Utilizing Hardware Performance Counters to Model and Optimize the Energy and Performance of Large Scale Scientific Applications on Power-aware Supercomputers

Xingfu Wu and Valerie Taylor
Department of Computer Science & Engineering
Texas A&M University, College Station, TX 77843
Email: {wuxf, taylor}@cse.tamu.edu

Abstract
Hardware performance counters are used as effective proxies to estimate power consumption and runtime. In this paper we present a performance counter-based power and performance modeling and optimization method, and use the method to model four metrics: runtime, system power, CPU power and memory power. The performance counters that compose the models are used to explore some counter-guided optimizations with two large-scale scientific applications: an earthquake simulation and an aerospace application. We demonstrate the use of the method using two power-aware supercomputers, Mira at Argonne National Laboratory and SystemG at Virginia Tech. The counter-guided optimizations result in a reduction in energy by an average of 18.28% on up to 32,768 cores on Mira and 11.28% on up to 128 cores on SystemG for the aerospace application. For the earthquake simulation, the average energy reductions achieved are 48.65% on up to 4,096 cores on Mira and 30.67% on up to 256 cores on SystemG.

1. Introduction
HPC systems, especially, petaflops supercomputers, consume a large amount of power. Building future exascale HPC systems will be constrained by power and energy consumptions. As such, monitoring power consumption of an HPC system is important for power management. It is important 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 [29, 1, 26].

Hardware performance counters are exposed to the user space on commercial hardware. The performance counters monitor CPU, 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 overhead. In this paper we present a performance counter-based modeling and optimization method, and use this method to model the four metrics: runtime, system power, CPU power and memory power. We explore the counter-guided optimizations with an earthquake simulation eq3dyna [36] and an aerospace application Parallel Multiblock Lattice Boltzmann (PMLB) [37] executed on two power-aware supercomputers Blue Gene/Q system Mira [17] at Argonne National Laboratory (ANL) and SystemG [33] at Virginia Tech.

Our counter-guided optimizations focus on the performance counters that dominate all four metrics, either explicitly or implicitly. The counter-guided optimizations result in a reduction in energy by an average of 18.28% on up to 32,768 cores on Mira and 11.28% on up to 128 cores on SystemG for the aerospace application. For the earthquake simulation, the average energy reductions are 48.65% on up to 4,096 cores on Mira and 30.67% on up to 256 cores on SystemG.

The contributions of this work are as follows:
- We provide a method to identify the performance counters that are important across runtime model and power models for system, CPU and memory.
- We utilize the insight from the counters, that are identified as important across all four models, to explore improvements with application runtime and power consumption so as to reduce overall energy.
- We demonstrate that our counter-guided optimizations result in performance and energy saving for the applications executed on two power-aware systems.

The remainder of this paper is organized as follows. Section 2 describes some related work. Section 3 discusses hardware performance counters and power profiling environments used in this paper. Section 4 presents a modeling and optimization method and a performance counter-based modeling system, and discusses the ranking and correlation analysis of the counters from runtime and power models. Section 5 uses two case studies to identify the most important counters for application optimization, and explore counter-guided optimizations in energy and performance. Section 6 concludes the paper and discusses future work. Note that all of our data was collected using our MuMMI instrumentation tool, MAIDE [18, 38], which was ported to support BlueGene/Q systems recently. In this
saving energy can be classified into three categories: 1) reduce time and power [4, 5, 32, 10, 14]; 2) reduce time and increase power [31, 10, 13]; 3) reduce power and increase time [18, 8, 9, 10]. Overall, as long as reduced percentage (time/power) is equal or bigger than increased percentage (power/time), energy saving can be obtained.

3. Hardware Performance Counters and Power Profiling Environments

3.1 Power-aware Supercomputers
In this work, we use two power-aware supercomputers Mira at ANL and SystemG at Virginia Tech to conduct our experiments. Mira [17], an IBM Blue Gene/Q supercomputer, is equipped with 786,432 cores, and has a peak performance of 10 petaflops. Each node of Mira has a PowerPC A2 1.6GHz processor containing 16 cores with 4 hardware threads supported per core running at 1.6 GHz, 32MB of shared L2 cache, and 16GB of main memory. The Blue Gene/Q features a quad floating point unit (FPU) that can be used to execute scalar floating point instructions, four-wide SIMD instructions, or two-wide complex arithmetic SIMD instructions. The compilers mpixlc_r and mpixlfr (version 14.1) are used to compile C and Fortran programs.

SystemG [33] is a 22.8 TFLOPS research platform that utilizes 325 Mac Pro computer nodes. Each node of SystemG contains two quad-core 2.8 GHz (adjusted to 2.4GHz) Intel Xeon E5462 processors and 8 GB of main memory. The compilers mpicc (gcc) and mpif90 (gfortran) are used for compiling C and Fortran programs.

3.2 Power Measurement Tools
Currently, there exist several research power measurement tools such as PowerPack [7], or PowerMon2 [11], PowerInsight [12], and vendor-specific power management techniques such as IBM EMON API on BlueGene/Q [2], Intel RAPL [27], and NVIDIA’s Management Library for power measurement [20]. In this paper, we use PowerPack to measure the power consumption for the node, CPU, memory, hard disk, and motherboard on SystemG; we use MonEQ [35], which is application-level power profiling tool based on IBM EMON API for BlueGene/Q systems, to measure the power consumption for the node, CPU, memory, network, etc. at the node-card level on Mira. Each node-card consists of 32 nodes. To obtain the power consumption at the node level, we calculate the average power consumption by dividing by 32, and conduct our experiments on multiple of 32 nodes (number of node-cards) to obtain the power profiling data.

3.3 Hardware Performance Counter Tools
There exist several tools to provide user-level APIs to measure performance counters, such as PAPI [21], perf_events [22], HPM [24], perfmon [23], and so on. In
this paper, we use PAPI, which provides a standardized interface to hardware performance counters.

On SystemG, we use 39 available counters listed below (without PAPI in this paper): L1_DCM, L1_ICM, L2_ICM, L1_TCM, L2_TCM, CA_SHR, CA_CLN, CA_ITV, TLB_DM, TLB_IM, L1_LDM, L1_STM, L2_STM, HW_INT, BR_CN, BR_TKN, BR_NTK, BR_MSP, TOT_IIS, TOT_INS, FP_INS, LD_INS, SR_INS, BR_INS, VEC_INS, RES_STL, TOT_CYC, L1_DCA, L2_DCR, L2_DCW, L1_ICA, L2_ICA, L2_TCA, L2_TCW, FML_INS, FDV_INS, FP_OPS, VEC_SP, and VEC_DP.

On Mira, we use 28 available counters listed below: L1_DCM, L1_ICM, TLB_DM, TLB_IM, L1_LDM, L1_STM, L1_ICR, L1_DCW, L1_DCR, HW_INT, BR_CN, BR_TKN, BR_NTK, BR_MSP, TOT_IIS, TOT_INS, FP_INS, LD_INS, SR_INS, BR_INS, VEC_INS, RES_STL, FP_STAL, TOT_CYC, FML_INS, FDV_INS, FP_OPS, and SYC_INS.

4. Modeling and Optimization Method

During an application execution we capture available performance counters as mentioned in the previous section. All performance counters are normalized using the total execution cycles of the execution to create performance event rates for each counter. Starting with the data for the 39 counters, the next step entails using Spearman correlation and principal component analysis (PCA) to identify the major counters for the four metrics. Then we use a non-negative multivariable regression analysis to generate the four models based upon the set of major counters and CPU frequencies. Our previous work [14, 38] indicates the runtime and power models are accurate with prediction error rate less than 7% in average for six scientific applications.

In this paper we build upon the previous modeling work to present a counter ranking method to identify the most important performance counters, then utilize these counters that can provide some insights for modifying the application to improve the runtime and power consumptions.

4.1. A Power and Runtime Modeling System

MuMMI provides the web-based automated modeling system used to generate the runtime and power models, from which the important counters are identified and used to provide areas for application optimizations. The MuMMI interface is a set of database-driven dynamic web pages. When a user requests to generate an analytical model and plots for an application through a web browser, the request is sent to the web server. In turn, the web server passes the request to a Ruby On Rails [28] web application, which returns a result to the server, which in turn downloads the result onto the user's browser. When the web application processes the request, it executes various SQL queries, processes and formats the data, feeds the data to Octave [25] to generate an analytical model and plots, then sends the result back up through the server, and is downloaded on the client's browser. This method allows for dynamic HTML pages to be generated based on user requests, and does not require hard coded design into a document. This automates the modeling process for generating application runtime model and power models for system, CPU and memory.

Figure 1. Snapshot for four models of PMLB

For instance, given the aerospace application PMLB [37], our online performance counter-based modeling system generates four models (runtime, system power, CPU power and memory power) of the application using 15 different performance counters on SystemG with two CPU frequencies (2.4, 2.8GHz) as shown in Figure 1.

4.2. Counter Correlation Analysis and Ranking

Once we have the models for the four metrics (runtime, system power, CPU power and memory power), we identify the performance counters that are important across the multiple metrics to guide application improvements.

Figure 2. Counter ranking algorithm

The ranking algorithm in Figure 2 is as follows. First, we form a counter list consisting of the counters with the highest coefficient percentage (the ratio of the coefficient to the sum of 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 minor counters with the percentage less than 1% from the counter list. Finally, we analyze the correlations among these counters using pair-wise Spearman correlation to identify the most important counters that significantly contributes to the runtime and power models to form the final counter list. If a counter with higher rank is highly correlated with a counter with lower rank, the counter with lower rank is eliminated from the counter list. This is the group of performance counters we identify for application improvements. We illustrate the use of this counter ranking algorithm in the next section.

5. Case Studies: Counter-guided Optimizations in Energy and Performance

In this section, we use two scientific applications: the parallel earthquake simulation eq3dyna [36] and the parallel aerospace application PMLB [37], to discuss counter-guided optimizations.

5.1 Case Study: PMLB

In the aerospace application PMLB [37], we use the D3Q19 lattice model (19 velocities in 3D) with the collision and streaming operations. The code is written in C, MPI and OpenMP. The fixed problem size is the 3D mesh with 128x128x128 executed on up to 128 cores on SystemG and with 128x128x128 and 512x512x512 executed on up to 32,768 cores on Mira.

5.1.1 Optimizations on SystemG

Figure 3 shows the performance counter ranking for four models with 15 different counters for the original PMLB code executed on SystemG, where the coefficient percentages for TLB.IM, TLB.DM, L2.ICM and L1.ICM are 64.29%, 14.03%, 10.49% and 9.75% for the runtime model; the coefficient percentages for VEC_INS and CA_SHR are 76.64% and 22.45% for the system power model; the coefficient percentages for VEC_INS and BR_NTK are 99.15% and 0.81% for the CPU power model; the coefficient percentages for VEC_INS and CA_CLN are 83.91% and 13.74% for the memory power model. We apply the ranking algorithm as shown in Figure 3 to the four groups of counters from the four models, and have the group of counters with the ranking order from high to low as follows: TLB.IM, VEC_INS, TLB.DM, and L2.ICM.

TLB.IM (instruction translation lookaside buffer (TLB) misses) has the highest rank, and VEC_INS (vector/SIMD instructions) has the second highest rank. We use pair-wise Spearman correlation to analyze the correlations between the two counters as follows:

TLB.IM: Occurred in Runtime
TLB.DM: Corr Value=0.89217296 : Occurred in Runtime
L2.ICM: Corr Value=0.88451013 : Occurred in Runtime
VEC_INS: Occurred in System, CPU, Memory

We found the counter TLB.IM only occurs in the runtime model and are correlated with TLB.DM (data translation lookaside buffer miss) and L2.ICM (L2 instruction cache misses), however, VEC_INS occurs in the models of system power, CPU power and memory power, and does not have any correlated counters. Both TLB.IM and VEC_INS are not correlated each other. Therefore, based on the information, we focus on the counters TLB.IM and VEC_INS for optimization on SystemG.

Figure 5. Energy per node on SystemG.

The current Linux system on SystemG supports two page sizes, 4KB as default and 2MB. To reduce TLB miss, the main kernel address space is mapped with huge pages -- a single 2MB huge page only requires a single TLB entry while the same memory, in 4KB pages, would need 512 TLB entries. We utilized the 2MB huge pages for the application execution using libhugetlbfs [16] to reduce the TLB misses. We also vectorized the code, and used the compiler option -ftree-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 optimized results in Figure 4 show that the application runtime improves by average 11.23% as shown in Figure 4(a), the system power consumption increases by only average 0.01% as shown in Figure 4(b), the CPU power consumption increases by average 2.13% as shown in Figure 4(c), and the memory power consumption increases.

Figure 3. Ranking for the original PMLB on SystemG

Figure 4. Optimizations on SystemG.
by only average 0.61% as shown in Figure 4(d). Overall, the total energy saves by the average 11.28% as shown in Figure 5. The average energy saving percentage (11.28%) is bigger than the runtime improvement percentage (11.23%), and this means that reducing the runtime and power consumption results in more energy saving.

Figure 4. Runtime and Average Power Comparison on SystemG

Figure 7. Runtime and Average Power Comparison on Mira

5.1.2. Optimizations on Mira

Figure 6 shows the performance counter ranking for four models with 10 different counters for the original code executed on Mira, where the coefficient percentages for HW_INT and FDV_INS are 59.57% and 40.42% for the Runtime model; the coefficient percentages for BR_MSP, BR_NTK, SR_INS, RES_STL, and FP_INS are 51.54%, 36%, 8.80%, 1.92%, and 1.74% for the system power model; the coefficient percentages for VEC_INS, BR_MSP, SR_INS, FML_INS, and BR_NTK are 42.98%, 39.94%, 9.51%, 4.33%, and 1.90% for the CPU power model; the coefficient percentages for L1_ICM, BR_NTK and BR_MSP are 72.91%, 19.47% and 7.25% for the memory power model. We apply the ranking algorithm to the four groups of counters from the four models, and have the group of counters with the ranking order from high to low as follows: HW_INT, BR_MSP, VEC_INS, L1_ICM, FDV_INS and BR_NTK.

HW_INT (number of hardware interrupts) has the highest rank, and BR_MSP (conditional branch instructions mispredicted) has the second highest rank. We use pair-wise Spearman correlation to analyze the correlations between the two counters as follows:

We found the counter HW_INT only occurs in the Runtime, and does not have any correlated counters, however, BR_MSP are correlated with the counters L1_ICM (L1 instruction cache misses), VEC_INS and BR_NTK (conditional branch instructions not taken) in the
final counter list, and VEC_INS are correlated with other counters BR_MSP, L1_ICM, FDV_INS (floating point divide instructions) and BR_NTK in the final counter list as well, where 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 the information, we focus on the counters BR_MSP and VEC_INS for optimizations on Mira.

- Figure 8. Energy for PMLB with 512x512x512 on Mira

- Figure 9. Energy for PMLB with 128x128x128 on Mira

For the application code, we inline several procedures, and unroll several loops and eliminate some conditional branches to reduce the number of BR_MSP. As discussed in Section 3.1, 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. To take advantage of vector instructions supported by BG/Q processors, we add the more compiler options -qarch=qp -qsimd=auto to compile the revised code, and utilizes up to four hardware threads supported by each core for the program executions.

The optimized application was run with a problem size of 512x512x512 on up to 32,768 cores (2048 nodes with 16 cores per node). The application runtime improves by average 14.85% as shown in Figure 7(a), where 2048x64 stands for 2048 nodes (1 MPI process per node) with 64 OpenMP threads per node (4 threads per core). The system power consumption increases by only average 0.43% as shown in Figure 7(b), the CPU power consumption increases by only average 0.27% as shown in Figure 7(c), and the memory power consumption increases by average 2.92% as shown in Figure 7(d). The total energy saves by the average 15.49% for the application with the problem size 512x512x512 as shown in Figure 8. The average energy saving percentage (15.49%) is bigger than the runtime improvement percentage (14.85%), and this means that reducing the runtime and power consumption results in the larger energy saving. The total energy saves by the average 26.64% for the application with the problem size 128x128x128 as shown in Figure 9.

Overall, the total energy saves by the average 18.28% for the application over the problem sizes 128x128x128 and 512x512x512 on up to 32,768 cores, and utilizing all four hardware threads per core not only reduces the runtime but also saves the energy of the optimized code because of better pipelining of the quad FPU.

5.1.3 Architecture Comparison
For the same PMLB executed on SystemG and Mira, it is interesting to see that there are five common counters VEC_INS, L1_ICM, SR_INS, RES_STL and BR_NTK across both architectures, and only is VEC_INS dominated in the models. From the final counter lists on SystemG and Mira, we observe that TLB and VEC_INS are main factors for the four models on SystemG, however, on Mira, the branch instructions (BR_MSP and BR_NTK) and VEC_INS are the main factors for these models. Notice that TLB is not a main performance factor on Mira, because the static memory map on a BlueGene/Q system is a translation between virtual addresses into physical addresses in the DDR3 memory [GM13].

5.2 Case Study: eq3dyna
The earthquake simulation eq3dyna [36] is a parallel finite element simulation of dynamic earthquake rupture along geometrically complex faults. The code is written in Fortran90, MPI and OpenMP. In this paper, we focus on the simulation with a fixed problem size of 200 meters (element resolution) executed on up to 256 cores on SystemG, and the simulation with fixed problem sizes of 200 meters and 100 meters executed on up to 4096 cores on Mira.

5.2.1 Optimizations on SystemG
Figure 10 shows the performance counter ranking for four models with 14 different counters on SystemG, where the coefficient percentages for L1_ICM, L2_ICA, and L2_DCW are 62.83%, 33.59% and 3.58% for the runtime model; the coefficient percentages for L1_ICM and L2_DCW are 97.87% and 1.54% for the system power model; the coefficient percentages for L1_ICM and TLB_DM are 97.84% and 1.37% for the CPU power model; the coefficient percentages for L1_ICM and L2_STM are 93.62% and 3.82% for the memory power model. We apply the ranking algorithm to the four group of counters from the four models, we have the group of counters with the ranking order from high to low as follows: L1_ICM, L2_ICA, L2_DCW, L2_TCW, and TLB_DM.
Figure 10. Counter ranking for eq3dyna on SystemG

L1_ICM (L1 instruction cache misses) has the highest rank among these counters, and L2_ICA (L2 instruction cache accesses) has the second highest rank. We use pairwise Spearman correlation to analyze the correlations between the two counters as follows:

$L1_ICM$: Occurred in Runtime, System, CPU, Memory
$L2_ICA$: Corr Value=0.96165995 : Occurred in Runtime

We found the counter L1_ICM is a common counter for the four models, and is a dominated factor (more than 93%) in three power models, however L2_ICA only occurs in the runtime model. They are highly correlated with the value of 0.96165995 because Level 1 instruction cache misses cause the increase of Level 2 instruction cache accesses. From the counter list described above, the other counters L2_DCW, L2_TCW and TLB_DM are not dominated shown in Figure 13. Therefore, based on the information, we will focus on the counter L1_ICM for optimization.

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 hrglss are the area of focus. After we carefully analyzed the data layout and memory footprint of the application, we found the pattern that many blocks were originally expanding 8x3 to 24x3 sparse blocks, so we rewrote the part of the original code so that it no longer does the expanding in order to improve cache locality performance. As a result of using smaller 8x3 blocks, 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.

Figure 11. Runtime and Average Power Comparison on SystemG

Figure 12. Energy per node (J) on SystemG

Figure 11 shows the runtime and power comparison between the original and optimized codes for CPU frequency of 2.8 GHz. We achieved average 31.65% performance improvement on up to 256 cores as shown in Figure 11(a) because of significant improvement of cache locality performance for the code. Notice that the application runtime on 4 cores increases a little bit because of the L2 cache contention caused by 4 OpenMP threads dispatched on the same quad-core chip.

For power consumption, the system power consumption increases by average 1.42% on up to 256 cores shown in Figure 11(b). This increase is caused by busier CPU units. Let’s look at the details of CPU and memory power consumptions shown in Figures 11(c) and 11(d). Figure 11(c) indicates that the optimized code keeps CPU units busier than the original one, and the CPU power increases by average 2.23% on up to 256 cores. When the number of cores increases to 8 cores (8 cores per node), the CPU power consumption per node reaches to the highest, then because the workload per node decreases with increasing the number of nodes, the CPU power decreases, however, our optimizations improve the cache locality so that it results in less memory power consumption as shown in Figure 11(d). The memory power consumption decreases by average 0.62%. Overall, our optimizations for the application with the problem size 200m save energy by average 30.67% on up to 256 cores as shown in Figure 12.
5.2.2. Optimizations on Mira

We ran the optimizations code discussed in Section 5.2.1 on Mira, and found there is only less than 10% performance improvement. So we further optimize the code.

Figure 13 shows the performance counter ranking for four models with 8 different counters on Mira, where the coefficient percentages for VEC_INS, BR_MSP, LD_INS are 89.66%, 5.97% and 3.47% for the runtime model; the coefficient percentages for VEC_INS and LD_INS are 99.80% and 0.14% for the system power model; the coefficient percentages for SR_INS, BR_MSP and RES_STL are 49.32%, 45.76% and 4.51% for the memory power model. We apply our ranking algorithm to the four groups of counters, we have the group of counters with the ranking order from high to low as follows: VEC_INS, BR_MSP, SR_INS, LD_INS and RES_STL.

We found the counter VEC_INS is a common counter for three of the four models, and is highly correlated with minor counters L1_DCM and L1_STM. Therefore, based on the information, we will focus on the counter VEC_INS for optimization.

From the counter list described above, the counter BR_MSP has the second highest rank, is a dominated factor in memory power model. We also use pair-wise Spearman correlation to analyze the correlation values between counters occurred in the models as follows:

- BR_MSP: Occurred in Runtime, Memory
- SR_INS: Corr Value=0.87258065: Occurred in Memory
- BR_NTK: Corr Value=0.99919355: Occurred in System, CPU
- RES_STL: Corr Value=−0.88225806: Occurred in Runtime, Memory

The counter occurs in the models of Runtime and Memory Power, and is correlated with SR_INS (store instructions), BR_NTK and RES_STL (cycles stalled on any resource). So we also consider the counter BR_MSP for optimization.

To take advantage of vector instructions supported by BG/Q A2 processors, we add the compiler options –qarch=qp –qsimd=auto to compile the optimized code, and utilizes up to four hardware threads supported by core for the program executions, and fuse several loops, and unroll several loops and eliminate some conditional branches to reduce the number of BR_MSP.

Based on the power profiling data for network per node (for the original) shown in Figure 14, we found that the power for network per node is decreased with increasing the number of nodes (1 MPI process per node), however, as we know, the communication overhead should increase. Then we found two MPI process synchronizations caused the waiting for MPI processes. During the parallel 3D mesh partitioning, we let one MPI process use its sub-mesh domain as long as it is ready in order to avoid the waiting for all sub-mesh domains ready. During boundary data exchange, as long as one MPI process receives the data, it correlations between the two counters in the models as follows:

- VEC_INS: Occurred in Runtime, System, CPU
  L1_DCM: Corr Value=0.94314516: Occurred in Memory
  L1_STM: Corr Value=0.91893161: Occurred in Runtime

We found the counter VEC_INS is a common counter for three of the four models, and is highly correlated with minor counters L1_DCM and L1_STM. Therefore, based on the information, we will focus on the counter VEC_INS for optimization.

From the counter list described above, the counter BR_MSP has the second highest rank, is a dominated factor in memory power model. We also use pair-wise Spearman correlation to analyze the correlation values between counters occurred in the models as follows:

- BR_MSP: Occurred in Runtime, Memory
- SR_INS: Corr Value=0.87258065: Occurred in Memory
- BR_NTK: Corr Value=0.99919355: Occurred in System, CPU
- RES_STL: Corr Value=−0.88225806: Occurred in Runtime, Memory

The counter occurs in the models of Runtime and Memory Power, and is correlated with SR_INS (store instructions), BR_NTK and RES_STL (cycles stalled on any resource). So we also consider the counter BR_MSP for optimization.
calls faulting subroutine to revise residual force on its domain in order to significantly increase the overlapping of computation and communication. The optimized result for network power in Figure 14 shows the optimized keeps the network busy during the program execution.

![Figure 16. Energy per node for eq3dyna with 100m](image)

![Figure 17. Energy per node for eq3dyna with 200m](image)

The optimized results for the application with the problem size 100m on up to 4096 cores in Figure 15 show that the application runtime improves by average 62.71% as shown in Figure 5(a), the system power consumption increases by average 4.42% as shown in Figure 15(b), the CPU power consumption increases by average 4.13% as shown in Figure 15(c), and the memory power consumption increases by average 8.76% as shown in Figure 15(d). The total energy saves by the average 61.73% for the application with the problem size 100m as shown in Figure 16. The total energy saves by the average 20.61% for the application with the problem size 200m as shown in Figure 17.

Overall, the total energy saves by the average 48.65% for the application with the problem sizes 100m and 200m on up to 4096 cores, and utilizing all four hardware threads per core not only reduces the runtime but also saves the energy of the optimized code because of better pipelining of the quad FPU.

5.2.3. Architecture Comparison
For the same earthquake simulation executed on SystemG and Mira, it is interesting to see that there are three common counters SR_INS, RES_STL and BR_NTK across the two architectures, and they are not dominated in the models. From the final counter lists for SystemG and Mira, we observe that L1 and L2 caches are main factors for the four models on SystemG, however, on Mira, the VEC_INS and BR_MSP are the main factors for the four models. Therefore, we have to consider different main factors to optimize the application on different architectures.

6. Conclusions
In this paper, we presented a performance counter-based modeling and optimization method, and used the method to model the runtime and power consumptions of two large-scale scientific applications eq3dyna and PMLB on two power-aware supercomputers, Mira and SystemG. These models focus on four metrics: runtime, system power, CPU power and memory power. We ranked the counters from the models, and analyzed the correlations among the counters to identify the most important counters for application optimizations. The resultant counters were used to provide the insights to explore counter-guided application optimizations on Mira and SystemG. The counter-guided optimizations resulted in a reduction in energy by an average of 18.28% on up to 32,768 cores on Mira and 11.28% on up to 128 cores on SystemG for the aerospace application. For the earthquake simulation, the average energy reductions were 48.65% on up to 4,096 cores on Mira and 30.67% on up to 256 cores on SystemG. We also found, 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 the counter-based performance and power modeling and optimization method can be applied to large-scale scientific applications executed on other architectures such as GPU and Xeon Phi by utilizing counters for GPU and Xeon Phi, and can provide a guidance that can be used by system and application developers for energy-efficient development.

Acknowledgements
This work is supported by NSF grants CNS-0911023 and DMS-1317131. The authors would like to acknowledge B. Duan from Dept. of Geophysics and Geology, TAMU for providing the earthquake simulations with different problem sizes, A. Salazar for collecting the data for the earthquake simulation on SystemG, and K. Cameron’s group for the use of PowerPack, R. Hunter for support of SystemG, and J. Love from WisdomGroup for developing the MuMMI modeling web interfaces, and S. Wallace from Illinois Institute of Technology for providing the latest MonEQ, and Argonne Leadership Computing Facility for the use of BlueGene/Q Mira under DOE INCITE project PEACES.

References


