MuMMI:
Multiple Metrics Modeling Infrastructure

Xingfu Wu*, Valerie Taylor*, Charles Lively*
Hung-Ching Chang**, Bo Li**, Kirk Cameron**
Dan Terpstra#, and Shirley Moore+

Abstract MuMMI (Multiple Metrics Modeling Infrastructure) is an infrastructure that facilitates systematic measurement, modeling, and prediction of performance and power consumption, and performance-power tradeoffs and optimization for parallel systems. MuMMI builds upon three existing frameworks: Prophesy for performance modeling and prediction of parallel applications, PAPI for hardware performance counter monitoring, and PowerPack for power measurement and profiling. In this paper, we present the MuMMI framework, which consists of an Instrumentor, Databases and Analyzer. The MuMMI Instrumentor provides automatic performance and power data collection and storage with low overhead. The MuMMI Databases store performance, power and energy consumption and hardware performance counters' data with different CPU frequency settings for modeling and comparison. The MuMMI Analyzer entails performance and power modeling and performance-power tradeoff and optimizations. For case studies, we apply MuMMI to a parallel earthquake simulation to illustrate building performance and power models of the application and optimizing its performance and power for energy efficiency.

1 Introduction

The burgeoning revolution in high-end computer architecture has far reaching implications for the software infrastructure of tools for performance measurement, modeling, and optimization, which has been indispensable to improved productivity in computational science over the past decade. Significant work has been done on ex-

* Dept. of Computer Science & Engineering, Texas A&M University
Email: {wuxf, taylor}@cse.tamu.edu
** Dept. of Computer Science, Virginia Tech
# Innovative Computing Lab, University of Tennessee
+ Dept. of Computer Science, University of Texas at El Paso

ploring power reduction strategies for large-scale, parallel scientific applications [2], [3], [4], [6], [7], [16]. The problem is that much of this work required manual data collection and analysis to explore the different power reduction techniques. Very little work has been done with respect to providing an infrastructure for automating as much of this process as possible in addition to archival of the data. The MuMMI (Multiple Metrics Modeling Infrastructure)[11] was developed to provide an infrastructure that facilitates systematic measurement, modeling, and prediction of performance and power consumption, and performance-power tradeoffs and optimization for multicore systems.

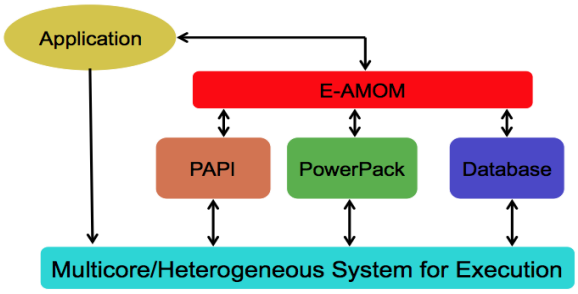

Fig. 1 Multiple Metrics Modeling Infrastructure (MuMMI).

MuMMI is depicted in Figure 1, as a multi-level infrastructure for integrated performance and power modeling for large-scale parallel systems. MuMMI builds upon an established infrastructure for performance modeling and prediction (Prophesy [22]), hardware performance counter measurement (PAPI [13]), and power profiling (PowerPack [4]). MuMMI extends these existing infrastructures in the following ways:

- Extension of Prophesy's well-established performance modeling interface to encompass multicores and to incorporate power parameters and metrics into the performance models and optimizations called E-AMOM (Energy-Aware Modeling and Optimization Methodology). A database component enables the modeling system to record and track relevant benchmark codes and results for characterizing multicore architectures.
- Building on top of PAPI's multi-component architecture, MuMMI provides a user-configurable layer for defining derived higher level metrics relevant to the performance modeling task at hand and mapping these to available native events on a given platform.
- Extension of an emerging power-performance measurement, profiling, analysis and optimization framework, called PowerPack, to multicore architectures. This work enables MuMMI to measure and predict power consumption at component (e.g. CPU, memory and disk) and function-level granularity for parallel architectures.

As a result of these combined efforts, MuMMI is able to offer an integrated and extensible performance modeling and prediction framework for use by application developers, system designers, and scientific application users, helping them to make key choices for configuring and selecting appropriate parallel systems and to evaluate and optimize power/performance on these systems.

In this paper, we discuss the MuMMI framework which consists of three main components: Instrumentor, Databases and Analyzer in detail. The MuMMI Instrumentor provides for automatic performance and power data collection and storage with low overhead. The Instrumentor has been used for our recent research work in [8], [9], [10]. MuMMI Databases extend the databases of Prophesy to store power and energy consumption and hardware performance counters' data with different CPU frequency settings. The MuMMI Analyzer (called E-AMOM) extends the data analysis component of Prophesy to support power consumption and hardware performance counters, and it entails performance and power modeling, and performance-power tradeoff and optimizations.

The remainder of this paper is organized as follows. Section 2 discusses the MuMMI framework in detail. Section 3 presents the MuMMI Instrumentor framework. Section 4 applies MuMMI to a parallel earthquake simulation to illustrate building performance and power models of the application and optimizing its performance and power for energy efficiency. Section 5 summarizes the paper.

2 MuMMI Framework

In this section, we discuss the four main components of the MuMMI framework shown in Figure 1: PAPI, PowerPack, MuMMI database and E-AMOM.

2.1 PAPI

Hardware performance monitors can provide a window on the hardware by exposing detailed information on the behavior of functional units within the processor, as well as other components of the system such as memory controllers and network interfaces. The PAPI project designed and implemented a portable interface to the hardware performance counters available on most modern microprocessors [13], [1]. The PAPI specification includes a common set of PAPI standard events considered most relevant to application performance evaluation – cycle and operation counts, cache and memory access events, cache coherence events, and branch prediction behavior – as well as a portable set of routines for accessing the counters. Moreover, performance monitors can now be found in memory controllers, network fabrics, and thermal monitors, as well as on special purpose accelerators such as GPUs. The work to restructure the PAPI library contains the exposed API and platform independent code, and a collection of independently loadable com-

ponents, with one component for each set of hardware monitoring resources. As part of MuMMI, this work allows the simultaneous monitoring and correlation of performance data from multiple components and levels of the system.

2.2 PowerPack

PowerPack [4] is a collection of software components, including libraries and APIs, which enable system component-level power profiling correlated to application functions. PowerPack obtains measurements from power meters attached to the hardware of a system. The framework includes APIs and control daemons that use DVFS (dynamic voltage and frequency scaling) to enable energy reduction with little impact on the performance of the system. As multicore systems evolve, the framework can be used to indicate the application parameters and the system components that affect the power consumption on the multicore unit. PowerPack allows the user to obtain direct measurements of the major system components' power consumption, including the CPU, memory, hard disk, and motherboard. This fine-grained measurement allows power consumption to be measured on a per-component basis.

2.3 MuMMI Database

The MuMMI database facilitates contributions from and sharing among the high performance computing research community. In particular, MuMMI leverages from the Prophesy database [27], [23] that allows for web-based entry of data in addition to automatic upload of data via SOAP[17]. MuMMI database is the extension of the Prophesy database, which includes additional information relevant to power profiling, energy and hardware counters' performance on multicore platforms.

The MuMMI database schema is shown in Figure 2. The MuMMI database is organized into four areas: application information, executable information, run information, and performance statistics. The application information includes details about the application and its developer. The executable information includes all of the entities related to generating an executable of an application. The run information includes all of the entities associated with running an executable and the system used for execution. Lastly, the performance statistics includes the performance data generated during the application execution.

PAPI Multiplexing [13] allows a user to count more events than total physical counters simultaneously by a system. We collected the data for 40 hardware performance counters with different CPU frequency settings. The power and energy data from PowerPack includes system energy, CPU energy, memory energy, motherboard energy, disk energy, and power over time for each component (CPU, mem-

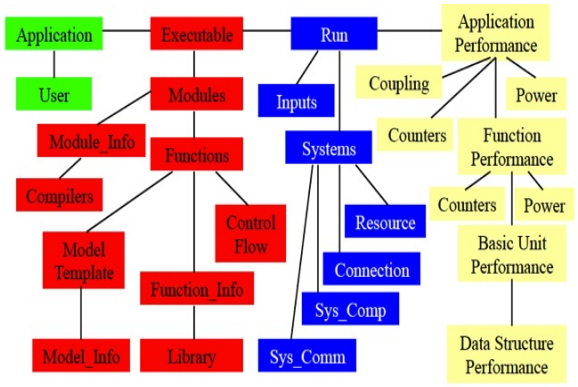

Fig. 2 MuMMI Database Schema.

ory, motherboard and disk). The power and energy data are for an entire application or functions of the application.

2.4 E-AMOM

Based on hardware performance counters' data and power data from the MuMMI Database, 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-aware optimizations for scientific applications. These performance counters-based modeling methods [10], [8] identify the different performance counters that are needed to accurately predict the application performance, and provides insight to improve the application performance.

During each application execution, we capture 40 performance counter events utilizing the PAPI. All performance counter events are normalized using the total cycles of the execution to create performance event rates for each counter. Using Spearman correlation and principal component analysis, we identify around 5 important performance counters for the multiple metrics of the application. The multiple metrics include runtime, system power consumption, CPU power consumption, and memory power consumption. Then we use multivariable regression analysis to obtain the model equations for them based on these important performance counts and CPU frequencies.

E-AMOM utilizes these 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.

3 MuMMI Instrumentor

Generally, collecting consistent performance and power data is difficult because it requires multiple executions of an application using each PAPI, PowerPack and Prophesy. Our goal for the MuMMI Instrumentor is to make performance and power data collection easy and automatic requiring only one application execution for the collection of performance and power data. To the best of our knowledge, this is the first tool to support automatic and simultaneous performance, power and energy data collection.

The MuMMI Instrumentor provides a way to do source-code level automatic instrumentation for Fortran77, Fortran90, C and C++ programs. It is an extension of Prophesy data collection component [23] with the addition of power and hardware counters data collection and dynamic CPU frequency scaling support. The basic framework of the MuMMI Instrumentor is illustrated in Figure 3, which entails automatically inserting instrumentation code into a source code file. The Instrumentor supports the following instrumentation options:

-a: Instrument all procedures and outer loops
-p: Instrument all procedures
-l: Instrument all loops
-n: Instrument all procedures not nested in loops
-F: Use Perl SOAP scripts to automatically transfer performance data to the MuMMI database
-W: Use PowerPack to collect power data
Default: Instrument procedures and outer loops

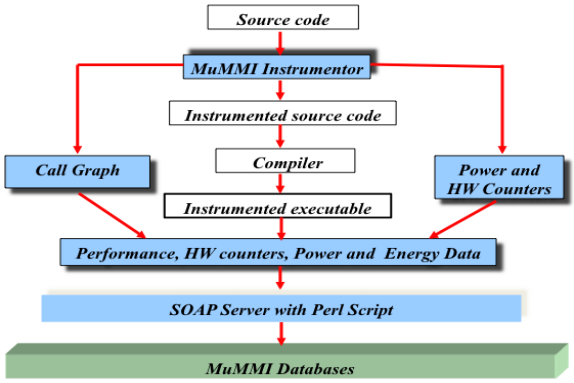

Fig. 3 MuMMI Instrumentor Framework.

These instrumentation options (-a, -p, -l, -n, or default) can be used to instrument a source code at different levels based on the user, ranging from only procedures and outler loops to nested loops. The user also can select the source files to be

instrumented. However, the following routines must be instrumented: the main file such as *main()* function for a C program or the *PROGRAM* for a Fortran program. The MuMMI Instrumentor counts each procedure and loop in the source code, and labels it with a unique instrumentation point number (or called event identifier).

For the case when the "-W" option is used, the MuMMI Instrumentor records the starting and end timestamps of the program, and at the end of a program execution, it sends these timestamps to PowerPack server to retrieve the power profiling data for the program, and appends the power and energy data to the output performance data files.

For the case when the "-F" option is used, at the end of the program execution, all performance data files are automatically transferred to the MuMMI database by the MuMMI Instrumentor. We use SOAP Lite package to develop Perl SOAP scripts to upload these data files to the MuMMI database (locally or remotely). Prior to invoking the SOAP client side to transfer the data files, the data files are converted to SQL scripts based on MuMMI database schema shown in Figure 2 so that the SOAP server side receives the data files, and uploads the data to the MuMMI database. Further, the MuMMI Instrumentor is able to provide automatic upload of the information obtained from widely used performance analysis tools such as gprof, IPM [5], TAU [20] or Score-P [15] to the MuMMI database via some Perl SOAP scripts as well.

When the instrumented source code is generated, it requires the following compiler options to support different functionalities:

-DMPI: Support MPI programs
-DPAPI: Allow collecting hardware counters' data using PAPI
-DFS: Support dynamic frequency scaling

It also requires PAPI library link -lpapi to support collecting hardware counters' data using PAPI. If the "-DFS" option is used, the MuMMI Instrumentor can support dynamic frequency scaling by adjusting the CPU frequency to what a user sets in the system file scaling_setspeed. It requires the user to specify CPU frequencies for scaling up or down in advance.

4 Case Studies

In this section, we apply MuMMI to a parallel earthquake simulation to illustrate building performance and power models of the application and optimizing its performance and power for energy efficiency.

4.1 A MuMMI Testbed and A Parallel Earthquake Simulation

In this section, we describe a MuMMI testbed system, and deploy the MuMMI Instrumentor on the system. Our experiments utilize one multicore system from Virginia Tech, SystemG [19], which has 325 Mac Pro computer nodes. This system is the largest PowerPack-enabled research system with each node containing more than 30 thermal sensors and more than 30 power sensors. Each node has two quad-core 2.8 GHz Intel Xeon processors and 8 GB memory. The CPU frequency for the Intel Xeon processor can only be adjusted to 2.4 GHz and 2.8 GHz.

Our experiments utilize a parallel earthquake simulation developed in [24], [25] with the SCEC/USGS benchmark problem TPV210. The benchmark problem TPV210 is the convergence test of the benchmark problem TPV10 [18]. In TPV10, a normal fault dipping at 60^o (30 km long along strike and 15 km wide along dip) is embedded in a homogeneous half space. Pre-stresses are depth dependent and frictional properties are set to result in a sub-shear rupture. In the benchmark problem TPV210, we conduct the convergence test of the solution by simulating the same problem at a set of element sizes, i.e., 200 m, 100 m, 50 m, and so on, where m stands for meters. The simulation is memory-bound. Our hybrid MPI/OpenMP finite element earthquake simulations discussed in [24], [25] target the limitation to reduce large memory requirements. For the sake of simplicity, we only use TPV210 with 200m element size in this paper.

4.2 Performance and Power Modeling

In this section, utilizing MuMMI Instrumentor, we develop application-centric models for the runtime and power consumption of the system, CPU, and memory.

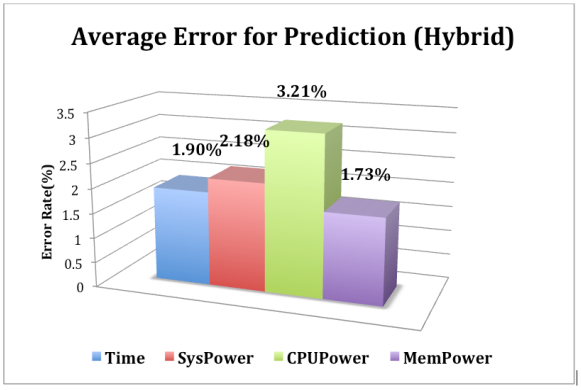

Fig. 4 Average Prediction Error Rate Using Models for the Hybrid Earthquake Simulation.

To build a model, we create a typical training set for various configuration points first. The configuration M x N stands for M nodes with N cores per node. In the case of our hybrid MPI/OpenMP finite element earthquake application with the resolution of 200m, M is the number of nodes and N is the number of cores with one OpenMP thread per core. For the hybrid application, a configuration 2x8 means that 2 MPI processes with 8 OpenMP threads per node with a total of 16 cores used.The training set for inter-node performance uses six data points consisting of configurations for points at 1x8, 3x8, 5x8, 7x8, and 9x8, 10x8. Performance counters' data and power profiling data for these training points are used to build performance and power models, then these models are used to predict performance and power consumption on larger number of processor cores (up to 64x8).

Figure 4 shows the average prediction error rate resulting from the modeling of the hybrid earthquake application. The average error rate for predicting the application runtime is 1.90%.The average error rate for predicting the system power consumption is 2.18%.The average error rate for predicting the CPU power consumption is 3.21%.The average error rate for predicting the memory power consumption is 1.73%. Overall, the models for the hybrid application have an average prediction error of less than 4%.

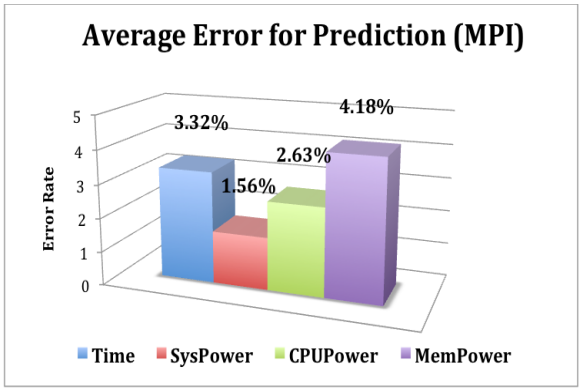

Fig. 5 Average Prediction Error Rate Using Models for the MPI-only Earthquake Simulation.

Figure 5 shows the average prediction error rate resulting from the modeling of the MPI-only earthquake application. The performance components modeled for the MPI-only earthquake application had the smallest error for predicting the system power consumption (1.56%) and the highest error rate occurred in modeling the memory power consumption (4.18%).

Overall, our modeling method identifies the different performance counters that are needed to accurately predict the application performance and power consumptions.

4.3 Performance-Power Trade-off and Optimization

In this section, we incorporate two common software-based approaches for reducing power consumption in large-scale parallel systems, Dynamic Voltage and Frequency Scaling (DVFS)[6] and Dynamic Concurrency Throttling (DCT) [2]. DVFS can be used to scale down the CPU frequency for a HPC application's workload resulting in lower power consumption. DVFS is most beneficial when applied to regions within an application where communication does not overlap with computation. DCT is used to control the number of threads assigned to a segment of a parallel code. DCT can be applied to a code region with a reduced workload that would not benefit from using the maximum number of cores on a chip.

Before applying DCT and DVFS to the application executed on SystemG, we investigated how the settings of having 1, 2, 4, 6, or 8 OpenMP threads per node impact the application performance by utilizing the MuMMI Instrumentor to collect the performance and power consumption data, and found using 2 OpenMP threads per node for the functions qdct3 and hourglass results in the best application performance. So DCT is applied to the functions hourglass and qdct3 so that they are executed using 2 OpenMP threads per node during the application execution. DVFS is applied to the functions input, communication, and final for the communication slacks by adjusting CPU frequency from 2.8GHz to 2.4GHz.

As we discussed before, the hybrid earthquake application is memory bound. From our experiments, we found that L2 cache behavior is a major factor for memory power consumption. So additional loop optimizations such as using loop blocking with a block size of 8x8 and unrolling nested loops four times are applied to the application in order to improve cache utilization.

#Cores	Program Type	Runtime(s)	System Energy (KJ)	System Power (W)	CPU Power (W)	Memory Power (W)
2x8	Hybrid	3156	1760.72	557.9	187.78	200.3
	Optimized-Hybrid	2980 (-5.9%)	1568.62 (-12.25%)	526.38 (-5.98%)	160.38 (-17.2%)	195.08 (-2.67%)
3x8	Hybrid	2166	1807.47	834.48	277.26	297.63
	Optimized-Hybrid	2031 (-6.65%)	1583.52 (-14.14%)	779.67 (-7.03%)	248.61 (-11.52%)	288.39 (-3.20%)
4x8	Hybrid	1681	1895.64	1127.68	375.56	404.12
	Optimized-Hybrid	1559 (-7.83%)	1639.4 (-15.63%)	1051.56 (-7.24%)	332.76 (-12.86%)	391.56 (-3.21%)
8x8	Hybrid	839	1900.32	2264.96	736.56	805.92
	Optimized-Hybrid	783 (-7.15%)	1656 (-14.75%)	2114.96 (-7.09%)	640.96 (-14.91%)	787.28 (-2.37%)
16x8	Hybrid	458	2117.76	4624.48	1506.72	1621.28
	Optimized-Hybrid	422 (-8.5%)	1789.28 (-18.35%)	4240 (-9.1%)	1379.04 (-9.25%)	1597.28 (-1.5%)
32x8	Hybrid	261	2411.84	9241.28	2900.16	3204.16
	Optimized-Hybrid	246 (-6.1%)	2055.36 (-17.34%)	8355.52 (-10.6%)	2694.08 (-7.65%)	3116.16 (-2.82%)
64x8	Hybrid	151	2693.12	17834.88	5703.04	6366.08
	Optimized-Hybrid	145 (-4.14%)	2318.72 (-16.15%)	15992.96 (-11.52%)	4753.28 (-19.98%)	6273.28 (-1.48%)

Fig. 6 Performance, Power and Energy Consumption Comparison.

Figure 6 shows the performance, power, and energy consumption comparison. 2x8 stands for 2 nodes with 8 cores per node. For the hybrid application executions, we use 1 MPI process per node with 8 OpenMP threads per node as default. "Optimized-Hybrid" stands for applying DVFS, DCT and loop optimization to the hybrid application for execution. "System Energy" means the total energy consumed by the number of nodes used, and its unit is kilojoule. "System Power" means the total power consumed by the number of nodes used. "CPU Power" means the total power consumed by all CPUs used. "Memory Power" means the total power consumed by all memory components used. Overall, we reduce the execution time and lower power consumption of the hybrid application by applying DVFS, DCT and loop optimizations. This is a good improvement in performance and power consumption.

As shown in Figure 6, the Memory Power consumption is larger than the CPU Power consumption for the hybrid earthquake application because the application is memory-bound, and applying DVFS, DCT and loop optimizations to the hybrid application benefit the CPU power consumption more than the Memory power consumption because we used 2 OpenMP threads per node when applying DCT to the hybrid application. It means that there are 6 idle cores per node during the execution of the dominated function qdct3 and hourglass. This results in a big CPU power saving. However, loop optimizations just improve cache utilization a little bit. For instance, on 512 cores (64x8), the optimized hybrid application execution results in 4.14% improvement in Runtime, 16.15% improvement in System Energy consumption, 11.52% improvement in System Power consumption, 19.98% improvement in CPU Power consumption, and 1.48% improvement in Memory Power consumption. So the System Energy has the best improvement percentage because the System Energy is the product of Runtime and System Power. Overall, applying DVFS, DCT and loop optimizations to the hybrid MPI/OpenMP earthquake application reduced up to 8.5% the execution time and lowered up to 18.35% energy consumption.

5 Summary

In this paper, we discussed the MuMMI framework which consists of an Instrumentor, Databases and Analyzer. The MuMMI Instrumentor provides for automatic performance and power data collection and storage with low overhead. The MuMMI Databases store performance, power and energy consumption and hardware performance counters' data with different CPU frequency settings for modeling and comparison. The MuMMI Analyzer entails performance and power modeling and performance-power tradeoff and optimizations. For case studies, we applied MuMMI to a parallel earthquake simulation to illustrate building performance and power models of the application and optimizing its performance and power for energy efficiency. Further work will extend MuMMI Instrumentor to support additional power management techniques such as IBM EMON API on BlueGene/P/Q [28], Intel RAPL[14], and NVIDIA's Management Library for power measurement

[12], and will work on performance and power modeling for CPU + GPU systems and exploring the use of correlations among performance counters to provide further optimization insights.

6 Acknowledgments

This work is supported by NSF grants CNS-0911023, CNS-0910899, and CNS-0910784.

References

1. S. Browne, J. Dongarra, N. Garner, G. Ho and P. Mucci, "A portable programming interface for performance evaluation on modern processors," International Journal of High-Performance Computing Applications, Vol. 14, No. 2, 2000.
2. M. Curtis-Maury, J. Dzierwa, et al., "Online Power-Performance Adaptation of Multithreaded Programs using Hardware Event-Based Prediction", the International Conference on Supercomputing, 2006.
3. R. Ge, X. Feng, and K. W. Cameron, "Performance-constrained Distributed DVS Scheduling for Scientific Applications on Power-aware Clusters", In IEEE/ACM SC 2005, Seattle, WA, 2005.
4. R. Ge, X. Feng, S. Song, et al., "PowerPack: Energy Profiling and Analysis of High-Performance Systems and Applications", IEEE Trans. on Parallel and Distributed Systems 21(5): 658-671, 2010.
5. Integrated Performance Monitoring (IPM), http://ipm-hpc.sourceforge.net/
6. 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.
7. D. Li, B. de Supinski, M. Schulz, K. Cameron and D. Nikolopoulos, "Hybrid MPI/OpenMP Power-Aware Computing", Proc. 24th IEEE International Conference on Parallel & Distributed Processing Symp., May 2010.
8. C. Lively, X. Wu, V. Taylor, S. Moore, H. Chang, C. Su and K. Cameron, "Power-Aware Predictive Models of Hybrid (MPI/OpenMP) Scientific Applications on Multicore Systems", Computer Science - Research and Development, Vol. 27, No. 4, Springer, 2012, pp. 245-253.
9. C. Lively, X. Wu, V. Taylor, S. Moore, H. Chang, and K. Cameron, "Energy and Performance Characteristics of Different Parallel Implementations of Scientific Applications on Multicore Systems", International Journal of High Performance Computing Applications (IJHPCA), Volume 25 Issue 3, Aug. 2011, pp. 342 - 350.
10. C. Lively, V. Taylor, X. Wu, H. Chang, C. Su, K. Cameron, S. Moore, and D. Terpstra, "E-AMOM: An Energy-Aware Modeling and Optimization Methodology for Scientific Applications on Multicore Systems", International Conference on Energy-Aware High Performance Computing, Sep. 2-3, 2013, Dresden, Germany.
11. Multiple Metrics Modeling Infrastructure (MuMMI) project, http://www.mummi.org.
12. NVIDIA, NVML API Reference Manual, 2012.
13. PAPI (Performance API), http://icl.cs.utk.edu/papi/
14. E. Rotem, A. Naveh, D. Rajwan, A. Ananthakrishnan, and E. Weissmann, "Power-management architecture of the intel microarchitecture code-named sandy bridge", IEEE Micro, 32(2):20-27, 2012.

15. Score-P, Scalable Performance Measurement Infrastructure for Parallel Codes, http://www.vi-hps.org/projects/score-p/
16. 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.
17. SOAP (Simple Object Access Protocol) , http://www.w3.org/TR/soap/
18. The SCEC/USGS Spontanous Rupture Code Verification Project, http://scecdata.usc.edu/cvws.
19. SystemG at Virginia Tech, http://www.cs.vt.edu/facilities/systemg.
20. TAU (Tuning and Analysis Utilities), http://www.cs.uoregon.edu/research/tau
21. V. Taylor, X. Wu, J. Geisler, and R. Stevens, "Using Kernel Couplings to Predict Parallel Application Performance." In Proc. of the 11th IEEE International Symposium on High-Performance Distributed Computing (HPDC 2002), Edinburgh, Scotland, July 24-26, 2002.
22. 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, Issue 4, March 2003, pp. 13-18.
23. X. Wu, V. Taylor, and R. Stevens, "Design and Implementation of Prophesy Automatic Instrumentation and Data Entry System." In the 13th International Conference on Parallel and Distributed Computing and Systems (PDCS2001), 2001.
24. 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.
25. X. Wu, B. Duan, and V. Taylor, "Parallel Earthquake Simulations on Large-scale Multicore Supercomputers" (Book Chapter), Handbook of Data Intensive Computing (Eds: B. Furht and A. Escalante), Springer-Verlag, 2011.
26. X. Wu, V. Taylor, J. Geisler, and R. Stevens, "Isocoupling: Reusing Coupling Values to Predict Parallel Application Performance," in 18th International Parallel and Distributed Processing Symposium (IPDPS2004). Santa Fe, New Mexico, 2004.
27. X. Wu, V. Taylor, et al., "Design and Development of Prophesy Performance Database for Distributed Scientific Applications", In Proc. the 10th SIAM Conference on Parallel Processing for Scientific Computing, Virginia, 2001.
28. K. Yoshii, K. Iskra, R. Gupta, P. Beckman, V. Vishwanath, C. Yu, and S. Coghlan, "Evaluating Power Monitoring Capabilities on IBM Blue Gene/P and Blue Gene/Q", IEEE Conference on Cluster Computing, 2012.