

The most difficult part in predicting application performance via performance counters is determining the appropriate counters to use for modeling each performance component. The following algorithm, consisting of six steps, is used to identify the appropriate performance counters and the model development. We utilize the hybrid NPB BT-MZ with Class C to illustrate the method for modeling the runtime of the application. We focus on the full BT-MZ application with respect to the results for most steps. The training set used is consistent with the training set description given in Section 5 on experimental results.

Step 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).*

Table 1 Correlation Coefficients for BT-MZ in Step 2

Counter	Correlation Value
PAPI_TOT_INS	0.9187018
PAPI_FP_OPS	0.9105984
PAPI_L1_TCA	0.9017512
PAPI_L1_DCM	0.8718455
PAPI_L2_TCM	0.8123510
PAPI_L2_TCA	0.8021892
Cache_FLD	0.7511682
PAPI_TLB_DM	0.6218268
PAPI_L1_ICA	0.6487321
Bytes_out	0.6187535

Equation 1 defines how the Spearman correlation coefficient is computed for identifying the rank between each performance counter and each performance component. The x_i and y_i variables represent the ranks of the performance counter and performance component (time, system power, CPU power, memory power) that are being correlated. The variables $\overline{x}$ and $\overline{y}$ represent the average of the samples for each variable. The Spearman rank correlation provides a value between -1 and 1 for ρ.

$$\rho = \frac{\sum_i (x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_i (x_i - \overline{x})^2 \sum_i (y_i - \overline{y})^2}} \quad (1)$$

The Spearmans rank correlation coefficient is used because it provides a correlation value that is not easily influenced by outliers in the dataset.

For the BT-MZ benchmark, we started with 40 counters. We do not give the coefficients for all 40 counters since many of the counters were eliminated based upon the Step 2.

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

The value α_{ai} is used to determine an appropriate threshold for eliminating performance counters with low correlation to the performance event that is to be modeled. The value for α_{ai} is established based on a cluster analysis using Gaussian mixture distribution. The Gaussian clustering analysis enables clusters to be determined based on the multivariate normal components of the representative points. Table 1 provides the results of this step for BT-MZ. The value α_{ai} was determined to be 0.55.

Step 3. *Compute a multivariate linear regression model based upon the remaining performance counter event rates. Recall that we restrict the coefficients to be non-negative.*

Step 4. *Establish a new threshold, α_{bi}, and eliminate performance counters with regression coefficients smaller than the selected threshold.*

Table 2 Regression Coefficients for BT-MZ in Step 3

Counter	Regression Coefficient
PAPI_TOT_INS	1.984986
PAPI_FP_OPS	1.498156
PAPI_L1_DCM	0.9017512
PAPI_L1_TCA	0.465165
PAPI_L2_TCA	0.0989485
PAPI_L2_TCM	0.0324981
Cache_FLD	0.026154
PAPI_TLB_DM	0.0000268
PAPI_L1_ICA	0.0000021
Bytes_out	0.000009

Table 3 Regression Coefficients for BT-MZ in Step 4

Counter	Regression Coefficient
PAPI_TOT_INS	1.984986
PAPI_FP_OPS	1.498156
PAPI_L1_DCM	0.9017512
PAPI_L1_TCA	0.465165
PAPI_L2_TCA	0.0989485
PAPI_L2_TCM	0.0324981
Cache_FLD	0.026154

The value α_{bi} serves as the second elimination threshold and has a similar purpose as α_{ai} to eliminate performance counters that do not contribute substantially to modeling in the initial multivariate linear regression model. The determination of α_{bi} is accomplished via the same method as given in Step 2 but applied to the regression coefficients. Table 2 provides the results of the Step 3. An appropriate value for α_{bi} is important because if the value is not correctly chosen, then values needed in the modeling will be eliminated. The value determined for BT-MZ was 0.02. Table 3 provides the results of Step 4.

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

Principal component analysis is a statistical method that is used to provide a linear transformation of data that can be used to reduce the dimensionality of data[1]. The principal components of data, Y_i, is given by a linear combination of variables $X_1, X_2, \ldots, X_p$. For example, the first principal components would be represented by Equation 2

$$Y_1 = a_{11}X_1 + a_{12}X_2 + ... + a_{1p}X_p \tag{2}$$

The values for $a_{11}, a_{12}, \ldots, a_{1p}$ are calculated with the constraint that the sum of their squares must equal to 1 as follows.

$$a_{11}^2 + a_{12}^2 + ... + a_{1p}^2 = 1 \tag{3}$$

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

$$Y_2 = a_{21}X_1 + a_{22}X_2 + ... + a_{2p}X_p \tag{4}$$

The number of principal components calculated is dependent upon the number of variables included in the original data. In our work, the number of principal components would be equal to the number of performance counters resulting from Step 4. PCA is used to identify the final performance counters to use. Table 4 provides the results for BT-MZ.

Table 4 Counter Reduction after Step 5 for BT-MZ

Preliminary Counters	Final Counters
PAPI_TOT_INS	PAPI_TOT_INS
PAPI_FP_OPS	
PAPI_L1_DCM	
PAPI_L1_TCA	
PAPI_L2_TCA	PAPI_L2_TCA
PAPI_L2_TCM	PAPI_L2_TCM
Cache_FLD	

The first two principal components represent the largest amount of variability, or information, in our data. Therefore, the first two principal components are used for further reducing the number of variables for creating our multivariate linear regression model. Using the vectors resulting from the first two principal components, the variables with the highest coefficients serve as the most accurate predictors for modeling.

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

The final step entails using the performance counters (with the highest coefficients as mentioned previously) along with a term to represent frequency to build the model for the desired performance component (runtime, system power, CPU power, and memory power). Table 5 presents the regression coefficients for the runtime model for BT-MZ. Equation 5 provides the general multivariate linear regression equation that is used for developing the model, where $r_i (i = 1, \ldots, n)$ represents the respective performance counter event rate for the counter i, and $\beta_i (i = 1, \ldots, n)$ is the regression coefficient for the counter i.

$$y = \beta_0 + \beta_1 * r_1 + ... + \beta_n * r_n \tag{5}$$

4 Optimization Methodology

In this section we discuss the methods that are used to improve the performance of scientific applications

Table 5 Final Regression Coefficients for BT-MZ

Counter	Regression Coefficient
Frequency	0.00476
PAPI_TOT_INS	0.105050
PAPI_L2_TCA	0.097108
PAPI_L2_TCM	0.178700

with respect to system power consumption and runtime since this is the focus of this paper. Details about the optimization scheme for CPU power and memory power can be found in [17]. Our optimization methods include DVFS, DCT, and improvements to the utilization of the memory subsystem via loop unrolling and loop blocking.

Recall that performance models are developed for each kernel in an application. Each optimization method is considered at the kernel level as well. When evaluating if an optimization method should be applied to a given kernel, we evaluate the performance of the full application. Equation 6 represents the relationship for each kernel to the full scientific application:

$$P_{total} = \sum_{i=0}^{n-1} K_i \tag{6}$$

where P represents the performance of the application and K_i represents the performance of kernel i.

We consider optimizations that can be used to reduce runtime and power separately because total energy consumption of an application is represented as the product of average power consumption and runtimes.

4.1 Applying DVFS and DCT

In considering DCT, we adjust the configuration of each hybrid application kernel by dynamically configuring the number of OpenMP threads used. With respect to DVFS, we lower the CPU frequency for running the kernels to reduce power consumption. To identify a good optimization in terms of DCT and DVFS, we use the following steps:

1) Determine the appropriate configuration settings
   a) DVFS Setting
      i) Compute expected power consumption and execution time at lower CPU frequency.
      ii) If frequency setting results in a 10% saving in power consumption, without increasing runtime more than 3%, then use the reduced frequency.
   b) DCT Setting
      i) Compute expected power consumption and execution time at concurrency settings using 1, 2, 4, and 6 threads.
      ii) Identify the concurrency setting that enables the best improvement in power consumption and runtime.
2) Determine the total application runtime and system power consumption including synchronization overhead costs mu_i from changing application settings.
3) Use new configuration settings for running the application.

Equation 7 represents the expected execution time for each kernel and the synchronization overhead costs that would be incurred from lowering and increasing the CPU frequency for running the kernel i in the HPC application:

$$P_{total\_optimized} = \sum_{i=0}^{n-1} (K_i + \mu_i) \tag{7}$$

We utilize the multivariate linear regression equation presented in Equation 5 to determine the appropriate configuration based on frequency and number of threads. The frequency, number of nodes, and threads per node, are incorporated into the regression equation with the performance counters to predict the performance of the application kernel at two CPU frequency settings (2.4 Ghz and 2.8 Ghz) and at concurrency settings of 1, 2, 4, 6, and 8 threads.

The performance for each kernel is predicted to determine if a 10% saving in power can be achieved without increasing the runtime more than 3%. If a kernel is not able to achieve 10% reduction in power with no more than 3% runtime increase then the original configuration setting for that kernel is used. A decrease in power consumption greater than 10% provides for a measurable improvement that cannot be attributed to error or system noise. Also, a runtime increase less than 3% does not largely affect the runtime of the application, which means that a reduction in power can be obtained without a large increase in runtime.

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

$$P_{sys\_power} = \frac{\sum_{i=0}^{n-1} K_{sysp\_i}}{n-1} \tag{8}$$

where K_{sysp_i} represents the predicted system power consumption of kernel i and n is the number of kernels in the application. We have determined the following scenarios in which it would be appropriate to apply changes to configuration settings in our applications:

1) DVFS and DCT changes to specific kernels in the application

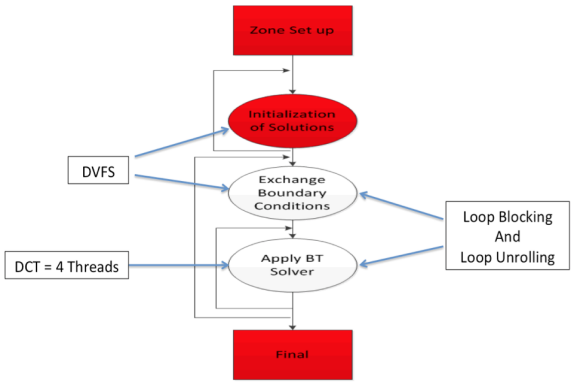

Fig. 3 NPB BT-MZ Hybrid Application Optimization

2) DVFS-only applied to specific kernels in the application
3) DCT-only applied to specific kernels in the application
4) DVFS applied to a small number of time-steps within an application.

4.2 Loop Optimizations

The optimal loop block size varies with different applications on different systems. In this work, we apply the following loop block sizes: 2x2, 4x4, 8x8 and 16x16 to our HPC applications to measure which loop block size is optimal. To determine the best block size for each application we measure the performance of the application for each block size using a reduced number of iterations to approximate the best block size. Previous work [29] has identified these block sizes as optimal sizes for achieving performance improvements within scientific applications. Outer loop unrolling can increase computational intensity and minimize load/stores, while inner loop unrolling can reduce data dependency and eliminate intermediate loads and stores. For most application, loops were unrolled 4 times dependent upon the number of iterations for each loop.

In considering the different configurations for the loop optimization, the focus is on the runtime. The selected loop optimization configuration is the one resulting in the best runtime. Figure 3 provides an illustration of the type of optimization that was applied to the hybrid NPB BT-MZ application. The figure illustrates the application of DVFS to the initialization of solutions and exchange of boundary conditions kernels. DCT is applied to the BT solver kernels and loop blocking and loop unrolling are applied to the exchange of boundary conditions and BT solver kernels.

5 Experimental Results

In this work, we use a power-aware cluster SystemG [24] to conduct our experiments. SystemG , available at Virginia Tech, is a 22.8 TFLOPS research platform that utilizes 325 Mac Pro computer nodes. Each node of SystemG contains 30 thermal sensors and more than 30 power sensors. Each node in the system has two quad-core 2.8GHz Intel Xeon 5400 series processor, with 8GB memory. Note that, in this paper, MxN stands for the number of nodes M with the number of cores N used per node.

The training of our models is based on performance data obtained for each application at different processor sizes for predicting intra-node and inter-node performance. The training set consists of 12 different system configurations for each application; each system configuration involves the use of one data set for each of the applications. The 12 training set configurations focus on intra-node performance (1x1, 1x2, 1x3 executed at 2.8Ghz and 1x3, 1x4, and 1x6 executed at 2.4 Ghz) and inter-node performance (1x8, 3x8, 5x8 at 2.8 Ghz and 7x8, 9x8, 10x8 at 2.4 Ghz).

Using the training set we construct a multivariate linear regression model for each application. This model is then used to predict 36 different system configurations (not included in the training set) for each application. The 36 different configurations included points that were outside of the training set including 2x8, 4x8, 6x8, 8x8, 9x8, 10x8, 11x8, 12x8, 13x8, 14x8, 15x8, and 16x8, which were executed at frequencies using 2.8Ghz and 2.4Ghz for different datasets for the applications. In Table 6 we present the scientific applications that are used throughout this paper. These applications include the three NAS Multi-Zone Parallel Benchmarks 3.3 [12] and three large-scale scientific applications: GTC [8], PMLB [28] and Parallel EQdyna [26] We consider the MPI and hybrid (MPI/OpenMP) versions of each application.

5.1 Modeling Results

The accuracy of our models across the six hybrid and MPI applications given in Table 6 are illustrated in Figures 4 and 5. The prediction included different data sets as well as different system configurations.

Figure 4 presents the modeling accuracy across the hybrid implementations of our six large-scale scientific applications. In terms of runtime, the BT-MZ and EQdyna applications had the lowest prediction error in the range of 1.9%. For system power, the lowest prediction error occurred for the PMLB application, with an error of 0.84%. The SP-MZ application had the lowest

Table 6 Overview of HPC Applications

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

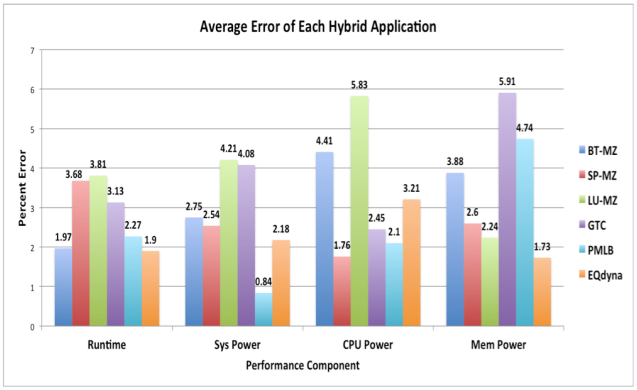

Fig. 4 Average Prediction Error Rates (%) for Hybrid Applications

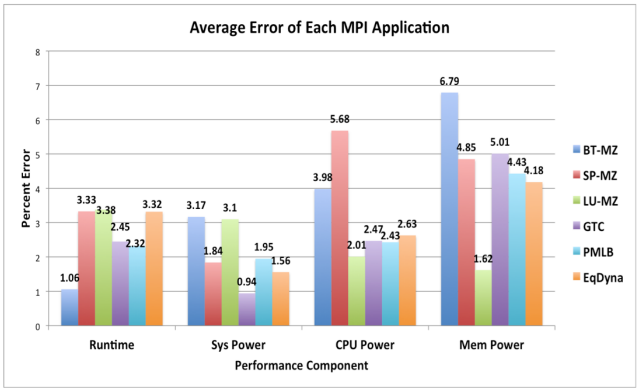

Fig. 5 Average Prediction Error Rates (%) for MPI Applications

prediction error for CPU power and EQdyna had the lowest prediction error for memory power (1.73%). Figure 5 presents the modeling accuracy across the MPI implementations of our six large-scale scientific applications. For runtime, the BT-MZ application has the lowest prediction error of 1.06%. For predicting system power, the GTC application had an error rate of 0.94%, which provides the lowest error across all applications and performance components for the MPI implementations. The LU-MZ application provides the lower error rate for both CPU power prediction (2.01%) and memory power prediction (1.62%). Overall, the prediction results indicate that the predictive models have the average error rate of up to 6.79% across six hybrid and MPI scientific applications on up to 128 processor cores and can be used to obtain insight into improving applications for better performance on multicore systems.

5.2 Optimization of Large-Scale Applications

In this section we present the optimization results obtained across four large-scale scientific applications. The optimizations were identified based upon the models as described in Section 4. Optimizations were applied to all six MPI and hybrid scientific application but four applications are presented due to space constraints. The four applications are representative of the results for all six applications.

5.2.1 BT-MZ

Applying DVFS and DCT to select application kernels to reduce power consumption and execution time for the application improves the performance of the hybrid NPB BT-MZ. DVFS is applied to the initialize solutions kernel, which sets up appropriate zones for execution, and the exchange of boundary conditions, which contains significant MPI communication. DCT is applied during the BT solver kernel reducing the power consumption during this phase for an optimal configuration using 4 threads.

Loop optimizations are applied to class D (block size = 4x4). Table 7 presents the performance results for the hybrid BT-MZ application with Class D, where Energy

Table 7 Performance Comparison of Hybrid BT-MZ with Class D

#Cores (MxN)	BT-MZ Type	Runtime (s)	Average Power (W)	Energy Per Node (KJ)
6x8	Hybrid	655	348.70	228.401
	Optimized -Hybrid	632 (-3.64%)	324.41 (-7.49%)	205.027 (-11.4%)
8x8	Hybrid	493	348.73	171.573
	Optimized -Hybrid	440 (-12%)	322.17 (-8.24%)	141.754 (-21.0%)
16x8	Hybrid	339	347.82	117.911
	Optimized -Hybrid	319 (-6.27%)	323.11 (-7.65%)	103.072 (-14.39%)
32x8	Hybrid	201	346.12	69.570
	Optimized -Hybrid	193 (-4.14%)	325.92 (-6.2%)	62.902 (-10.60%)
64x8	Hybrid	119	347.27	41.325
	Optimized -Hybrid	112 (-6.25%)	324.45 (-7.03%)	36.338 (-13.7%)

Table 8 Performance Comparison of MPI BT-MZ with Class D

#Cores (MxN)	BT-MZ Type	Runtime (s)	Average Power (W)	Energy Per Node (KJ)
6x8	MPI	729	330.732	241.104
	Optimized -MPI	700 (-4.14%)	323.81 (-2.14%)	226.667 (-6.36%)
8x8	MPI	545	327.014	178.223
	Optimized -MPI	489 (-11.45%)	320.17 (-2.14%)	156.563 (-13.83%)
16x8	MPI	387	329.12	117.911
	Optimized -MPI	329 (-14.15%)	318.19 (-3.44%)	104.685 (-12.63%)
32x8	MPI	233.14	328.33	76.547
	Optimized -MPI	220.78 (-5.59%)	310.38 (-5.72%)	68.526 (-11.70%)
64x8	MPI	138.58	327.45	44.396
	Optimized -MPI	125.73 (-10.22%)	307.83 (-6.37%)	38.703 (-14.71%)

Per Node stands for the total energy consumption per node; Average Power stands for the average power consumption per node during the application execution. The combination of DCT+DVFS with loop optimizations yields the average power reduction in the range of 6-8%, and energy reduction by up to 21%. Table 8 present the performance results for the MPI BT-MZ application with Class D. The percentage improvements are similar to that shown in Table 7. DVFS is applied during the initialize solutions kernel and the exchange of boundary conditions, which contains significant MPI communication. Additional loop optimizations are applied to class D (block size = 4x4). Comparing Table 8 with Table 7, we find that the use of DCT for hybrid BT-MZ contributes a little bit more power saving.

Table 9 Performance Comparison of Hybrid SP-MZ with Class D

#Cores (MxN)	SP-MZ Type	Runtime (s)	Average Power (W)	Energy Per Node (KJ)
6x8	Hybrid	862	340.453	293.404
	Optimized -Hybrid	819 (-5.25%)	320.19 (-6.32%)	262.236 (-11.89%)
8x8	Hybrid	653	340.59	222.408
	Optimized -Hybrid	607 (-7.57%)	323.12 (-5.4%)	196.134 (-13.39%)
16x8	Hybrid	389	341.14	132.703
	Optimized -Hybrid	344 (-13%)	322.57 (-5.71%)	110.964 (-19.59%)
32x8	Hybrid	225	340.11	76.525
	Optimized -Hybrid	205 (-9.76%)	315.03 (-7.96%)	64.581 (-18.49%)
64x8	Hybrid	154	341.14	52.536
	Optimized -Hybrid	142 (-8.45%)	316.87 (-7.66%)	44.996 (-16.75%)

5.2.2 SP-MZ

SP-MZ represents a fairly balanced workload. To reduce the CPU frequency during the application execution, we apply DVFS to the initial solutions kernel and take the approach of reducing the frequency for the first 100 time steps of the application kernel to limit the additional overhead that would be introduced from lowering the frequency. DCT is applied during the SP solver kernel to reduce the power consumption during this phase. Additional loop optimizations are applied to class D (block size = 4x4). Table 9 presents the performance results for the hybrid SP-MZ application for Class D. The combination of DCT+DVFS with loop optimizations results in the average power reduction by up to 7.96%, and saves energy by up to 19.59%.

Table 10 presents the performance results for the MPI SP-MZ. The percentage improvements are similar to that shown in Table 9. We reduce the power consumption of the application by applying DVFS to the initialization kernel and first 150 time steps of the application to limit the additional overhead that would be introduced from lowering the frequency for different kernels as the program executes. Loop optimizations are applied (block size = 8x8) with loop unrolling being applied to the inner loops of the SP solver kernel.

5.2.3 GTC

To reduce the CPU frequency during the application execution, we apply DVFS to the initialization kernel and the first 25 time steps of the application during execution. This provides the optimal execution setting to limit the additional overhead that would be introduced from lowering the frequency throughout the en-

Table 10 Performance Comparison of MPI SP-MZ with Class D

#Cores (MxN)	SP-MZ Type	Runtime (s)	Average Power (W)	Energy Per Node (KJ)
6x8	MPI	881	339.14	298.782
	Optimized -MPI	807 (-9.18%)	315.02 (-6.59%)	254.221 (-17.53%)
8x8	MPI	689	338.46	233.199
	Optimized -MPI	619 (-11.3%)	321.07 (-5.41%)	198.742 (-17.33%)
16x8	MPI	413	337.45	139.367
	Optimized -MPI	369 (-11.9%)	322.45 (-4.65%)	118.984 (-17.13%)
32x8	MPI	241	338.14	81.492
	Optimized -MPI	229.53 (-5.0%)	315.15 (-7.30%)	72.336 (-12.66%)
64x8	MPI	173.87	337.63	58.704
	Optimized -MPI	165.81 (-4.84%)	312.29 (-8.11%)	51.781 (-13.37%)

Table 11 Performance Comparison of Hybrid GTC (100PPC)

#Cores (MxN)	GTC Type	Runtime (s)	Average Power (W)	Energy Per Node (KJ)
6x8	Hybrid	928	330.54	306.74
	Optimized -Hybrid	904 (-2.65%)	301.45 (-9.65%)	272.51 (-12.29%)
8x8	Hybrid	934	333	311.02
	Optimized -Hybrid	902 (-3.55%)	297 (-12.12%)	274.21 (-13.42%)
16x8	Hybrid	947	334	316.30
	Optimized -Hybrid	906 (-4.53%)	298 (-12.1%)	269.99 (-17.15%)
32x8	Hybrid	954	328.89	313.76
	Optimized -Hybrid	918 (-3.92%)	296.80 (-10.81%)	272.46 (-15.16%)
64x8	Hybrid	958	328.79	314.98
	Optimized -Hybrid	923 (-3.79%)	294.15 (-11.77%)	271.5 (-16.01%)

tire application. Additional loop optimizations are applied to the problem sizes of 50ppc (block size = 2x2) and 100ppc (block size = 4x4). The inner-most loops of the pushi and chargei subroutines are the most computationally intensive kernels of the application and are unrolled four times. Table 11 presents the performance results for the hybrid GTC application for the problem size of 100ppc. The combination of DCT+DVFS with loop optimizations results in the average power reduction by up to 12.12%, and saves energy by up to 17.15%.

Table 12 presents the performance results for the MPI implementation of the GTC application for the problem size of 100ppc. The percentage improvements are similar to that shown in Table 11. To reduce the frequency during the application execution we apply DVFS to all kernels that are executed during the first 30 time steps of the application to limit the additional overhead that would be introduced from lowering the frequency throughout the entire application.

Table 12 Performance Comparison of MPI GTC (100PPC)

#Cores (MxN)	GTC Type	Runtime (s)	Average Power (W)	Energy Per Node (KJ)
6x8	MPI	1413.19	338.19	477.93
	Optimized -MPI	1389.38 (-1.71%)	302.12 (-11.94%)	419.76 (-13.86%)
8x8	MPI	1440.02	339.32	488.63
	Optimized -MPI	1401.9 (-2.71%)	306.57 (-10.68%)	429.78 (-13.69%)
16x8	MPI	1456	339.45	494.24
	Optimized -MPI	1413.34 (-3.02%)	306.19 (-10.86%)	432.75 (-14.21%)
32x8	MPI	1483.13	339.12	502.96
	Optimized -MPI	1451.39 (-2.19%)	304.19 (-11.48%)	441.50 (13.92%)
64x8	MPI	1513.39	339.05	513.14
	Optimized -MPI	1459.10 (-3.72%)	301.38 (-12.50%)	439.74 (-16.69%)

Loop blocking is applied to the MPI implementation with an optimal block size of 4x4 for both problem sizes of 50 ppc and 100 ppc. Similar to the hybrid implementation, the inner-most loops of the pushi and chargei subroutines are unrolled four times. The manual loop optimizations are able to achieve strong reductions in execution time for 50 ppc, but smaller optimization benefits are obtained in terms of execution time for 100 ppc. It is important to note that the GTC application benefits greatly for the use of OpenMP threads during parallelization shown in Tables 11 and 12.

5.2.4 PMLB

We apply DVFS to the initialization and final kernels of the applications. Additional loop optimizations are applied to execute the application using a block size of 4x4 and nested loops within the application are unrolled four times. Table 13 presents the performance results for the hybrid PMLB application with the problem size of 256x256x256. The combination of DCT+DVFS with loop optimizations results in the average power reduction by up to 10.37%, and saves energy by up to 16.9%.

The MPI PMLB application is optimized by applying DVFS to reduce power consumption during the application execution. To reduce the CPU frequency during the execution we apply DVFS to the initialization, communication, and final kernels of the applications. Additional loop optimizations are applied to execute the application using a block size of 4x4 and nested loops within the application are unrolled four times. Table 14 presents the performance results for the MPI PMLB application with the problem size of

Table 13 Performance Comparison of Hybrid PMLB

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

Table 14 Performance Comparison of MPI PMLB

#Cores (MxN)	PMLB Type	Runtime (s)	Average Power (W)	Energy Per Node (KJ)
1x8	MPI	1259.87	302.76	381.44
	Optimized -MPI	1247.13 (-1.02%)	284.90 (-6.27%)	355.27 (-7.37%)
2x8	MPI	689.31	302.98	208.85
	Optimized -MPI	664.19 (-3.78%)	282.14 (-7.39%)	187.39 (-11.45%)
4x8	MPI	379.12	301.18	114.18
	Optimized -MPI	362.29 (-4.65%)	282.33 (-6.68%)	102.29 (-11.62%)
8x8	MPI	185.35	300.79	55.75
	Optimized -MPI	180.13 (-2.90%)	281.13 (-6.99%)	50.64 (-10.1%)
16x8	MPI	88.93	300.84	26.75
	Optimized -MPI	89.46 (0.59%)	285.14 (-5.51%)	25.51 (-4.86%)
32x8	MPI	43.12	301.29	12.99
	Optimized -MPI	46.79 (7.84%)	286.91 (-5.01%)	13.42 (3.2%)

256x256x256. Although our optimization method results in the average power reduction by up to 7.39%, the energy consumption was increased by 3.2% for the case of 32x8 because of the large increase (7.84%) of the execution time. For the case of 16x8, although the execution time was increased by 0.59%, the power consumption was saved by 5.51%, this results in the overall energy saving. It is important to note that the PMLB application does not benefit from the use of OpenMP threads during parallelization shown in Tables 13 and 14.Overall, we have presented the optimization results for four of the six large-scale scientific applications. These results show improvements in runtime by up to 14.15% and reductions in energy consumption by up to 21%. For the sake of space, the LU-MZ and EQdyna applications are not presented in detail, their optimization results show the similar improvement as well.

6 Related Work

The use of performance counters to predict power consumption has been explored in the previous work [3, 5–7, 16, 22, 23]. In general, the previous work identifies a set of common performance counters to be used across all of the applications considered. This approach develops a unified model using a group or class of applications to estimate power consumption, measures activity on the system and correlates it to the power consumption being used by the application. The same counters and correlation coefficients are used for the class or group of applications, but this approach doesn't capture some characteristics unique to each application.

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

In [23] power estimations using counters are presented with median errors of 5.63%. This work makes uses of performance counters to measure the effects of cache resource and thermal effects to develop a power-aware thread scheduler. In [14] two energy-saving techniques, DVFS and DCT, are applied to hybrid HPC application codes to improve energy consumption with energy savings in the range of 4.1% to 13.8% with negligible performance loss. This work focuses on reducing energy consumption by identifying the effects that DCT has on other MPI tasks during execution and identifying slack due to intra and inter-node interaction in hybrid HPC applications. Our work utilizes the software-based power reduction strategies with algorithmic changes to improve application performance and save power. Our scheme makes use of performance models that are used for predicting the effects that DVFS and DCT strategies have on application performance by refining the regression model for each applications characteristics.

Overall, our work differs from previous approaches in that we identify alternative frequency and concurrency settings for an applications kernel to reduce power consumption. Unlike prior work, we also consider algorithmic changes to optimize the kernel for better performance through loop blocking and loop unrolling. The reduced power consumption and reduced execution time reduces the energy consumption of the application. Previous methods focus largely on only introducing software-based power reduction strategies. In our work, we utilize software-based power reduction strategies with algorithmic changes to improve application performance.

7 Summary

The E-AMOM framework provides an accurate methodology for predicting and improving the performance and power consumption of HPC applications. E-AMOM includes two software-based approaches for reducing power consumption in HPC applications, DVFS and DCT. Specifically, E-AMOM determines efficient execution configurations of HPC application kernels with regard to the number of OpenMP threads to utilize to execute each application kernel in the hybrid applications. Overall, our E-AMOM predictive models have the average error rate of up to 6.79% across six hybrid and MPI scientific applications. Further, our work is able to identify the trends exhibited by each applications implementations to determine which will provide for the best energy consumption. Our optimization approach is able to reduce the energy consumption by up to 21% with a reduction in runtime by up to 14.15% and a reduction in power consumption by up to 12.50% in six hybrid and MPI HPC applications.

Future research will focus on obtaining detailed power and energy profiles to determine the power consumption of the application in terms of utilization of the system, CPU, memory, motherboard, and hard disk. Additional work will focus on identifying appropriate optimization strategies to handle these alternative classes of applications to include in E-AMOM. E-AMOM will be further integrated into the MuMMI framework to automate the modeling and optimization processes to identify optimal configuration points.

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, and Chee Wai Lee for his review comments.

References

1. E. Bair, T. Hastle, D. Paul, and R. Tibshirani, Prediction by Supervised Principal Components, J. of the American Statistical Ass., vol. 101, no. 473, pp. 119-137, 2006.
2. F. Bellosa, The Benefits of Event-Driven Energy Accounting in Power-Sensitive Systems, Proc. 9th ACM SIGOPS European Workshop, pp. 37-42, Sep. 2000.
3. W. Lloyd Bircher, Lizy K. John, Complete System Power Estimation Using Processor Performance Events,, IEEE Transactions on Computers, Vol. 61, No. 4, April 2012.
4. B. Cmelik, D. Keppel, Shade: A Fast Instruction Set Simulator for Execution Profiling, Proc. Intl Symp. Measurement and Modeling of Computer Systems (SIGMETRICS 04), pp. 128-137, June 1994.
5. M. Curtis-Maury, J. Dzierwa, C. Antonopoulos, and D. Nikolopoulos, Online Power-Performance Adaptation of Multithreaded Programs using Hardware Event-Based Prediction, Proc. Intl Conf. on Supercomputing (ICS 06), pp. 157-166, June 2006.
6. M. Curtis-Maury et al., Prediction-Based Power-Performance Adaption of Multithreaded Scientific Codes, Proc. IEEE Transactions on Parallel and Distributed Systems (TPDS 08), vol. 19, no. 10, pp. 1396-1410, 2008.
7. M. Curtis-Maury, A. Shah, F. Blagojevic, D. Nikolopoulos, B. R. de Supinski, et al., Prediction Models for Multi-dimensional Power-Performance Optimization of Many Cores, Proc. 17th Intl Conf. Parallel Architectures and Compilation Techniques (PACT 08), pp. 250-259, 2008.
8. S. Ethier, First Experience on BlueGene/L, BlueGene Applications Workshop, April, 2005.
9. R. Ge, X. Feng, S. Song, H. Chang, D. Li and K.W. Cameron, PowerPack: Energy Profiling and Analysis of High-Performance Systems and Applications, IEEE Transactions on Parallel and Distributed Systems 21(5): 658-671, 2010.
10. V. Freeh, D. Lowenthal, F. Pan, N. Kappiah, R. Springer, B. Rountree, and M. Femal. Analyzing the Energy-Time Trade-Offs in High-Performance Computing Applications, IEEE Transactions on Parallel and Distributed Systems, pp. 835-848, Oct. 2007.
11. V. Freeh, Feng Pan, D. Lowenthal, and N. Kappiah. Using Multiple Energy Gears in MPI Programs on a Power-Scalable Cluster, Proc. 10th ACM Symp. Principles and Practice of Parallel Programming, pp. 164-173, June 2005.
12. H. Jin, R. F. Van der Wijngaart, Performance characteristics of the Multi-Zone NAS Parallel Benchmarks, J. Para. Dist. Comp., vol. 66, no. 5, pp. 674-685, 2004.
13. N. Kappiah, V. Freeh, and D. Lowenthal. Just In Time Dynamic Voltage Scaling: Exploiting Inter-Node Slack to Save Energy in MPI Programs, the 2005 ACM/IEEE Conference on Supercomputing (SC05), 2005.
14. D. Li, B. de Supinski, M. Schulz, K. Cameron and D. Nikolopoulos, Hybrid MPI/OpenMP Power-Aware Computing, Proc. 24th IEEE Intl Parallel & Distributed Processing Symp., pp. 1-12, May 2010.
15. D. Li, D. Nikolopoulos, K. Cameron, B. de Supinski, and M. Schulz, Power-Aware MPI Task Aggregation Prediction for High-End Computing Systems, Proc. 24th IEEE Intl Parallel & Distributed Processing Symp., pp. 1-12, 2010.
16. M. Lim, A. Porterfield, and R. Fowler, SoftPower: Fine-Grain Power Estimations Using Performance Counters, Proc. 19th Intl Symp. High Performance Distributed Computing (HPDC 10), pp. 308-311, June 2010.
17. C. Lively, E-AMOM: An Energy-Aware Modeling and Optimization Methodology for Scientific Applications on

Multicore Systems. Doctoral dissertation, Texas A&M University, 2012.
18. C. Lively, X.Wu, V.Taylor, S. Moore, H.-.C. Chang, et al., Power-Aware Predictive Models of Hybrid (MPI/OpenMP) Scientific Applications on Multicore Systems, Proc. Intl Conf. on Energy-Aware High Performance Computing (EnA-HPC 11), Sept 2011.
19. C. Lively, X.Wu, V.Taylor, S. Moore, H.-.C. Chang, et al., Energy and Performance Characteristics of Different Parallel Implementations of Scientific Applications on Multicore Systems, Intl J. of High Performance Computing Applications (IJHPCA), vol. 25, no. 3, pp. 342 350, Aug. 2011.
20. A. Miyoshi, C. Lefurgy, E. Hensbergen, R. Rajamony, and R. Rajkumar, Critical power slope: Understanding the runtime effects of frequency scaling, Proc. 2005 ACM/IEEE Conf. Supercomputing (SC 05), pp. 3544, Nov. 2002.
21. Performance Application Programming Interface, papi, http://icl.cs.utk.edu/papi/, 2012.
22. K. K. Pusukuri, D. Vengerov, and A. Fedorova, A Methodology for Developing Simple and Robust Power Models using Performance Monitoring Events, Proceedings of WISOCA, Austin, Texas, USA, June 2009.
23. 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.
24. SystemG at Virginia Tech, http://www.cs.vt.edu/facilities/systemg.
25. V. Taylor, X. Wu, and R. Stevens, Prophesy: An Infrastructure for Performance Analysis and Modeling of Parallel and Grid Applications, ACM SIGMETRICS Performance Evaluation Review, vol. 30, no. 4, pp. 13-18, Mar. 2003.
26. X. Wu, B. Duan and V. Taylor, Parallel Simulations of Dynamic Earthquake Rupture Along Geometrically Complex Faults on CMP Systems, Journal of Algorithm and Computational Technology, Vol. 5 No. 2, 2011, pp. 313-340.
27. X. Wu, C. Lively, V. Taylor, H. Chang, C. Su, K. Cameron, S. Moore, D. Terpstra and V. Weaver, MuMMI: Multiple Metrics Modeling Infrastructure, the 14th IEEE/ACIS International Conf. on Soft. Eng., Art. Intel., Net. and Para./Dis. Comp., July 2013, Honolulu, Hawaii. Also see the link http://www.mummi.org.
28. X. Wu, V. Taylor, S. Garrick, D. Yu, and J. Richard, Performance Analysis, Modeling and Prediction of a Parallel Multiblock Lattice Boltzmann Application Using Prophesy System, IEEE International Conference on Cluster Computing, Sept. 2006.
29. X. Wu, V. Taylor, C. Lively, and S. Sharkawi, Performance Analysis and Optimization of Parallel Scientific Applications on CMP Clusters, Scalable Computing: Practice and Experience, vol. 10, no. 1, pp. 61-74, 2009