Parallel Computing 57 (2016) 73–86

Contents lists available at ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

Application power profiling on IBM Blue Gene/Q

Sean Wallace[a,*], Zhou Zhou[a], Venkatram Vishwanath[b], Susan Coghlan[b], John Tramm[b], Zhiling Lan[a], Michael E. Papka[b,c]

[a] Illinois Institute of Technology, Chicago, IL, USA
[b] Argonne National Laboratory, Argonne, IL, USA
[c] Northern Illinois University, DeKalb, IL, USA

ARTICLE INFO

Article history:
Received 16 March 2015
Revised 22 April 2016
Accepted 30 May 2016
Available online 3 June 2016

Keywords:
Power profiling
Energy efficiency
Blue Gene/Q
Power performance analysis

ABSTRACT

The power consumption of state of the art supercomputers, because of their complexity and unpredictable workloads, is extremely difficult to estimate. Accurate and precise results, as are now possible with the latest generation of IBM Blue Gene/Q, are therefore a welcome addition to the landscape. Only recently have end users been afforded the ability to access the power consumption of their applications. However, just because it's possible for end users to obtain this data does not mean it's a trivial task. This emergence of new data is therefore not only understudied, but also not fully understood.

In this paper, we describe our open source power profiling library called MonEQ, built on the IBM provided Environmental Monitoring (EMON) API. We show that it's lightweight, has extremely low overhead, is incredibly flexible, and has advanced features which end users can take advantage. We then integrate MonEQ into several benchmarks and show the data it produces and what analysis of this data can teach us. Going one step further we also describe how seemingly simple changes in scale or network topology can have dramatic effects on power consumption. To this end, previously well understood applications will now have new facets of potential analysis.

1. Introduction

As the field of computational science continues to push towards the exascale era of supercomputing, power consumption has emerged as an increasingly vital area of research. The leading system on the November 2014 Top500 list [1], Tianhe-2, achieves 33.8 PFlops Rmax while consuming 17.8 MW of power. The US Department of Energy has set a goal of 20 MW for exascale systems [2]. This implies that to achieve exascale, current supercomputers will need to scale their performance by $\sim$ 60X while increasing their power consumption by just $\sim$ 2X – a challenging task.

The majority of performance studies conducted on large-scale high performance computing (HPC) systems (including those published by the Green500 [3]) focus on the Flops/Watt metric. While this metric is useful for measuring the energy consumption of a particular architecture, it fails to convey much information about the individual components and how they affect the system on the whole. Moreover, it is also recognized that a different metric to measure energy consumption, namely *time to solution*, would likely result in different rankings [4]. Therefore, the analysis of power consumption on

* Corresponding author.
E-mail addresses: swallac6@iit.edu (S. Wallace), zzhou1@iit.edu (Z. Zhou), venkat@anl.gov (V. Vishwanath), smc@anl.gov (S. Coghlan), jtramm@mcs.anl.gov (J. Tramm), lan@iit.edu (Z. Lan), papka@anl.gov (M.E. Papka).

http://dx.doi.org/10.1016/j.parco.2016.05.015
0167-8191/

state-of-the-art supercomputers such as the Blue Gene/Q is imperative to better understand how these new systems differ from their predecessors and in which direction key characteristics are heading.

Going further, while it's one thing to understand the current generation of systems and the hardware they contain, it is another thing entirely to imagine what the next generation might bring. Aspects such as the power distribution, what the applications running on those systems will have in terms of power requirements, and even basics of the system design such as the network topology could have potentially dramatic effects on different classes of applications. As we will highlight in this work, some applications can experience dramatic changes in time to solution (and therefore energy consumption) when something as simple as the network topology is modified.

Recognizing the importance of fine-grained power measurement, hardware vendors have started to deploy environmental sensors in various locations inside the computer. These sensors report physical readings such as motherboard, CPU, GPU, hard disk and other peripherals' temperature, voltage, current, fan speed, etc. Depending on the systems, these environmental data can be accessed via different facilities in an "out-of-band" fashion. In the case of the Blue Gene/Q (BG/Q) there are two methods by which environmental data can be accessed. One is an environmental database and the other is vendor-supplied application programming interfaces (APIs) to profile power usage through the code of jobs actually running on the system. They provide information about the power consumption at two different scales.

The environmental database is maintained primarily to help identify and eliminate insufficient cooling as well as inadequate distribution of power. The power profiling APIs, on the other hand, provide power consumption data directly to the running process across more power domains with respect to components and at much finer granularity with respect to time. Unfortunately, these APIs only provide the total power consumption of all domains and are quite complicated for application developers to use.

In this work we describe our open source power profiling library called MonEQ that is designed to address the aforementioned issues of the vendor-supplied APIs. By means of parallel benchmarks and applications, we further present extensive power performance studies using MonEQ on the production 48-rack BG/Q system at Argonne named Mira. We present the details of our profiling library. We show that it's lightweight, has extremely low overhead, is incredibly flexible, and has more advanced features such as tagging which allows for detailed profiling of tagged sections of code. Furthermore, We discuss interesting power performance studies by using MonEQ with a number of parallel benchmarks and applications. Our analysis is conducted in two forms: first, we show the complete range of results that users can expect and how they compare to other applications; second, we evaluate what happens to applications when they are run at different scales and with different network topologies.

The second form of our analysis intends to answer several key questions. Does the application itself play a significant role? Some applications are very compute intensive, will they use more power than applications which aren't? What about the network topology? While many applications run on modern day supercomputers are finely tuned to take advantage of the complex networks, in terms of power consumption does it really make that big of a difference? Finally, does the scale at which an application is run affect power consumption? For applications which show linear speedup with respect to scale, is the power consumption at one scale different than another? The answers to these questions are important not just for application developers who need to know how their application will perform, but they also have implications in power aware scheduling.

The remainder of this paper is as follows: An overview of the IBM Blue Gene architecture as well as the environmental data collection of the system is presented in Section 2. Section 3 provides a detailed description of our profiling library, MonEQ. Section 4 presents a detailed representation of what data is obtained from profiling an application with MonEQ as well as analysis of this data. The effects of running codes at different scales under different network configurations as well as the answers to the questions we asked are presented in Section 5. The related work is provided in Section 6. Finally, we will provide our conclusions in Section 7.

2. Blue Gene/Q architecture and environmental data collection

The Blue Gene/Q architecture is described in detail in [5]. Our analysis of power usage on BG/Q is based on Argonne National Laboratory's 48-rack BG/Q system, Mira. A rack of a BG/Q system consists of two midplanes, eight link cards, and two service cards. A midplane contains 16 node boards. Each node board holds 32 compute cards, for a total of 1024 nodes per rack. Each compute card has a single 18-core PowerPC A2 processor [6] (16 cores for applications, one core for system software, and one core inactive) with four hardware threads per core, with DDR3 memory. BG/Q thus has 16,384 cores per rack. Fig. 1 depicts how power is distributed from the power substation to the Mira BG/Q system. It shows the various measurement points along this path where we can monitor the power consumption.

In each BG/Q rack, bulk power modules (BPMs) convert AC power to 48 V DC power, which is then distributed to the two midplanes. Blue Gene systems have environmental monitoring capabilities that periodically sample and gather environmental data from various sensors and store this collected information together with the timestamp and location information in an IBM DB2 relational database – commonly referred to as the environmental database [7]. These sensors are found in locations such as service cards, node boards, compute nodes, link chips, BPMs, and the coolant environment. Depending on the sensor, the information collected ranges from various physical attributes such as temperature, coolant flow and pressure, fan speed, voltage, and current. This sensor data is collected at relatively long polling intervals (about 4 min on average but can be configured anywhere within a range of 60-1800 s), and while a shorter polling interval would be ideal, the resulting

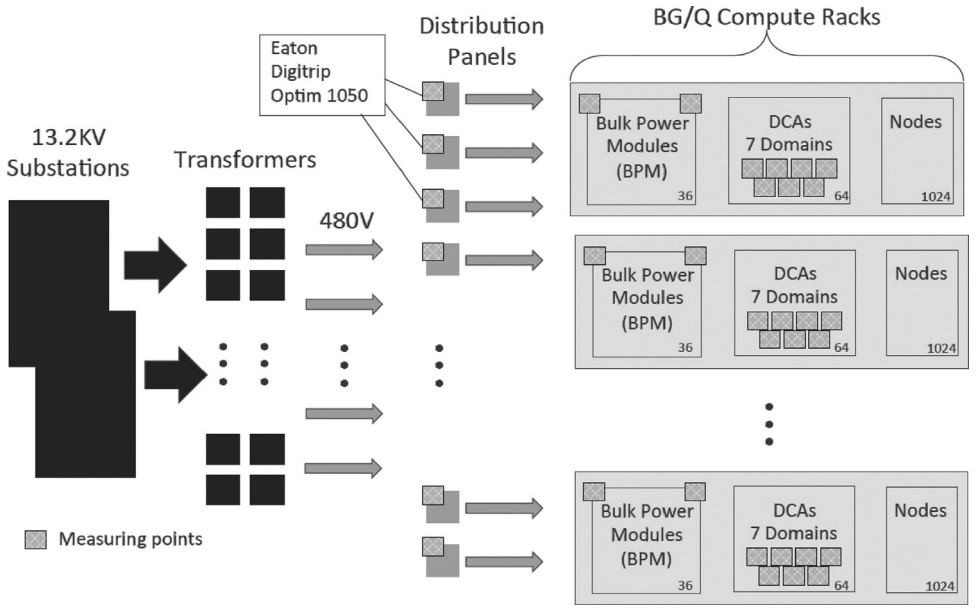

Fig. 1. Mira 480V compute power distribution system.

Table 1
Node board power domains.

Domain ID	Description
1	Chip core voltage
2	Chip memory interface and DRAM voltage
6	HSS network transceiver voltage compute+link chip
7	Chip SRAM voltage
3	Optics
4	Optics + PCIExpress
8	Link chip core

volume of data alone would exceed the server's processing capacity. The Blue Gene environmental database stores power consumption information (in watts and amperes) in both the input and output directions of the BPM.

3. MonEQ: a user-level power profiling library for application users

For BG/Q, IBM provides new interfaces in the form of an environmental monitoring API called EMON that allows one to access power consumption data from code running on compute nodes, with a relatively short response time. However, EMON by itself is insufficient for our needs. The power information obtained using EMON is *total* power consumption from the oldest generation. Furthermore, the underlying power measurement infrastructure does not measure all domains at the exact same time. This may result in some inconsistent cases, such as the case when a piece of code begins to stress both the CPU and memory at the same time. In this case, we might not see an increase in power for both domains in the same generation of data if there is a gap in time between when the CPU and memory power are measured. There is active research by IBM to improve the power monitoring systems, so this problem may change in the future.

To get past this limitation, we designed, MonEQ[1], a "power profiling library" that allows us to read the individual voltage and current data points for each of the domains in a BG/Q nodecard. The exact domains and their corresponding BG/Q IDs are displayed in Table 1. While the astute reader will notice there isn't a 5th BG/Q ID listed, it is not an omission on our part, these IDs are obtained directly from the IBM Redbook for BG/Q [8]. It's also worth noting that while this work is concerned with the BG/Q, MonEQ is designed to work with many other systems including NVIDIA GPUs, Intel CPUs, and the Xeon Phi. More information on this can be found in our related work [9].

In its default mode, MonEQ will pull data from the EMON API every 560 ms, which is currently the lowest polling interval possible; however users have the ability to set this interval to whatever valid value is desired. With the value of the polling interval set, MonEQ then registers to receive a `SIGALRM` signal at that polling interval. When the signal is delivered, MonEQ

[1] Available at: https://repo.anl-external.org/repos/PowerMonitoring/trunk/bgq

```
int status, myrank, numtasks, itr;

status = MPI_Init(&argc, &argv);

MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

status = MonEQ_Initialize();      // Setup Power

/* User code */

status = MonEQ_Finalize();        // Finalize Power

MPI_Finalize();
```

Listing 1. Simple MonEQ example.

```
mpi_wtime,nodecard,chip_core,dram,network,sram,optics,pciexpress,link_chip_core
6.0483130,1743.9889,915.7174,417.1284,53.1389,57.5081,208.6086,52.5118,39.3755
6.6082960,1585.8319,922.7428,252.8799,53.1389,57.1425,208.4763,52.0234,39.4279
7.1682930,1701.5392,924.2439,366.8283,53.1942,57.5481,208.3977,51.9439,39.3831
7.7283010,1592.2231,929.1704,253.8852,52.8120,57.6001,207.4297,51.5351,39.7906
8.2882880,1585.3076,923.7773,251.8973,52.8099,57.1292,208.6797,51.6920,39.3222
```

Listing 2. Sample output from one of the output CSV files. Note that several columns pertaining to date/time and location have been omitted.

calls down to the EMON API and records the latest generation of power data available in an array local to agent rank of the node card.

One limitation of the EMON API is that it can only collect data at the node card level (every 32 nodes). At first this might seem rather significant, however given the scale at which most applications are run (i.e., some number of racks) and the nature of large scale jobs, having 32 data points per rack per time step is usually sufficient. This limitation is part of the design of the system; as such, it is not possible to change the number of nodes for which data is collectible in software.

If the "automatic" mode of MonEQ has not been disabled, the collection will continue on in this fashion collecting power data for all of the domains from the initialization call (usually called shortly after `MPI_Init`) to the finalize call (usually called shortly before `MPI_Finalize`). If desired, the `MPI_Init` and `MPI_Finalize` calls can be overloaded to include automatic calls to MonEQ making integration completely automatic.

As soon as finalize is called on MonEQ, it proceeds to dump all of the data it has collected in its working array to CSV files. A simple example of how MonEQ is implemented is shown in Listing 1. A sample of the data that would be output to CSV file from this simple example is available in Listing 2. Since one of the primary goals of MonEQ is to be scalable, it is of the utmost importance that the data generated by one node card matches up exactly with the data from another node card in terms of time. Fortunately, the BG/Q is designed with very accurate system-wide accounting built-in via the clock card hardware which means that at least on this system, keeping data properly aligned is relatively simple.

Before collecting data the user can specify how the output files should be partitioned according to two modes. In the first mode, the number of CSV files will depend on the number of node cards utilized by the application (i.e., one per node card). In the second, the user can specify to have output files generated based on the ranks of the MPI program. Thus, if it was desirable to have the power data from ranks 1–10 reported in one file and ranks 11–32 in another, that is possible with MonEQ. It should be noted this does not get around the "one data point per node card per unit of time" limitation. If this feature is utilized, MonEQ will bunch ranks running on a node card together just the same as it always would, but upon call of the finalize method data is aggregated for the appropriate ranks into a single file. That is, if ranks 1-32 are running on a node card and the user wants files for ranks 1–10 and 11–32, both files will contain the same data.

If automatic collection is disabled, power data is no longer dumped to a flat file at the finalize call. Instead, the application being profiled by MonEQ can access the data being collected through a set of abstracted calls. As one might expect, this data is now available to the running application during the execution of the program. Thus, if desired, a program could alter itself during execution based on its power profile. While this is possible, we have not performed any experimentation of this nature in this work.

Oftentimes application developers have logically and functionally distinct portions of their software which are of primary interest for profiling. To address this, we have implemented a tagging feature. An example of how tagging is implemented is shown in Listing 3. This feature allows for sections of code to be wrapped in start/end tags which inject special markers in the output files for later processing. In this way, if an application had three "work loops" and a user wanted to have separate profiles for each, all that is necessary is a total of 6 lines of code. Better yet, because the injection happens after

```
int status, myrank, numtasks, itr;

status = MPI_Init(&argc, &argv);

MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

status = MonEQ_Initialize();              // Setup Power

/* User code */

status = MonEQ_StartPowerTag(''work_loop'');   // Start Tag

/* Work loop */

status = MonEQ_EndPowerTag(''work_loop'');     // End Tag

/* User code */

status = MonEQ_Finalize();                // Finalize Power

MPI_Finalize();
```

Listing 3. Simple MonEQ example with tags.

Table 2
Time overhead for MonEQ in seconds on Mira

	32 Nodes	512 Nodes	1024 Nodes
Application runtime	202.78	202.73	202.74
Time for initialization	0.0027	0.0032	0.0033
Time for finalize	0.1510	0.1550	0.3347
Time for collection	0.3871	0.3871	0.3871
Total time for MonEQ	0.5409	0.5455	0.7251

the program has completed, the overhead of tagging is almost negligible. More details on the performance of MonEQ will be discussed later.

Regardless if tagging is utilized, it's possible to end up with a substantial amount of data (a full system run for 5 min at the default polling interval will produce 1536 files and a total of about 830,000 data points). To address this we wrote two more pieces of software: a reducer and an analyzer. The reducer, as the name suggests, takes as input the output files generated by MonEQ and reduces them into a single CSV file. If tags are utilized, this is when they are parsed; the reducer will create a CSV file for each tag, an overall file containing all of the data points, and an "untagged" file containing all of the data points that did not fall between tags. Once the output has been reduced it can then be more easily analyzed. Because the data is still in its original form, if a user wishes to analyze the data themselves, they may do so. However, if the user has no desire to analyze the data themselves, they can utilize our analyzer. Utilizing gnuplot and R, it will produce power utilization graphs as a function of time for all of the reduced files as well as descriptive statistics. Examples of this output can be found in the following subsections.

In terms of overhead, we've made sure to design MonEQ so that it is as robust as possible without weighing down the application to be profiled. As already discussed, the overhead is mostly dependent on the number of nodes being utilized. The reasoning for this is simple, the more nodes the more data points. For this reason we've designed MonEQ to perform its most costly operations when the application isn't running (i.e, before and after execution). The only unavoidable overhead to a running program is the periodic call to record data. Here again, the method which records data does so as quickly and efficiently as possible.

Starting with overhead in terms of time, Table 2 shows the same application profiled at the most frequent interval possible at three different scales. The application is designed to run for exactly the same amount of time regardless of the number of processors making it an ideal case to study the overhead of MonEQ. It should be noted that the overhead incurred by MonEQ on this toy application would be the same as any other application profiled at the same rate. Since the time necessary to facilitate the actual collection call (i.e., to the EMON interface) is constant as it is performed by hardware,

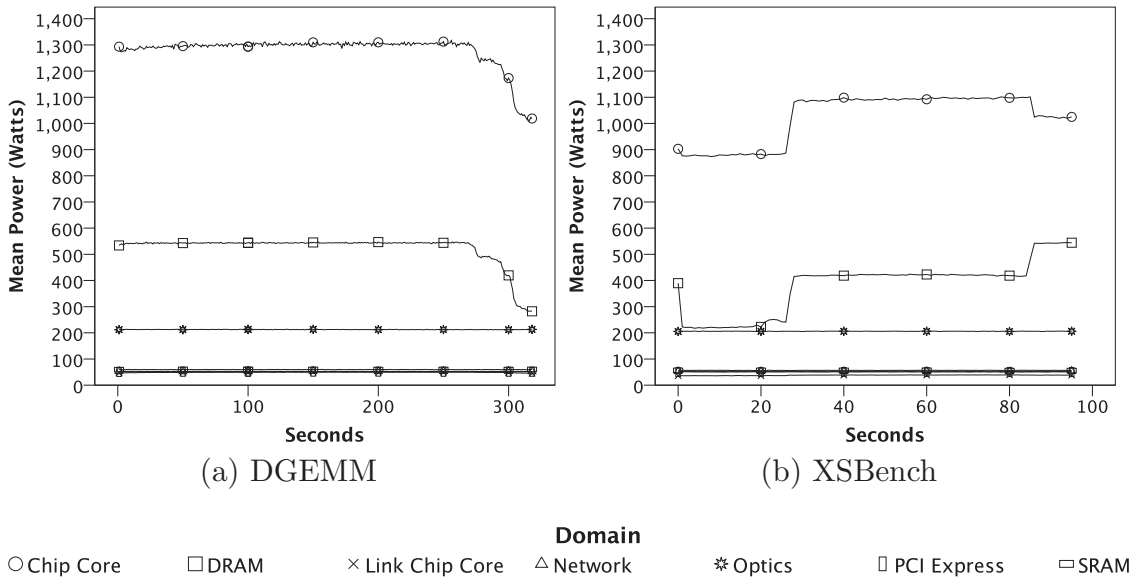

Fig. 2. Power profile of DGEMM run for about 5 min and XSBench run for about 2 min. While DGEMM shows relatively stable power consumption across all domains for the duration of the run, XSBench shows much more variation with two points of interest at 30 s when program starts generating data and 90 s when it switches from generation to random table lookups.

the only variable factor is further overhead by MonEQ in recording this data. MonEQ is completely application agnostic in its design, it simply records data retrieved, therefore the overhead for collection, which is by far the step with the most impact on the application, does not depend on the application which is being profiled.

From the data we can see that the time spent during initialization and collection is the same in all three cases with only the time spent during finalization having any real variability. This makes sense considering the initialization method only has to setup the data structures and register timers while the collection method simply records that data. Given that MonEQ collected 353 samples for each of the runs we can further say that each collection takes about 0.001 s. The finalization method really has the most to do in terms of actually writing the collected data to disk and therefore depends on the scale the application was run at. In total, at the 1K scale we see that MonEQ has a total time overhead of about 0.4%.

Memory overhead is essentially a constant with respect to scale. In the initialization stage of MonEQ an array of a custom C struct is allocated. The size of the struct is 8 bytes on BG/Q and the size of the array is dictated by a maximum number of samples, currently set to 16,384. This number is derived from a run of 8 h at the most frequent polling interval. At this default, the total memory overhead of MonEQ is about 131 kilobytes per node card or 201 megabytes for all 48 racks. Of course, this number isn't set in stone and can be modified if desired to decrease the memory overhead of MonEQ or to support a longer execution time. Thus, in a way memory overhead is essentially up to the person who is integrating MonEQ into their application.

4. Experiences of power profiling benchmarks using MonEQ

Benchmarking is a crucial part of understanding the capabilities of a supercomputing system. The Argonne Leadership Computing Facility (ALCF) staff have a variety of benchmarks designed to test both the capabilities and limitations of each system they deploy. Some of these benchmarks are designed based on real science applications that are commonly run on the system, while others are designed simply to push components to their maximum capability.

In this section we will provide detailed power analysis of two benchmarks: `DGEMM` and `XSBench`, by using MonEQ. Each is intended to test a very specific portion of the system and as such serve as excellent test platforms for profiling. For each we will show two figures: the first will be a breakdown of the various domains and how much they contributed to the total power consumption; the second a power profile from beginning to end of execution.

One of the most important areas to consider in any system with regards to power consumption is the memory. A very well known memory intensive benchmark, `DGEMM` [10], was therefore chosen for profiling. `DGEMM`, a matrix multiplication microbenchmark, is designed to overload the caches present on the CPU and randomly access data in very large matrices therefore causing massive DRAM activity. This benchmark was run on 4 racks with 16 ranks per node with an aggregate of 426.203 TFlops.

Looking at Fig. 2a, immediately it can be seen that both the chip core and DRAM domains are using the most power in `DGEMM`, which is also reflected in the domain breakdown in Fig. 3a. It can also be seen that towards the end of execution of

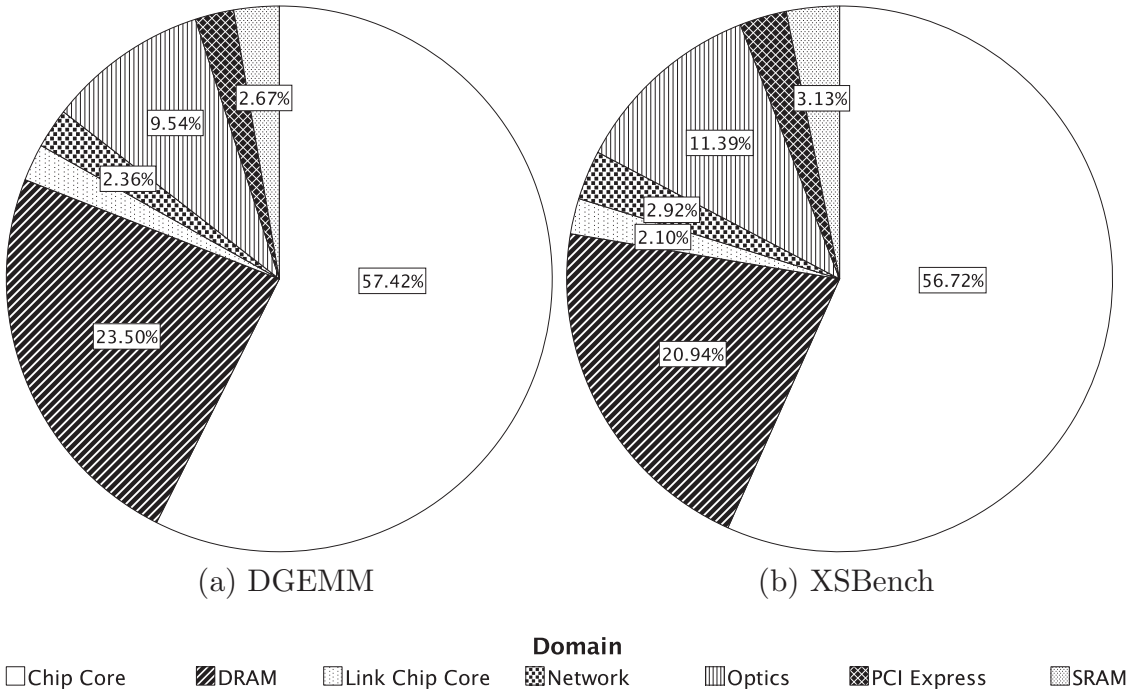

Fig. 3. Domain breakdown of DGEMM and XSBench. DGEMM again shows increased chip core and DRAM percentages.

DGEMM both the chip core and DRAM taper off back to a level closer to idle. This is because the last thing this benchmark does before terminating is free any resources it used during simulation.

XSBench [11] is a simple application that executes only the most computationally expensive steps of Monte Carlo particle transport, the calculation of macroscopic cross sections, in an effort to expose bottlenecks within multi-core, shared memory architectures.

In a particle transport simulation, every time a particle changes energy or crosses a material boundary, a new macroscopic cross section must be calculated. The time spent looking up and calculating the required cross section information often accounts for well over half of the total active runtime of the simulation. XSBench uses a unionized energy grid to facilitate cross section lookups for randomized particle energies. There are a similar number of energy grid points, material types, and nuclides as are used in the Hoogenboom-Martin benchmark. The probability of particles residing in any given material is weighted based on that material's commonality inside the Hoogenboom-Martin benchmark geometry.

The end result is that the essential computational conditions and tasks of fully featured Monte Carlo transport codes are retained in XSBench, without the additional complexity and overhead inherent in a fully featured code. This provides a much simpler and clearer platform for stressing different architectures, and ultimately for making determinations as to where hardware bottlenecks occur as cores are added.

This benchmark was run on 4 racks with 16 OpenMP threads per MPI rank. The result of the benchmark was an average of 513,749 lookups per MPI rank per second.

As with the DGEMM benchmark, an overview of XSBench is presented in Fig. 2b and the breakdown of the domains and their respective power usage is provided in Fig. 3b. As can be seen clearly, XSBench has two points of interest. The first 30 s are spent setting up the program, at which point data generation starts. This is expressed as an uptick in both the chip core and DRAM domains. The second point of interest, at about 90 s into the execution, is when XSBench switches from generating the data in the matrices to performing the random table lookups described above. These lookups result in a decrease in chip core power usage and an increase in the DRAM power usage, as expected.

One feature of MonEQ is its ability to tag specific portions of code. In Fig. 4 we show the result of placing tags around the "work loop" which is the portion of the benchmark that performs the random table lookups.

Because of the nature and design of these benchmarks, it's not difficult to look at the code and figure out why there might be a significant change in the power profile of a given domain at a certain point in time. However, the ability to wrap tags around newly profiled applications can provide an easy way to determine which portions of the application constitute dramatic shifts in power usage.

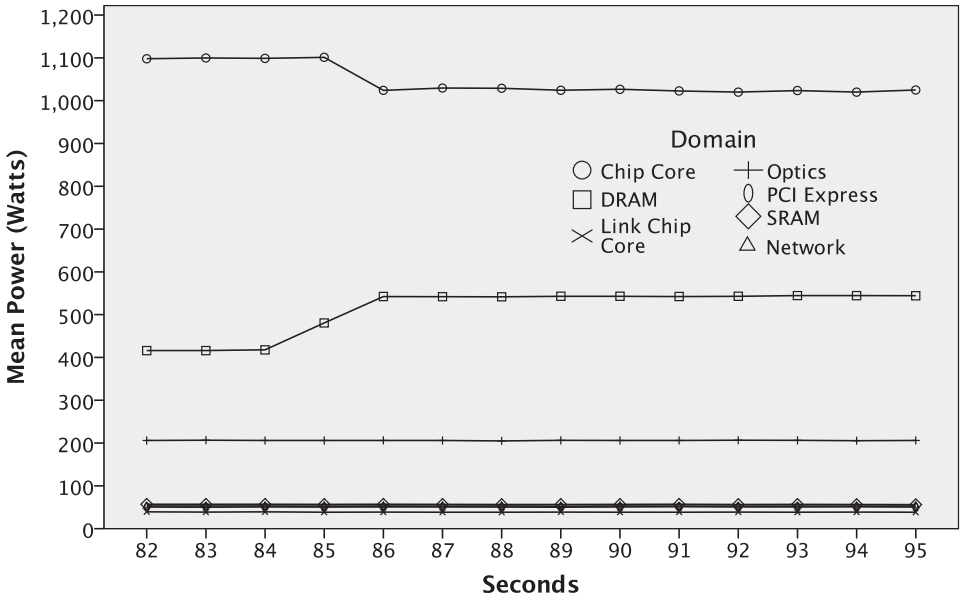

Fig. 4. Tagged "work loop" section of XSBench benchmark. Shows greater detail for portion of profiled code.

5. Effects of system variation on application power and performance

In this section, we present case studies using MonEQ for power performance analysis. In particular, we perform analysis of four parallel applications at both a variety of scales as well as with two different network topologies, 5D torus and mesh. We discuss how metrics such as execution time and energy usage change as a result of scale (ranging from 1K to 8K nodes) and network topology. Using the data obtained from these experiments, we will seek to answer the following three fundamental questions:

1. Do applications have different power requirements?
2. For a given application, does the network topology have an effect on its power consumption?
3. For a given application, does the scale at which it is run have an effect on its power consumption?

While the answer to the first question might already be obvious from the preceding sections in this work, the answers to the second and third a likely more surprising.

5.1. Experimental results

In this work, we look at two kernel benchmarks and one pseudo-application from the NAS Parallel Benchmarks (NPB) [12]. The NPB are a set of macro-benchmarks which are derived from computational fluid dynamics applications consisting of five kernels and three pseudo-applications. For the purposes of this work we looked at two out of the five kernels: Conjugate Gradient (CG) which has irregular memory access and communication, discrete 3D fast Fourier Transform (FT) which has all-to-all communication, and one of the pseduo-applications: Lower-Upper Gauss-Seidel solver (LU). Each of these kernels and pseudo-applications were run at scales ranging from 1K to 8K with both mesh and torus network topologies. Additionally, each of the kernels/pseudo-applications were run at the largest class supported, the "E" class.

Additionally, we also study a leadership application that is used routinely at the ALCF. DNS3D is a direct numerical simulation (DNS) code that solves viscous fluid dynamics equations in a periodic rectangular 3-D domain with a pseudo-spectral method of fourth-order finite differences and with the standard Runge-Kutta fourth-order time-stepping scheme [13]. DNS3D is highly dependent on network performance, since during each time step it executes three Fourier transforms for three 3-D scalar variables. This approach can effectively be transformed into all-to-all type computations.

Fig. 5 presents the execution time and energy consumption of these parallel benchmarks and applications at scales raning from 1K to 8K for both mesh and torus network allocations. 8K scale is not included for NPB:FT due to short run time generating an insignificant amount of data.

In the figure there are a few trends which present themselves. First, execution time and energy follow the same trend across all applications. The one exception to this is in Fig. 5b where at the 1K scale for the torus network topology we see that per nodecard the energy usage is about 60,000 Joules higher than the equivalent scale with a mesh network despite the execution time being exactly the same for both network topologies. Interestingly, this increase in energy usage is only seen at the 1K scale and only with the torus network topology. Unfortunately, we can not say for certain why this trend

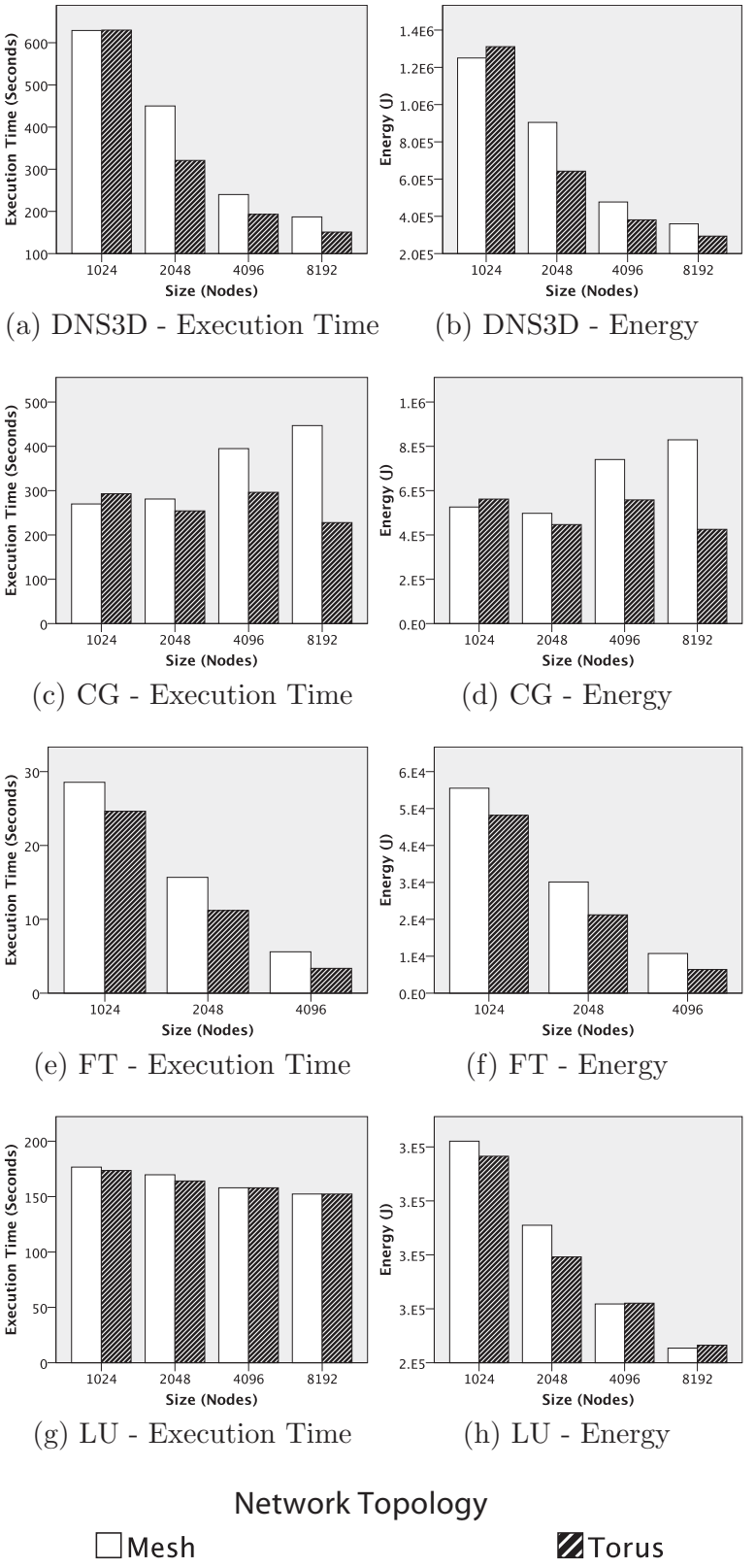

Fig. 5. Execution time and energy of DNS3D and NPB kernels/pseudo-applications.

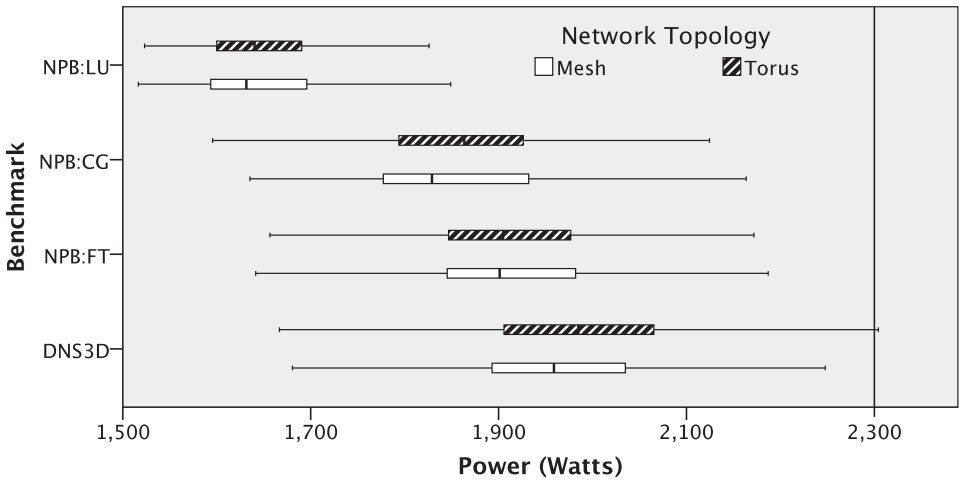

Fig. 6. Box plot comparison of average power consumption at the nodecard level for DNS3D and NPB benchmarks. Reference line at 2300 is mean power consumption of DGEMM, the most power hungry application we have profiled.

exists, but we can say that it is reproducible. To ensure that this data was not the result of a fluke run, both experiments were repeated at all scales a total of four times. The average power usage at the 1K scale for the mesh network topology was about 2.5% less than the torus network topology at the same scale. That might not seem like much, but that actually equates to a power difference of about 1.6 kW at the whole rack level between the two network topologies.

Second, the torus topology outperforms the mesh topology in most cases with the exceptions being for NPB:CG at 1K and DNS3D at 1K. While we can't say for sure why these two outliers exist, the fact that tours usually outperforms the mesh is not surprising. These applications all involve significant portions of communication with most of them using all-to-all style communication. If for no other reason than the high level of connectivity, this type of communication will perform better on torus than it will on mesh. Additionally, the torus network on BG/Q was designed from the very beginning to be exceptionally good at MPI collectives.

Third, the effect of the network topology differs from application to application. For example, looking at NPB:LU Fig. 5(g/h), there is almost no difference in execution time and energy consumption between the torus and mesh topologies. NPB:CG Fig. 5(c/d) on the other hand shows the most susceptibility to changes in execution time and energy usage due to network topology, especially for the mesh topology. The reason for the degradation as scale increases comes down to the fact the halo exchange is written in a very synchronous way with a barrier being used after a blocking send. This is the worst possible way of making a halo exchange; on a 40 off-node machine 1 link is synchronously used at a time.

5.2. Questions relating to power consumption

In the introduction of this work we brought up a number of questions which generally pertain to how power consumption changes with respect to application, network topology, and scale. In this section, we seek to answer those questions as concretely as possible providing a number of observations as a result.

5.2.1. Do applications have different power requirements?

Table 4 and Fig. 6 show the average power consumption for each of the applications we've looked at in this section. The answer to the question of whether or not applications have different power requirements is quite obviously yes. Looking at all of the data we have obtained together we see that between the most power hungry applications (`DGEMM` with an average of about 2300 Watts) and the least (NPB:LU at the 8K scale with an average of about 1600 Watts) there is a range of about 600 watts per nodecard which translates to about 19.2 kilowatts per rack. Clearly then the application can have a dramatic effect on power consumption at the rack level. For the applications looked at in this work, from most to least power hungry we have: `DGEMM`, `DNS3D`, NPB:FT, NPB:CG, `XSBench`, and NPB:LU.

At a fundamental level, this result isn't very surprising. We have shown in our previous work as well as this work that the CPU and DRAM are the most power hungry components in the BG/Q. As different applications seek to solve different problems, they will use either more or less of at least these domains resulting in different power profiles for each application. Leveraging this idea, in our previous work [14] we presented a power aware scheduling design which utilized this information which resulted in savings of up to 23% on the electricity bill.

Observation 1: Different applications have different power profiles based on which domains they utilize most.

Table 3
Combined network power of DNS3D and NPB benchmarks. Consists of the average sum of the link chip core, optics, HSS Network, and PCI Express domains per nodecard. 8K scale of NPB:FT omitted due to short run time not generating enough data to produce a significant result.

Application	Size	Network topology	Average network power per nodecard (W)
DNS3D	1024	Mesh	358.39
		Torus	362.26
	2048	Mesh	359.76
		Torus	357.31
	4096	Mesh	361.13
		Torus	358.11
	8192	Mesh	360.39
		Torus	362.31
NPB:CG	1024	Mesh	356.61
		Torus	360.94
	2048	Mesh	357.47
		Torus	357.23
	4096	Mesh	360.25
		Torus	359.68
	8192	Mesh	359.56
		Torus	358.19
NPB:FT	1024	Mesh	358.93
		Torus	360.18
	2048	Mesh	358.58
		Torus	357.56
	4096	Mesh	360.31
		Torus	360.22
NPB:LU	1024	Mesh	357.86
		Torus	358.89
	2048	Mesh	356.36
		Torus	356.69
	4096	Mesh	357.86
		Torus	359.02
	8192	Mesh	357.24
		Torus	359.27

5.2.2. For a given application, does the network topology have an effect on its power consumption?

We've already seen that from an energy consumption perspective the network topology used can have a significant impact. The question worth asking then is whether this is due to a change in power consumption, a change in execution time, or both. Table 3 shows the average power consumption for the domains which together make up the "network" on BG/Q: the link chip core, optics, HSS network, and PCI express. Surprisingly, there is practically no difference between the torus and mesh topologies at any scale for any application when it comes to power consumption.

This result is counter intuitive for a number of reasons. For one, it seems reasonable to expect that an application which is written to take full advantage of all-to-all communication such as NPB:FT would use more power in the network domains than an application which has irregular communication patterns such as CG. For another, considering NPB:FT again it also seems reasonable to expect that should the network topology be changed as to make it less efficient for all-to-all communication (i.e., from torus to mesh) that would also result in different power consumption for the network.

While it's likely this is not the case on all supercomputers, for BG/Q there is one very good reason why this is the case. The network on BG/Q is designed in a sort of "always on" mode. What this means is even for an application which has no communication with other nodes in the job the network is still at full power as if the application is hammering communication to all of the nodes. The exact reasoning behind this design is beyond the scope of this work, but it essentially comes down to a trade-off between power consumption and and complexity. If the network were to be shut off or even put into some sort of power savings mode, when it was powered back up it would require calibration which can take a long time to complete due to the sensitive nature of the optics. Simply put, it's actually better to "waste" energy and just leave the network at full power all the time than have to reconfigure it frequently.

Observation 2: At least on the BG/Q, there is no difference in power consumption in the network domains for a given topology, meaning that any reduction in energy usage is almost always going to be dictated by a reduction in time-to-solution.

5.2.3. For a given application, does the scale at which it is run have an effect on its power consumption?

To answer whether scale has an impact on power consumption, we look again to Table 4. As we can see, for each of the applications presented there is at least some reduction in power consumption at the nodecard level from the smallest to the largest scale. While it is beyond the scope of this work to say why this is the case, we can say that at least for these

Table 4
Average power consumption per nodecard and per rack for DNS3D and NPB benchmarks. 8K scale of NPB:FT omitted due to short run time not generating enough data to produce a significant result.

Application	Size	Network topology	Average power per nodecard (W)	Average power per rack (kW)
DNS3D	1024	Mesh	1987.44	63.60
		Torus	2081.08	66.59
	2048	Mesh	2009.76	64.31
		Torus	2000.96	64.03
	4096	Mesh	1987.10	63.59
		Torus	1975.10	63.20
	8192	Mesh	1923.14	61.54
		Torus	1944.28	62.22
NPB:CG	1024	Mesh	1948.52	62.35
		Torus	1918.90	61.40
	2048	Mesh	1770.24	56.65
		Torus	1757.02	56.22
	4096	Mesh	1875.19	60.01
		Torus	1885.14	60.32
	8192	Mesh	1856.14	59.40
		Torus	1866.46	59.73
NPB:FT	1024	Mesh	1943.69	62.20
		Torus	1954.67	62.55
	2048	Mesh	1920.20	61.45
		Torus	1893.79	60.60
	4096	Mesh	1917.23	61.35
		Torus	1910.39	61.13
NPB:LU	1024	Mesh	1823.96	58.37
		Torus	1823.42	58.35
	2048	Mesh	1714.83	54.87
		Torus	1701.80	54.46
	4096	Mesh	1657.69	53.05
		Torus	1659.55	53.11
	8192	Mesh	1611.29	51.56
		Torus	1618.47	51.79

applications at these scales there is a reduction in power consumption. Clearly, there is a limit to these results; it's not going to be the case that this trend continues infinitely as scale increases.

There is one interesting point that could go unnoticed which should be discussed. From an energy usage point of view, for those applications which show good speedup the reduction in power consumption is completely dominated by the reduction in execution time. Looking at the definition of energy, $E_{(J)} = P_{(kW)} \times t_{(s)}$, it's clear that a reduction by an order of magnitude in time will vastly overpower a couple hundred Watts reduction in power. Using DNS3D for example, a reduction of 123% in execution time from 1K to 8K nodes contributes much more to the reduction in energy usage than does the 7% reduction in power consumption for the same scales.

Observation 3: While we have shown there is a reduction in power consumption as scale increases, it's both lower bounded and far less important than a reduction in execution time.

6. Related work

Research in energy-aware HPC has been active in recent years, and existing studies have mainly focused on the following topics: power monitoring and profiling energy-efficient or energy-proportional hardware, dynamic voltage and frequency scaling (DVFS) techniques, shutting down hardware components at low system utilizations, power capping, and thermal management. These studies however focus on evaluating power consumption of individual hardware components and neglect to consider the system as a whole [15–25].

In terms of tools other than MonEQ which allow for the collection of power data, one of the more popular performance libraries, PAPI, also supports power collection [26,27]. PAPI is traditionally known for its ability to gather performance data, however the authors have recently begun including the ability to collect power data. PAPI supports collecting power consumption information for Intel RAPL, NVML, and the Xeon Phi. PAPI allows for monitoring at designated intervals (similar to MonEQ) for a given set of data.

From the system-level perspective, power consumption of HPC systems has increasingly become a limiting factor as running and cooling large computing systems comes with significant cost [28–30].

Hennecke et al. [31] provided an overview of the power measuring capabilities of Blue Gene/P (BGP). The measured power consumption a production workload of HPC applications and presented the integration of power and energy. However, no in-depth analysis on the accuracy was presented in that study.

Alam et al. [32] measured power and performance results of different kernels and scientific applications on BGP. They also compared these results to other large-scale supercomputers such as Cray's XT4. They concluded that while BGP has good scalability and better performance-per-watt characteristics for certain scientific applications, XT4 offers higher performance per processor.

Yoshii et al. [33] evaluated early access power monitoring capabilities on IBM BG/Q utilizing the EMON API. While they did provide an in-depth analysis of the monitoring capabilities of BG/Q, they did not analyze data from actual jobs that had run on the system as at the time only access to a single rack of BG/Q was available. This work also was limited to profiling only one MPI rank per node. We have overcome this in our monitoring mechanism and can profile any number of MPI ranks per node. Typical applications use 8-16 MPI ranks per node and MonEQ can now be used to profile applications on the BG/Q system. We have also defined an API for applications to use. Additionally, features include tagging to profile selective code fragments. Our earlier work didn't provide any of these features. Previously no tools were available for analysis and the task of dealing with the numerous output files was left up to the user. Our current work includes several utilities to analyze and plot the profiled data. Finally, the current work has demonstrated scalable performance up to 128K cores of BG/Q.

In our previous work on this subject [34,35] we further evaluated the power monitoring capabilities on BG/Q by providing an in depth comparison between the environmental data and data obtainable from the EMON API. We seek to further our experiments in this work by providing further analysis of the EMON API on more benchmarks as well as results at scale.

7. Conclusions

In this paper, we have presented a power profiling library called MonEQ for application power profiling on IBM Blue Gene/Q systems. It is extremely accurate, and capable of obtaining power data at the sub-second range with marginal overhead. MonEQ utilizes vendor-supplied low-level APIs which poll data from internal sensors located in the hardware. In its default mode, MonEQ captures data for all devices across the entire system for which there are the appropriate calls in the user's code. It also includes a tagging feature which provides users an interface to wrap particular sections of application code in start and end tags. In this way the user has the flexibility to analyze energy consumption for a particular device on a particular compute node. Furthermore, the frequency in which power data is collected (referred to as the polling interval) is user configurable with MonEQ, so users have control over how much data they generate. MonEQ is deployed on the production 48-rack Mira at Argonne Leadership Computing Facility, as well as some other Blue Gene/Q systems in US and Europe. To the best of our knowledge, MonEQ has been gaining traction among other researchers [14,36–39].

Furthermore, we have presented benchmarking results to understand the use of MonEQ. As expected, these benchmarks, which are designed to stress very specific portions of the system, had power profiles to match. In this work we looked at two benchmarks: `DGEMM`, and `XSBench`, which are designed to stress communication, memory, and computation abilities of the system.

While not the topic of this work, the data obtained from MonEQ can be used in a number of interesting ways aside from just straight analysis of power usage. For instance, in this work we have seen that different classes of benchmarks have different power profiles. Since leadership computing facilities are frequently used by teams of scientists working on different problems, they frequently have code bases which have different power profiles. Armed with this knowledge, one could use the power data obtained from MonEQ together with information about the jobs themselves (such as the "project" the job belongs to) to perform very intelligent power-aware scheduling. This is just one topic of future research we are currently investigating.

We have also provided power performance analysis of several parallel applications at scales ranging from 1K to 8K with different network topologies (i.e., torus vs. mesh). Interestingly, depending on the application, both scale and network topology can have a potentially great effect on energy usage. For example, with the LU pseudo-application scale and network topology seem to have little to no impact on execution time or energy usage. The FT micro-kernel on the other hand shows near linear speedup with respect to execution time when scale in increased and reduced energy usage when using a torus network topology. Generally, these results seem to fall in line with the fact that the 5D torus on the BG/Q is highly optimized with respect to all-to-all communication patterns. Those applications which use all-to-all communication and do so well see great reward in the form of decreased execution time and energy usage. Conversely, those applications which do no have all-to-all communication patterns don't see much benefit in using the torus topology and perform similarly as they do on a mesh network.

Acknowledgments

This research has been funded in part and used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357. The research has been funded in part by the Center for Exascale Simulation of Advanced Reactors (CESAR) Co-Design center, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357. Zhiling Lan is supported in part by US National Science Foundation grants CNS-1320125 and CCF-1422009.

The authors would also like to thank Paul Coteus and Christopher M. Marroquin from IBM for their help in clarifying results as well as providing essential information of the inner workings of the EMON system. The authors thank Vitali

Morozov, Kalyan Kumaran, and the ALCF staff for their help. They also gratefully acknowledge the help provided by the application teams whose codes are used herein.

References

[1] The Top500 List, 2014. URL http:// www.green500.org/lists/2014/11/.
[2] DOE, (2009) Architectures and technology for extreme scale computing http://www.top500.org/lists/2014/11/.
[3] The Green500 List, 2012. URL http://www.green500.org/lists/green201211.
[4] C. Bekas, A. Curioni, A new energy aware performance metric, Comput. Sci. - Res. Dev. 25 (2010) 187–195, doi:10.1007/s00450-010-0119-z.
[5] IBM, Introduction to Blue Gene/Q, 2011. URL http://public.dhe.ibm.com/common/ssi/ecm/en/dcl12345usen/DCL12345USEN.PDF.
[6] R. Haring, M. Ohmacht, T. Fox, M. Gschwind, D. Satterfield, K. Sugavanam, P. Coteus, P. Heidelberger, M. Blumrich, R. Wisniewski, A. Gara, G.-T. Chiu, P. Boyle, N. Chist, C. Kim, The IBM Blue Gene/Q compute chip, Micro, IEEE 32 (2) (2012) 48–60, doi:10.1109/MM.2011.108.
[7] G. Lakner, B. Knudson, IBM system Blue Gene solution: Blue Gene/Q system administration, IBM Redbooks, 2012. URL http://www.redbooks.ibm.com/abstracts/sg247869.html
[8] IBM, Ibm system blue gene solution: Blue Gene/q system administration, 2013. URL http://www.redbooks.ibm.com/abstracts/sg247869.html.
[9] S. Wallace, V. Vishwanath, S. Coghlan, Z. Lan, M. Papka, Comparison of vendor supplied environmental data collection mechanisms, Workshop on Monitoring and Analysis for High Performance Computing Systems Plus Applications (HPCMASPA), Chicago, USA, 2015.
[10] J.J. Dongarra, J. Du Croz, S. Hammarling, I.S. Duff, A set of level 3 basic linear algebra subprograms, ACM Trans. Math. Softw. 16 (1) (1990) 1–17, doi:10.1145/77626.79170.
[11] J. Tramm, A.R. Siegel, Memory bottlenecks and memory contention in multi-core Monte Carlo transport codes, Joint Int. Conf. Supercomput. Nuclear Appl. Monte Carlo 2013 (SNA + MC 2013) (2013).
[12] NAS parallel benchmarks. URL http://www.nas.nasa.gov/publications/npb.html.
[13] S. Kurien, M. Taylor, Direct numerical simulation of turbulence: data generation and statistical analysis, Los Alamos Sci. 29 (2005) 142–151.
[14] X. Yang, Z. Zhou, S. Wallace, Z. Lan, W. Tang, S. Coghlan, M.E. Papka, Integrating dynamic pricing of electricity into energy aware scheduling for hpc systems, in: Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis, SC '13, ACM, Denver, USA, 2013, pp. 60:1–60:11, doi:10.1145/2503210.2503264.
[15] B.-G. Chun, G. Iannaccone, G. Iannaccone, R. Katz, G. Lee, L. Niccolini, An energy case for hybrid datacenters, SIGOPS Oper. Syst. Rev. 44 (1) (2010) 76–80, doi:10.1145/1740390.1740408.
[16] V.W. Freeh, D.K. Lowenthal, F. Pan, N. Kappiah, R. Springer, B.L. Rountree, M.E. Femal, Analyzing the energy-time trade-off in high-performance computing applications, IEEE Trans. Parallel Distrib. Syst. 18 (6) (2007) 835–848, doi:10.1109/TPDS.2007.1026.
[17] R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, K. Cameron, PowerPack: energy profiling and analysis of high-performance systems and applications, Parallel Distributed Syst., IEEE Trans. 21 (5) (2010) 658–671, doi:10.1109/TPDS.2009.76.
[18] I. Goiri, K. Le, M. Haque, R. Beauchea, T. Nguyen, J. Guitart, J. Torres, R. Bianchini, GreenSlot: scheduling energy consumption in green datacenters, in: High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for, 2011, pp. 1–11.
[19] K. Kant, M. Murugan, D. Du, Willow: a control system for energy and thermal adaptive computing, in: Parallel Distributed Processing Symposium (IPDPS), 2011 IEEE International, 2011, pp. 36–47, doi:10.1109/IPDPS.2011.14.
[20] J. Laros, K. Pedretti, S. Kelly, J. Vandyke, K. Ferreira, C. Vaughan, M. Swan, Topics on measuring real power usage on high performance computing platforms, in: Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on, 2009, pp. 1–8, doi:10.1109/CLUSTR.2009.5289179.
[21] O. Mammela, M. Majanen, R. Basmadjian, H. Meer, A. Giesler, W. Homberg, Energy-aware job scheduler for high-performance computing, Comput. Sci. 27 (4) (2012) 265–275, doi:10.1007/s00450-011-0189-6.
[22] V. Patil, V. Chaudhary, Rack aware scheduling in HPC data centers: An energy conservation strategy, in: Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on, 2011, pp. 814–821, doi:10.1109/IPDPS.2011.227.
[23] Q. Tang, S. Gupta, G. Varsamopoulos, Energy-efficient thermal-aware task scheduling for homogeneous high-performance computing data centers: A cyber-physical approach, Parallel Distributed Syst, IEEE Trans. 19 (11) (2008) 1458–1472, doi:10.1109/TPDS.2008.111.
[24] T.V.T. Duy, Y. Sato, Y. Inoguchi, Performance evaluation of a green scheduling algorithm for energy savings in cloud computing, in: Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on, 2010, pp. 1–8, doi:10.1109/IPDPSW.2010.5470908.
[25] Z. Xu, Y.-C. Tu, X. Wang, Exploring power-performance tradeoffs in database systems, in: Data Engineering (ICDE), 2010 IEEE 26th International Conference on, 2010, pp. 485–496, doi:10.1109/ICDE.2010.5447840.
[26] V. Weaver, M. Johnson, K. Kasichayanula, J. Ralph, P. Luszczek, D. Terpstra, S. Moore, Measuring energy and power with papi, in: Parallel Processing Workshops (ICPPW), 2012 41st International Conference on, 2012, pp. 262–268, doi:10.1109/ICPPW.2012.39.
[27] V. Weaver, D. Terpstra, H. McCraw, M. Johnson, K. Kasichayanula, J. Ralph, J. Nelson, P. Mucci, T. Mohan, S. Moore, Papi 5: measuring power, energy, and the cloud, in: Performance Analysis of Systems and Software (ISPASS), 2013 IEEE International Symposium on, 2013, pp. 124–125, doi:10.1109/ISPASS.2013.6557155.
[28] N. Rasmussen, Calculating total cooling requirements for data centers, White Paper, 2011.
[29] W. chun Feng, K. Cameron, The Green500 list: Encouraging sustainable supercomputing, Computer 40 (12) (2007) 50–55, doi:10.1109/MC.2007.445.
[30] E. Pakbaznia, M. Ghasemazar, M. Pedram, Temperature-aware dynamic resource provisioning in a power-optimized datacenter, in: Design, Automation Test in Europe Conference Exhibition (DATE), 2010, 2010, pp. 124–129.
[31] M. Hennecke, W. Frings, W. Homberg, A. Zitz, M. Knobloch, H. Böttiger, Measuring power consumption on IBM Blue Gene/P, Comput. Sci. 27 (4) (2012) 329–336, doi:10.1007/s00450-011-0192-y.
[32] S. Alam, R. Barrett, M. Bast, M.R. Fahey, J. Kuehn, C. McCurdy, J. Rogers, P. Roth, R. Sankaran, J.S. Vetter, P. Worley, W. Yu, Early evaluation of IBM BlueGene/P, in: Proceedings of the 2008 ACM/IEEE conference on Supercomputing, SC '08, IEEE Press, Piscataway, NJ, USA, 2008, pp. 23:1–23:12. URL http://dl.acm.org/citation.cfm?id=1413370.1413394
[33] K. Yoshii, K. Iskra, R. Gupta, P. Beckman, V. Vishwanath, C. Yu, S. Coghlan, Evaluating power-monitoring capabilities on IBM Blue Gene/P and Blue Gene/Q, in: Cluster Computing (CLUSTER), 2012 IEEE International Conference on, 2012, pp. 36–44, doi:10.1109/CLUSTER.2012.62.
[34] S. Wallace, V. Vishwanath, S. Coghlan, Z. Lan, M. Papka, Measuring power consumption on IBM Blue Gene/Q, The Ninth Workshop on High-Performance, Power-Aware Computing, 2013 (HPPAC'13), Boston, USA, 2013.
[35] S. Wallace, V. Vishwanath, S. Coghlan, J. Tramm, Z. Lan, M.E. Papka, Application power profiling on IBM Blue Gene/Q, in: IEEE International Conference on Cluster Computing 2013, Indianapolis, USA, 2013.
[36] A. Malossi, Y. Ineichen, C. Bekas, A. Curioni, E. Quintana-Ortí, Performance and energy-aware characterization of the sparse matrix-vector multiplication on multithreaded architectures, in: Proceedings of 43rd ICPP, Minneapolis, MN, USA, 2014.
[37] E. Leon, I. Karlin, Characterizing the impact of program optimizations on power and energy for explicit hydrodynamics, in: Parallel Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International, 2014, pp. 773–781, doi:10.1109/IPDPSW.2014.89.
[38] L.A. Bautista-Gomez, P. Balaprakash, M. Bouguerra, S.M. Wild, F. Cappello, P.D. Hovland, Energy-performance tradeoffs in multilevel checkpoint strategies, in: 2014 IEEE International Conference on Cluster Computing, CLUSTER 2014, Madrid, Spain, September 22–26, 2014, 2014, pp. 278–279, doi:10.1109/CLUSTER.2014.6968749.
[39] L. Bautista Gomez, P. Balaprakash, M.-S. Bouguerra, S. Wild, F. Cappello, P. Hovland, Poster: Energy-performance tradeoffs in multilevel checkpoint strategies, in: Cluster Computing (CLUSTER), 2014 IEEE International Conference on, 2014, pp. 278–279, doi:10.1109/CLUSTER.2014.6968749.