Using Performance-Power Modeling to Improve Energy Efficiency of HPC Applications

Xingfu Wu*, Valerie Taylor*, Jeanine Cook^, and Philip Mucci#

*Department of Computer Science & Engineering, Texas A&M University, TX 77843
^Sandia National Laboratory, Albuquerque, NM 87185
#Minimal Metrics LLC, Navarre, FL 32566

Abstract
To develop energy efficient applications on HPC systems, it is important to understand the relationships between runtime, power and the unique characteristics of each application. In this paper we present a power and performance modeling and energy saving framework, and use this framework to model runtime, system power, CPU power and memory power. The performance counters that compose the models, when combined with the application information, provide the basis for identifying possible application modifications that can reduce its energy. Based on these models, we develop a web-based what-if prediction system to predict the outcomes of the possible optimizations theoretically. We illustrate the use of this framework with two applications, an earthquake simulation and an aerospace application, executed on two supercomputers: Mira, the BG/Q system at Argonne National Laboratory and SystemG, the x86-64 system at Virginia Tech. Our counter-guided optimizations result in a reduction in energy by an average of up to 48.65% on up to 32,768 cores of Mira and up to 30.67% on up to 256 cores of SystemG.

Keywords: Performance modeling, power modeling, performance counters, energy optimization, HPC systems

1. Introduction

HPC systems, especially, petaflops-scale supercomputers, currently consume a tremendous amount of power. The top five systems in the current Top 500 list consume a sum total of over 50 MW, with an average power consumption of 10MW and an average performance of 17.54 petaflops (http://www.top500.org/lists/2015/11/). Taking into consideration the desire to achieve exascale systems with 20 MW of power, it is recognized that such systems will be greatly constrained by power and energy consumptions, and it requires a unique approach to balancing power and performance. To this end, it is important to understand the relationships between runtime, power consumption and the unique characteristics of each large-scale scientific application (e.g., looping constructs, data structures, data movement, communication overlapping, synchronization, etc.). Insights about these relationships provide guidance on application optimizations to reduce power and energy. The optimizations may involve application modification, system tuning, or a combination of both. In this paper, we explore the combinations.

There are many application optimization approaches to reducing runtime such as algorithm optimizations, loop-nest optimizations, compiler optimization techniques, and so on. There are also numerous programming models, languages and run-time systems that, when applied thoughtfully, can also achieve a reduction in runtimes.

Most vendors utilize power monitoring techniques at the hardware level to dynamically reduce power consumption of the various resources. This can be as simple as spinning down disk and down-clocking idle cores, to as complex as implementing asynchronously clocked circuits. Today, nearly all microprocessors contain numerous dynamic resource allocation techniques to conserve power, yet deliver performance on demand.

In addition, there are two software-based techniques to reduce power consumption for arbitrary workloads. The first is Dynamic Voltage and Frequency Scaling (DVFS), where the CPU frequency is adjusted dynamically over some time window according to both policy and demand. The other is Dynamic Concurrency Throttling (DCT), a technique that adapts the level of concurrency at run-time under similar constraints.

Saving energy intuitively implies a reduction in power consumption or a reduction in runtime, or both. Thus all research in this area can be classified into the three categories: 1) Reduce time and power; 2) Reduce time at the sacrifice of an increase in power; 3) Reduce power at the sacrifice of an increase in time. Energy (E) is the power (W) average power) over time (T), i.e., E=T *W. To clarify the three categories, we assume that the percentage change of time is a (0<a<1), and the percentage change in power consumption is b (0<b<1) compared to a baseline from which modifications are made. Below, we provide a simple mathematical analysis of the three categories.

Reduce Time and Power
Assume that the reduced time is \( T \cdot (1-a) \), and the reduced power consumption is \( W \cdot (1-b) \), we have the resultant energy: \( T^* (1-a) \cdot W^* (1-b) = (1-a)(1+b) \cdot T^* W \). A reduction in energy occurs if \( (1-a)(1+b) \cdot T^* W < T^* W \), i.e., \( b-a-ab<0 \). If \( b \leq a \), the energy saving occurs. This approach is common in methods that reduce runtime by the increase in resource utilization (e.g., increase in concurrency).

Reduce Time and Increase Power
Assume that the increased time is \( T \cdot (1+a) \), and the increased power \( W \cdot (1-b) \), we have the resultant energy: \( T^* (1+a) \cdot W^* (1-b) = (1-a)(1+b) \cdot T^* W \). A reduction in energy occurs if \( (1+a)(1-b) \cdot T^* W < T^* W \), i.e., \( a-b-ab<0 \). If \( a \leq b \), the energy saving occurs. This approach is common in methods that use DVFS. The theory being that most applications spend time waiting on everything else other than the CPU, therefore slowing down the CPU frequency saves more power than it costs in performance. This concept has led to the development of new architectures from vendors such as IBM (BlueGene), SiCortex (MIPS) and Calxeda (ARM).

In this paper, we use performance and power modeling to guide energy-efficient application developments by utilizing the first two categories because BG/Q has only one CPU frequency setting (1.6GHz).

2. Performance and Power Modeling

It is necessary to measure or estimate power consumption accurately. Since direct online power measurement at high frequencies is impractical, hardware performance counters have been widely used as effective proxies to estimate power consumption. Hardware performance counters are already incorporated within most modern architectures and are exposed to user space on commercial hardware. The performance counters monitor system components such as processor, memory, network and I/O by counting specific events such as cache misses, pipeline stalls, floating point operations, bytes in/out, bytes read/write, and so on. Statistics of such events can be collected at hardware level with little or no overhead. This makes performance counters a powerful means to monitor an application, to analyze its usage of hardware resources and to estimate its runtime and power consumption.

Much of the previous work on power modeling and estimation is based on performance counters [5, 4, 2, 11, 7, 1, 10, 8, 13, 17]. These approaches used performance counters to monitor multiple system components. The authors then attempted to correlate this data with the power consumed by each system component. That correlation was used to derive a model that could estimate the power consumption for each system component. The accuracy of their results depended on the choice/availability of the performance counters, the benchmarks/applications evaluated as well as the specific statistical data-fitting method(s) used. Many of the aforementioned approaches used a small set of performance counters (often less than 10 counters) for power modeling. In our recent work [8], we developed four different models for the metrics: runtime, system power, CPU power and memory power based upon the 40 performance counters. We found that the performance counters used for each of the different models were not the same. In studying six scientific applications, we found that a total of 37 different performance counters were used for the models, and only 3 or 4 counters were the same from model to model.

To develop models for runtime and power consumptions we collect 40 available performance counters on one system with different system configurations (number of cores, number of nodes) and application problem sizes. We then use Spearman correlation and principal component analysis (PCA) to identify the major performance counters \( (r_1, r_2, ..., r_n \ (n < 40)) \), which are highly correlated with the metric runtime, system power, CPU power or memory power. Then we use a non-negative multivariate regression analysis to generate the four models based upon the small set of major counters and CPU frequency \( f \) as follows:

For the model of runtime \( t \), we develop the following:

\[
 t = \beta_0 + \beta_1 \cdot r_1 + ... + \beta_n \cdot r_n + \beta \cdot f^{\frac{1}{2}} \tag{1}
\]

Here, \( \beta_0 \) is the intercept. \( \beta_n \) represents the regression coefficient for the performance counter \( r_n \), and \( \beta \) is the coefficient for the CPU frequency \( f \).

Similarly, we can model CPU power consumption \( p \) using the following equation

\[
p = \alpha_0 + \alpha_1 \cdot r_1 + ... + \alpha_n \cdot r_n + \alpha \cdot f^3 \tag{2}
\]

Here, \( \alpha_0 \) is the intercept. \( \alpha_n \) represents the regression coefficient for the performance counter \( r_n \), and \( \alpha \) is the coefficient for the CPU frequency \( f \). The equations for system power and memory power models are similar to Equation 2.

3. Modeling and Energy Saving Framework
Figure 1 is a general diagram of the functioning of our counter-based modeling and energy saving framework. We use MuMMI [16] to collect the performance counters as well as the four metrics we wish to correlate with and upload the data to a MuMMI database. All performance counters are normalized against the total number of cycles of the execution to create performance event rates for each counter. Next, Spearman correlation and principal component analysis (PCA) are performed to identify the significant counters that correlate with the four metrics. Then, a non-negative multivariate regression analysis is used to generate each of the four models, based upon the reduced set of counters and CPU frequencies using Equations 1 and 2. Our previous work [8, 16] indicates that the runtime and power models have prediction error rate of less than 7% on average for the six scientific applications used. MuMMI provides a web-based modeling system to automatically generate the runtime and power models based on the data for the counters and four metrics from the MuMMI database.

Fig. 1. Modeling and Energy Saving Framework

Building upon the four models for the metrics, we implement a counter ranking method to identify which of the measured counters make a significant contribution. These counters are then used to guide application modifications to achieve a reduction in both the runtime and power consumptions.

Counter Correlation Analysis and Ranking

Once we have the models for runtime, system power, CPU power and memory power, we then identify the most significant performance counters for each of the four models.

The ranking algorithm, shown in Figure 2, works as follows. First, a counter list is created consisting of the counters having the highest coefficient percentage. These counters have the highest ratio of their coefficient to the sum of the all coefficients from four models in the order of runtime, system power, CPU power and memory power. Second, in the same order, we eliminate the insignificant counters (those with less than 1%) from the counter list. Finally, we analyze and sort the correlations among these counters using pair-wise Spearman correlation to identify the counters that most significantly contribute to the models to form the final counter list. The pruning is done such that if a counter with a higher rank is highly correlated with a counter of a lower rank, the counter with the lower rank is eliminated. This resultant list of counters is used to guide application modifications.

Recommendation for Energy Saving

As we know, performance counters’ values are correlated with the properties of applications that affect performance and power. Many code optimizations solely focus on improving cache reuse to reduce the application runtime because memory access is known to be a main bottleneck on most architectures. However, these efforts are often based on performance data from just a few runs with little consideration to data-dependency, problem size and/or system configuration. Furthermore, they tend to ignore the issue of power consumption. In this work, the performance and power models are generated from different system configurations and problem sizes, and thus provide a broad understanding of the application’s usage of the underlying architecture. This leads to a better understanding of an application’s energy consumption on a given architecture. For instance, if we identify the counters $r_1$ and $r_2$ as the most important counters, and the $r_1$ dominates in the runtime model, and the $r_2$ dominates in the power models, and both are found to be uncorrelated, then our application modifications will focus on both counters. In this way, our modifications not only reduce the application runtime but also its power consumption. However, Using general purpose, power-unaware performance tools like gprof, TAU, ScoreP, HPCToolkit, HPM Toolkit and CrayPat, the impact of the counter $r_2$ maybe go entirely unnoticed.

Consider the following: Assume that $r_1$, $r_2$, and $r_k$ are three performance counters that contribute significantly to the runtime model (Equation 1) or power model (Equation 2) (power models for system, CPU or memory), and $r_1$ is identified as the most significant. $r_1$ is correlated to $r_2$ with the value of 0.9, and to $r_k$ with the value of 0.6. If the value of the counter $r_1$ is reduced by 20%, then the value of $r_2$ is reduced by 18% (0.9 * 20%), and the value of $r_k$ is reduced by 12% (0.6 * 20%). Under this assumption, we use
Equations 1 and 2 to predict the theoretical impact on runtime and power consumptions.

In general, based on runtime and power models and counter correlations, we develop a web-based what-if prediction system to predict the outcomes of possible optimizations theoretically as shown in Figure 3, where, for instance, given the application PMLB [15], if the number of TLB IM is reduced by 20%, the correlated counters are reduced based on their correlation values with TLB IM, and the runtime is reduced by 4.13%, the average node power is almost the same, and CPU and memory power consumptions are reduced a little bit.

![Fig. 3. A Web-based What-If Prediction System](image_url)

The question then immediately arises, as to how the value of the counter \( r_1 \) can be reduced by 20%. This requires a thorough understanding of the application characteristics and the portion of the underlying architecture that affects that particular metric. In [12], several typical code performance patterns are discussed, and are mapped to some hardware metrics that can assist in code optimization. It is important to realize that generalized performance counters, like PAPI presets [9], may be easily misinterpreted on different architectures by users. The users must look up the exact definition in the architecture manual, and understand how the application characteristics and the underlying architecture units affect the counters.

### 4. Case Studies: Performance Counter-Guided Energy Optimization

In this section, we use two scientific applications: the parallel aerospace application PMLB [15] and the parallel earthquake simulation eq3dyna [14] to discuss performance counter-guided energy optimization on two power-aware supercomputers, Mira, the BG/Q system at Argonne National Laboratory and SystemG, the x86-64 system at Virginia Tech.

PMLB is a Lattice Boltzmann application using the D3Q19 lattice model corresponding to 19 velocities in 3D including with the collision and streaming written in C, MPI and OpenMP. eq3dyna is a parallel finite element simulation of dynamic earthquake rupture along geometrically complex faults written in Fortran90, MPI and OpenMP.

#### 4.1 PMLB on SystemG

Figure 4 shows the performance counter rankings of the four models using 15 different counters for PMLB with the problem size of 128x128x128 on SystemG. We apply the ranking algorithm in Figure 2 to the counters for each of the four models. Ranking them from the most significant to least, they are: TLB IM (instruction translation lookaside buffer (TLB) misses), VEC_INS(vector/SIMD instructions), TLB_DM, and L2_ICM.

TLB IM has the highest rank followed by VEC_INS. We use pair-wise Spearman correlation to analyze the correlations between the two counters as follows:

- TLB IM: Contributed in Runtime
- TLB_DM: Corr Value=0.89217296 : Contributed in Runtime
- L2_ICM: Corr Value=0.88451013 : Contributed in Runtime
- VEC_INS: Contributed in System, CPU, Memory

We found the counter TLB IM only contributes in the runtime model and that it is correlated with TLB_DM and L2_ICM. VEC_INS, however, contributes in the models of system power, CPU power and memory power, and is not correlated with any other counters. Therefore, we focus on our optimization efforts on the counters TLB IM and VEC_INS on SystemG. Theoretically, using our what-if prediction system, reducing the number of TLB IM by 20% results in the reduction in runtime by 4.13%; accelerating the VEC_INS by 20% leads to the reduction in node power by 1.85%.

![Fig. 4. Ranking for the original PMLB on SystemG](image_url)

The Linux system on SystemG supports two page sizes, 4KB as default and 2MB (huge pages). A single 2MB huge page only requires a single TLB entry while the equivalent amount of memory would need 512 TLB entries using 4KB pages. As such, enabling such page sizes for an application whose performance is bound by TLB misses can be of significant benefit. Here, we enabled the 2MB pages for the application execution using libhugetlbfs to reduce the TLB misses. We also vectorized the code, and used the compiler option -fno-loop-distribution to perform loop distribution to improve cache performance on big loop bodies and allow further loop optimizations like vectorization to take place.

The results of these optimizations are shown in Figure 5. SystemG has 8 cores per node. We observe a decrease in application runtime by an average of 11.23% (Figure 5(a)),...
with an increase in system power by an average of 0.01% (Figure 5(b)). The CPU power increases by an average of 2.13% in Figure 5(c), and the memory power increases by an average of 0.61% in Figure 5(d). Overall, this represents an average energy savings of 11.28%. It is observed that the average energy saving percentage (11.28%) is bigger than the runtime improvement percentage (11.23%), meaning that reducing the runtime and power results in a larger energy saving.

### 4.2 PMLB on Mira

For 10 performance counters used in the four models for PMLB with the problem size of 512x512x512 on Mira, our ranking algorithm results in the following counters, from highest to lowest: HW_INT (number of hardware interrupts), BR_MSP, VEC_INS, L1_ICM, FDV_INS and BR_NTK.

We use pair-wise Spearman correlation and find the counter HW_INT only contributes in the Runtime, and does not have any correlated counters, however, BR_MSP are correlated with the counters L1_ICM, VEC_INS and BR_NTK. VEC_INS is correlated with BR_MSP, L1_ICM, FDV_INS and BR_NTK. FDV_INS is one of two main counters in the Runtime model, L1_ICM is the dominated one in the Memory model. Therefore, based on this information, we focus our optimization efforts on the counters BR_MSP and VEC_INS on Mira. Theoretically, using our what-if prediction system, reducing the number of BR_MSP by 20% results in the reduction in runtime by 2.07% and in node power by 1.44%; accelerating the VEC_INS by 20% leads to the reduction in node power by 1.32% and in runtime by 4.18%.

To reduce branch mispredictions, we inline several procedures, and unroll several loops and eliminate some conditional branches. Mira features a quad floating point unit (FPU) that can be used to execute four-wide SIMD instructions, or two-wide complex arithmetic SIMD instructions. We utilize the quad FPU to accelerate the vector operations using the compiler option –qarch=qp –qsimd=auto, and use up to four OpenMP threads per core for the program executions.

The total energy saved by these optimizations was an average of 15.49% for the problem size of 512x512x512 in Figure 6, where 2048x64 stands for 2048 nodes (1 MPI process per node) with 64 OpenMP threads per node (4 threads per core; 16 cores per node). The average percentage of energy saving (15.49%) is bigger than the percent improvement in runtime (14.85%), meaning that reducing the runtime and power consumption results in a larger energy saving. For the problem size of 128x128x128, the average total energy saved is 26.64%. Overall, the
average total energy saved is 18.28% over two problem sizes of 128x128x128 and 512x512x512 on up to 32,768 cores on Mira.

4.3 eq3dyna on SystemG
For 14 performance counters used in the four models for eq3dyna with the problem size of 200m (element resolution), the ranking algorithm results in the following counters, from highest to lowest: L1_ICM and L2_ICA, L2_DCW, L2_TCW, and TLB_DM. We find the counter L1_ICM is a common counter for the four models, and is a dominant factor in performance. However, L2_ICA only contributes in the runtime model. They are highly correlated because Level 1 instruction cache misses cause the increase of Level 2 instruction cache accesses. The other counters L2_DCW, L2_TCW and TLB_DM are minor. Therefore, based on this information, we will focus our optimization efforts on L1_ICM.

We look at the source code to determine which section of the code contributed significantly to the runtime. We found that two major functions qdct3 and hrqlss are the area of focus, and also found the pattern that many blocks were originally expanding 8x3 to 24x3 sparse blocks, so we rewrote this part of the code so that it no longer does the expanding in order to improve cache locality performance. As a result, more data is able to fit in L1 cache to reduce the cache misses. To further reduce the L1 instruction cache misses, we added the compiler option -fprefetch-loop-arrays to prefetch caches and memory to improve the performance of loops that access large arrays. The total energy saved by these optimizations was an average of 30.67% for eq3dyna with the problem size of 200m on up to 256 cores.

4.4 eq3dyna on Mira
We apply the ranking algorithm to 8 performance counters used in the four models for eq3dyna with the problem size of 100m on Mira, and find VEC_INS is a common dominant factor in runtime, system power and CPU power models. BR_MSP has the second highest rank, and is a dominant factor in memory power model. Therefore, based on this information, we focus the optimization effort on VEC_INS and BR_MSP.

We utilize the quad FPU of BG/Q to accelerate the vector operations using the compiler option -arch=qp -qsimd=auto, and use up to four OpenMP threads per core for the program executions, and fuse several loops, and unroll several loops and eliminate some conditional branches to reduce the BR_MSP. We also remove two MPI process synchronizations to improve the overlapping of computation and communication.

The optimized application was run with the problem size of 100m on up to 4096 cores. The total energy saved by these optimizations was an average of 61.73% in Figure 7. For the problem size of 200m, the total energy saved by these optimizations was an average of 20.61%. Overall, the average total energy saved was 48.65% over two problem sizes of 100m and 200m on up to 4096 cores.

Conclusions
Using our performance counter-based modeling and energy saving framework, we developed a web-based what-if prediction system to predict the outcomes of possible optimizations theoretically, and modeled and improved both the runtime and power consumptions of two large-scale scientific applications eq3dyna and PMLB on two power-aware supercomputers, BG/Q Mira and X86-64 SystemG. Our optimizations resulted in an average energy savings of 18.28% on up to 32,768 cores on Mira and 11.28% on up to 128 cores on SystemG for PMLB. For eq3dyna, an average energy savings were 48.65% on up to 4,096 cores on Mira and 30.67% on up to 256 cores on SystemG. However, setting the environment variable OMP_DYNAMIC=true to apply DCT to the optimized application executions did not improve performance and energy because of the overhead caused by enabling dynamic adjustment of the number of threads.

We believe that this framework can be applied to large-scale scientific applications executed on other architectures including hardware like GPGPUs and many-core accelerators like Intel’s Xeon Phi, and power and performance models of these applications on one architecture can be used to predict the power consumption and performance on large scale systems with similar architectures. Our methodology represents a generalizable approach to comprehensive optimization, focused on the most efficient use of available resources, balancing runtime with power consumption. In this way, we hope our method can provide additional guidance to both application and system developers to develop the next generation of energy-efficient applications and supercomputers.

Acknowledgements
This work is supported by NSF grants CNS-0911023 and DMS-1317131. The authors would like to acknowledge B. Duan from TAMU for providing the earthquake simulations with different problem sizes, K. Cameron at Virginia Tech
for the use of PowerPack and SystemG, Argonne Leadership Computing Facility for the use of BlueGene/Q Mira under DOE INCITE project PEACES, and the reviewers for their valuable comments.

References


Xingfu Wu is a TEES Research Associate Professor in Department of Computer Science & Engineering at Texas A&M University. His research interests are performance evaluation and modeling, power modeling, parallel and cloud computing, and energy-efficient computing. He received a PhD in computer science from Beijing University of Aeronautics and Astronautics (now renamed Beihang University). His monograph: Performance Evaluation, Prediction and Visualization of Parallel Systems, was published by Kluwer Academic Publishers in 1999. He is a senior ACM member and an IEEE Computer Society member. Contact him at wuxf@tamu.edu.

Valerie E. Taylor is a Senior Associate Dean of Dwight Look College of Engineering, Regents Professor, the holder of the Royce E. Wisenbaker Professorship at Texas A&M University. Her research interests are high performance computing, performance evaluation and modeling, power modeling, and energy-efficient computing. She has authored or co-authored over 100 papers in these areas. She received a PhD in Electrical Engineering and Computer Science from the University of California, Berkeley. She is an IEEE Fellow and a member of ACM. Contact her at vtaylor@tamu.edu.

Jeanine Cook is currently a Principal Member of Technical Staff in the Scalable Architectures group at Sandia National Laboratories. Her research interests include processing-in-memory architectures, next-generation memory technologies and subsystems, performance analysis tools for
Exascale systems, and performance modeling and simulation. Prior to joining Sandia, she was an Associate Professor in The Klipsch School of Electrical and Computer Engineering at New Mexico State University. She remains Affiliated Faculty at NMSU, directing the research of several Ph.D students. She was honored with a Presidential Early Career Award in Science and Engineering (PECASE) in 2008 for her work in performance modeling. Contact her at jeacook@sandia.gov.

Philip J. Mucci is currently the president and co-founder of Minimal Metrics, and a research consultant for the Innovative Computing Laboratory at University of Tennessee. His research interests are high performance computing, performance optimization and performance analysis tools. Contact him at phil@minimalmetrics.com.