Adaptive Power Profiling for Many-Core HPC Architectures

Jaimie Kelley, Christopher Stewart
The Ohio State University

Devesh Tiwari, Saurabh Gupta
Oak Ridge National Laboratory

***Abstract*— State of the art schedulers use workload profiles to help determine which resources to allocate. Traditionally, threads execute on every available core, but increasingly, too much power is consumed by using every core. Because peak power can occur at any point in time during the workload, workloads are commonly profiled to completion multiple times in an offline architecture. In practice, this process is too time consuming for online profiling and alternate approaches are used, such as profiling for k% of the workload or predicting peak power from similar workloads. We studied the effectiveness of these methods for core scaling. Core scaling is a technique which executes threads on a subset of available cores, allowing unused cores to enter low-power operating modes. Schedulers can use core scaling to reduce peak power, but must have an accurate profile across potential settings for number of active cores in order to know when to make this decision.**

We devised an accurate, fast and adaptive approach to profile peak power under core scaling. Our approach uses short profiling runs to collect instantaneous power traces for a workload under each core scaling setting. The duration of profiling varies for each power trace and depends on the desired accuracy. Compared to k% profiling of peak power, our approach reduced the profiling duration by up to 93% while keeping accuracy within 3%.

I. Introduction

Many-core architectures are now pervasive in HPC. In 2009, 51.2% of HPC workloads deployed on Jaguar at Oak Ridge National Laboratory used more than 7832 4-core nodes, and Jaguar has since expanded into Titan, with 18688 16-core compute nodes and associated GPUs [29], [30]. Soon, 72-core nodes will be available for HPC environments [26]. Unfortunately, few workloads use every core effectively. For example, PARSEC benchmarks achieve 90% of the speedup provided by 442 cores using just 35 cores [7]. Our tests with NAS workloads on Intel Xeon Phi confirm these results, using 32 cores provides 85% geometric-mean speedup relative to 61 cores.

Workloads that execute threads on every core have large and inflexible power needs. Further, their peak power needs can be significantly larger than average power needs. To be clear, peak power is the largest aggregate power draw sustained for at least a tenth of a second [10]. Large peak-power needs present a challenge for modern HPC centers that operate under a myriad of increasingly tight power caps. Circuit breakers enforce hard power caps that safeguard electrical equipment [10]. Emerging demand-response systems enforce soft power caps that shape power usage over time. For example, workloads that cap their peak power consumption during mid-afternoon hours provide significant cost savings [33]. Running Average Power Limit (RAPL) enforces soft power caps on low priority workloads [24], [14], [35].

HPC workloads can lower their peak power needs by restricting software threads to a subset of available cores. This approach is called *core scaling*. Core scaling reduces the dynamic power used to access memory, manage on-chip caches, and operate processors. With core scaling, cores that do not execute software threads have lower dynamic power needs and transition into low power operating modes. However, core scaling can increase peak power for the cores used to execute threads, because each active core executes more instructions. Resource contention can cause data movement between threads that increases power usage. Higher power usage can lead to increased temperature and potentially higher system failure rate [28], [22], [12], [3], [30].

HPC schedulers use workload profiles to allocate resources to jobs at runtime, and can change provisioning as the workload progresses [21], [5]. Workloads decide which active cores will execute threads at runtime. However, resource contention between threads is dynamic. Consequently, a wide range of runtime factors affect a workload's power usage. Offline models of power usage miss these factors [11], [4]. These models can capture the *potential* effects of core scaling but may not accurately reflect the *actual* effects.

For this paper, we studied the effects of core scaling on peak power. Our experimental design measures power usage during workload execution. As such, our measurements capture power usage caused by dynamic data movement. Specifically, we applied core scaling to HPC workloads running on modern architectures and widely used parallelization platforms. We used RAPL to trace instantaneous power usage during execution. Key findings from our study are listed below:

1. Power savings from core scaling varied across workloads and architectures.
2. Workloads that exhibited similar peak-power savings on one architecture were often unalike on another architecture.
3. Workload phases trigger power spikes at similar execution points across core scaling settings. Often, these power spikes occur early in execution. After 40% of a workload's execution, peak power up to that point predicted final peak power within 5% for 13.8% of the configurations studied.

We focus our study on peak power profiling, which partially executes a workload on a target architecture. Peak power during partial execution is used to model final peak power. Accurate peak-power models are critical for schedulers and capacity planning [24]. Our study showed that peak power changes significantly under core scaling. Further, the changes are not easily modeled analytically. Thus, we explored the challenge of peak-power profiling across core scaling settings.

State of the art profiling approaches partially execute workloads for a fixed sampling duration (e.g., 5 min) [5]. We developed adaptive peak-power profiling. Our approach accepts

Factor: Architecture

Values	Features
i7	4 cores, 256 KB L2, 8 MB L3, 3.4 GHz, 95 W TDP, 32 nm
Sandy Bridge	8 cores, 256 KB L2, 20 MB L3, 2.6 GHz, 115 W TDP, 32 nm
Xeon Phi	61 cores, 512 KB L2, N/A, 1.05 GHz, 225 W TDP, 22 nm

Factor: Parallelization Platform

Values	Features
MPI	Message passing interface, limited support for shared memory
OMP	Open Multi-Processing platform provides extensive shared memory support

Factor: Workload

Values	Features
CoMD	N-body molecular dynamics. Frequent inter-thread communication
Snap	PARTISN workload used at LANL. Iterative inter-thread communication
MiniFE	Finite-element workload composed from 4 parallelizable kernels
NAS benchmarks	Widely used benchmark suite (CG, EP, FT, IS, LU, MG)

TABLE I
SECONDARY FACTORS IN OUR EXPERIMENTAL DESIGN. THERMAL DESIGN POWER (TDP) IS MAXIMUM POWER CONSUMPTION.

two inputs: sampling duration and desired accuracy. First, it partially executes the workload for the sampling duration and collects a power trace. To estimate peak power under other settings, it executes the workload until either 1) the workload causes a power spike or 2) the sampling duration is reached. Our approach reduced profiling time by more than 40% for most applications while allowing for scaled estimation accuracy.

The remainder of this paper is as follows. Section II describes the methodology for our study. Section III characterizes peak power as workloads and architectures changes. Section V characterizes peak power relative to instantaneous power early in a workload. Section VI presents our adaptive profiling approach driven by our results. Section VII presents related work. Section VIII concludes.

II. EXPERIMENTAL METHODOLOGY

Core scaling can increase or decrease peak power. As core scaling idles more cores, more cores have reduced their power consumption compared to the active cores. On the other hand, reducing the number of active cores causes the active cores to sustain larger work load which can increased power needs. Our experimental design characterizes the relative importance of these diametric forces. In this section, we first provide details on our power measurements. Then, we discuss independent secondary factors that affect core scaling. These are listed in Table I.

A. Power measurement

In our experiments, we used RAPL to measure power usage during workload execution. RAPL stores power measurements per CPU socket in Linux Machine State Registers (MSR). The open-source libRAPL library provides an API to access MSR [17]. Specifically, we measured on-core, and uncore (memory), and total energy per CPU socket, since initialization. We converted total energy into power by associating each libRAPL call with its corresponding wall-clock time. We issued libRAPL calls every 100 milliseconds. For Xeon Phi, we read power from micsmc [18] instead of RAPL.

We verified our approaches with Like I Knew What I Was Doing (likwid) [31], an open-source tool for reading hardware counters. Likwid measured total energy for a whole workload execution. However, likwid verified the energy measurements from our approach within 1% on average.

B. Architectures

We experimented with 3 architectures: an i7-2600K, Sandy Bridge (Xeon e5-2670) and Xeon Phi (5110P). The specifications for each architecture are shown in Table I. Each architecture used low-power operating modes when cores idled. For the i7 processor, we set up the On-Demand CPU Governor, a kernel driver that adjusts operating frequency. The maximum frequency was 3.4 GHz. The minimum frequency was 1.6 GHz. For the Sandy Bridge processor, we enabled processor controlled P-states. The maximum and minimum operating frequencies were 1.2 GHz and 2.6 GHz, respectively. The Xeon Phi used the micsmc controller with P-states and C3 package [18]. The C3 package can reduce power usage 45 W when the whole processor is under used. P-states reduce power based on each core's usage.

C. Platforms

Message Passing Interface (MPI) and OpenMP (OMP) platforms provided mechanisms for core scaling. Specifically, we specified the number of active cores when we launched workloads. Both platforms exhibited similar peak power when we ran the micro-benchmarks. The average difference was less than 2%. However, the platforms achieve parallelism using qualitatively different approaches. MPI uses message passing with limited transparent support for shared memory. OMP uses shared memory. We hypothesized that these platforms could cause different peak power under core scaling.

We used likwid to read hardware counters related to data movement. Specifically, in this paper, data movement refers to data path transfers between: L1 and registers, L1 and L2, L2 between cores, L2 and L3, LLC and memory. Data path transfers include loading data into the cache, writing data back to memory, and coherence related transfers. We use data movement throughout the paper to explain the root causes for the effects of core scaling.

D. Workloads

For our research, we used the set of NAS Parallel Benchmarks available from the NASA website [13]. To ensure

fairness, we only used benchmarks which could run on a number of cores equal to a power of 2. These six benchmarks are Conjugate Gradient (CG), Embarrassingly Parallel (EP), discrete 3D fast Fourier-Transform (FT), Integer Sort (IS), Lower-Upper Gauss-Seidel solver (LU), and Multi-Grid on a sequence of meshes (MG). These benchmarks, commonly used in the HPC community, run on a number of cores that is a power of 2. We used application size B unless otherwise specified. Application size B is the appropriate size to run on a single node. Integer Sort was written in C, while the other 5 NAS benchmarks were written in Fortran.

The Conjugate Gradient algorithm (CG) is known for its irregular memory access patterns [13]. A member of the unstructured grid type of computations, it approximates the smallest eigenvalue of a positive definite symmetric matrix. The matrix in this data set is large and sparsely populated. The Embarrassingly Parallel algorithm (EP) tests the limits of floating point performance, and has no significant communication between cores. The Fourier-Transform algorithm (FT) solves a 3D partial differential equation, and is representative of spectral computations. Its primary feature is all-to-all communication. The Integer Sort algorithm (IS) is known for random memory access. This sorting algorithm tests the speed of integer computation. The Lower-Upper Gauss-Seidel solver (LU) uses a system of nonlinear partial differential equations to simulate computational fluid dynamics. LU contains a limited degree of parallelism compared to these other benchmarks. The Multi-Grid algorithm (MG) is memory intensive and features long and short distance communication [2].

In addition to the NAS Parallel Benchmarks, we also use three kernels used to simulate real high performance computing workloads running on Titan, named CoMD, miniFE, and snap. CoMD is a classical molecular dynamics application. MiniFE is a finite-element mini-application which assembles a sparse linear system using a conduction equation, then solves this sparse linear system using a conjugate-gradient algorithm. The final result is then compared to the analytic solution [1]. Snap is a proxy application for modeling modern discrete ordinates neural particle transport application performance. Snap mimics the communication patterns, memory requirements, and computational workload of PARTISN, which solves the linear Boltzmann transport equation.

III. Observations on Power Consumption

In this section, we present observations from our study of the effect of increasing the number of cores on the power consumption on different architectures and workloads.

Observation 1: Different architectures observe significantly different relative increase in power as the number of active cores increases. Reasons for this include the power consumption of other on-chip resources and built-in support for dynamically managing the power consumption of those resources based on the activity.

We investigated how only increasing the core counts affects the power consumption by running a synthetic benchmark. This synthetic benchmark stresses the processor registers only

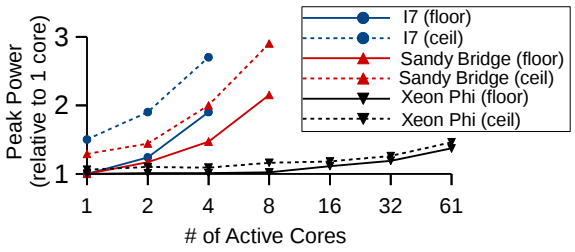

Fig. 1. Peak power of workloads that repeatedly access L1, L2 and L3 caches and memory. Y-axis is relative to power consumed on 1 core by the workload with minimum (floor) peak power.

and does not access any level of cache. The floor measurements in Figure 1 show the relative increase in power found with this simple synthetic benchmark. The relative increase in power is highly dependent even for such a simple synthetic benchmark. For example, Intel i7 and Sandy Bridge platforms observe a 1.9x and 2.15x increase respectively in the power at their maximum number of core compared to a single core count case, while such an increase in Intel Xeon Phi is only 1.37x. This indicates that as the number of cores scale for a processor core intensive workload, other resources (e.g., caches, interconnect) may also observe increased power usage in certain architectures. One of the reasons for this is that bandwidth to DRAM and L3 cache are dependent on frequency in the Intel Sandy Bridge architecture [25].

L1, L2, and L3 caches also affect power usage. We created micro-benchmarks that targeted each layer. Specifically, each micro-benchmark repeatedly accesses data stored in the targeted cache layer but not in the layers above it. Our micro-benchmarks capture the power caused by fetching data from the targeted layer. For example, a fetch from L3 causes data movement in the L1 and L2 layers. A fourth micro-benchmark repeatedly accessed data from memory.

The ceiling measurements in Figure 1 shows the effect of core scaling on the power consumption of our cache and memory micro-benchmarks. One may expect that the memory micro-benchmark will have provided a ceiling for each of our experiments, but this is not always the case. The micro-benchmark that targeted memory provided a ceiling only at lower core counts. On the Sandy Bridge when all cores were active, the microbenchmark that accessed L3 cache consumed 16% more power than the one that targeted memory. This reflects bandwidth saturation in the microbenchmark that accessed memory; Sandy Bridge automatically lowered operating frequencies when cores idled waiting for memory. The i7 architecture also showed evidence of bandwidth saturation as the number of active cores increased. On the i7 architecture, the memory-targeting microbenchmark only consumed the most energy at a single core. The L3-targeting microbenchmark consumed 1% more power at 2 cores than the memory-targeting microbenchmark. The L2-targeting microbenchmark consumed 14% more power at 4 cores than the memory-targeting microbenchmark and 7.6% more power than the L3-targeting microbenchmark. Even on the Xeon Phi

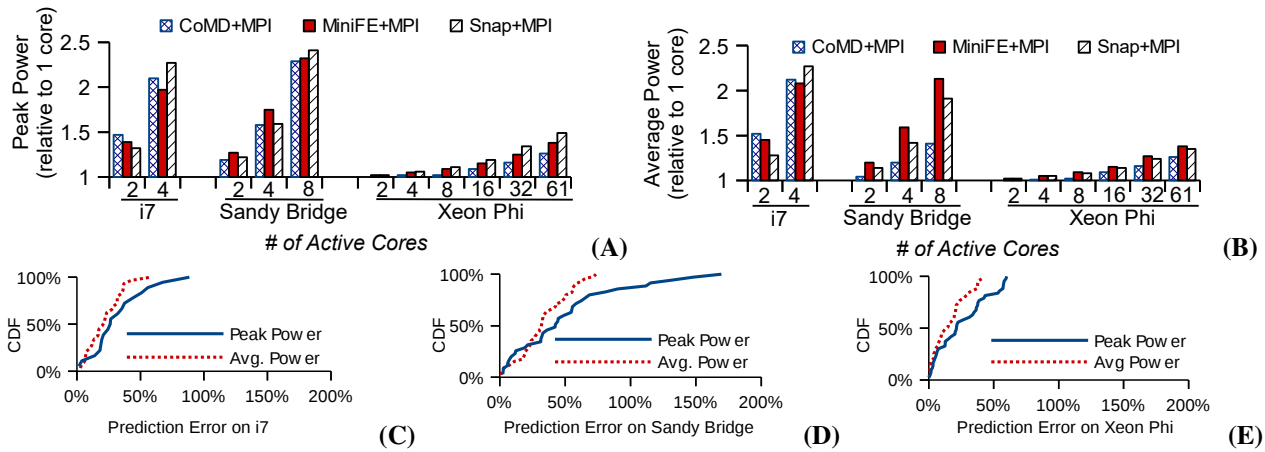

Fig. 2. Power and prediction error ranges vary across architectures.

architecture, the 8 core experiment showed that the LLC-targeting microbenchmark used 2.7% more power than the microbenchmark that targeted memory. At larger core counts, the LLC and memory microbenchmarks peaked at the same power consumption on the Xeon Phi.

Core scaling has the greatest effect on peak power on i7 and Sandy Bridge, since the ceiling power at maximum active cores was as high as 2.7X and 2.9X the floor power on one core. In contrast, on the Xeon Phi, using 61 cores cost at most 1.45X the floor power of a single core. The Xeon Phi uses a ring-based inter-connect that is fully powered if just 1 core is active. This reduced the power savings provided by core scaling.

Core scaling caused larger increases in peak power as the number of active cores increased. This was especially true for the ceiling micro-benchmarks that caused large amounts of data movement on each access. Note, the micro-benchmarks do not reflect upper or lower bounds on peak power. Workloads can stress multiple components at the same time.

Observation 2: There exists significant variation in relative increase in power consumption across workloads on different architectures. In fact, the same workload may exhibit significant variation in peak power consumption between platforms with increasing core counts.

We found this observation while studying how power consumption varies for different numbers of active cores across different real MPI workloads. Figure 2(A-E) present the peak and average power consumed by different MPI workloads at different core counts on different architectures. These results show a couple of interesting trends. We observe that there exists a significant amount of variation in power consumption across workloads on all architectures. As the core count increases, the relative difference in the power consumption across workloads varies significantly as well. For example, on eight cores of the Intel Sandybridge platform, there is more than 20% variation in peak power consumption across all tested workloads, while such a variation is very limited at single core count (less than 5%). This variation is even more pronounced for average power consumption. These results indicate that workloads may benefit significantly by reducing the numbers of cores used, both in terms of peak power and average power consumption. However, such a variation is highly architecture dependent. For example, we observed less than 10% variation in peak power consumption across workloads on Intel Xeon Phi platform even at very high core count. This occurs in part because there is only one clock source in the coprocessor, so Intel Xeon Phi runs all active cores at the same frequency. Regardless of how much bandwidth the application requires, the Intel Xeon Phi always enables full bandwidth if even a single core is active [15]. Therefore, the only separation that occurs in the peak power between different benchmarks is in the processor frequency requirements.

Finally, we also observed that relative increase in power consumption for the same workload changes significantly across architectures. As an example, MiniFE has the highest relative increase of peak power on the Intel Sandy Bridge architecture of the workloads shown, but has a much lower relative increase in peak power on the Intel Xeon Phi compared to Snap.

IV. Predicting Peak Power using Reference Workloads

In Section III, we observe that there may be some variance in peak power across workloads and architectures (Figure 2). Given the limited variance in some cases, it is intuitive to investigate if one workload can be used as a reference for predicting the peak power of other workloads. Clearly, this approach has its practical limitations. For example, it may not be possible to examine all the possible reference workloads and identify the most suitable reference workload in a production environment. Identifying suitable reference workloads will require knowing the intrinsic characteristics and properties of workloads that affect the power consumption. Unfortunately, knowing such intrinsic properties and quantifying them may require high overhead profiling. In some cases, this may not be possible at all because production workloads may be mission critical (or proprietary).

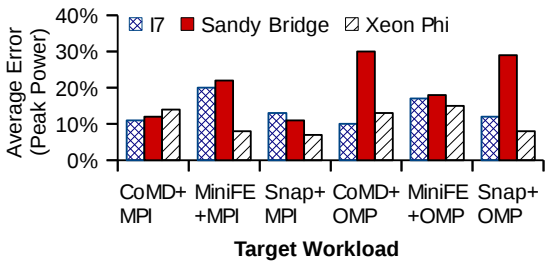

Fig. 3. Average error in predicting peak power varies across architectures.

Nevertheless, we explore this approach to understand the limits of peak power prediction when using other workloads as a reference.

Observation 3: We observed that using one reference workload to predict peak power of other workloads can result in highly variable inaccuracy, especially across architectures. The best reference workload may even change across architectures.

To discover this, we first investigated the effect of using a single workload as a reference for predicting peak power of multiple applications. While it is possible that each application may have its own best suited reference workload, we first tested the limits of this approach using a randomly selected workload as the reference. Figure 3 shows the error in predicting peak power using a randomly selected NAS benchmark as the reference to predict peak power of three applications on different architectures. We observe that the peak power error rates can be as high as 30% in some cases and can vary significantly across architectures. This indicates that the same reference workload can cause different amounts of error in peak power prediction on different architectures for the same target application.

However, as pointed out earlier, the best reference workload may be different for each application. Therefore, to further test the limits the peak power prediction from using reference workload, we found a pair of applications that have almost zero prediction error on one architecture (i.e., almost same peak power) and measured the difference in the peak power on a different platform. Figure 4 shows that while the NAS IS benchmark and CoMD application have very similar peak power on Intel i7, their peak power on the Intel Xeon Phi differ especially at higher core counts. Second, Figure 4 also shows that CoMD MPI and OpenMP versions have very similar peak power on Intel Xeon Phi even at very high core counts, but significantly different peak power on Intel Sandy Bridge. These results illustrate that using the same reference workload may result in poor peak power prediction.

V. Analyzing the Power Consumption Profile of Scientific Applications

In this section, we analyze the power consumption profiles of scientific applications on different architectures to derive insights about dynamic behavior of power consumption (Figures 6 - 8). We make several interesting observations about the dynamic power-profiles of these applications.

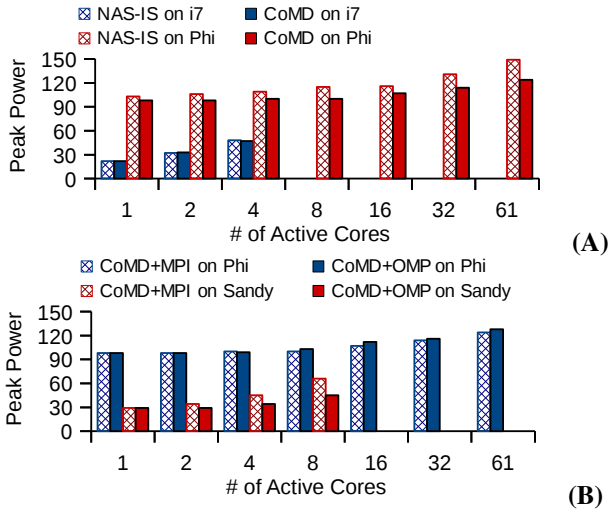

Fig. 4. Peak power can be predicted using other workloads, with varying results. (A) CDF of error seen from all pairwise combinations of benchmarks. (B) Pairs with error under 15% on Intel i7 show that correlations do not continue across architectures.

First, we investigate what fraction of the total execution time is spent at different power consumption levels. Figure 5 shows the cumulative distribution function of power consumption over the whole execution of applications on different architectures. We observe that different applications spend different amounts of time in or near their peak power consumption level. This characteristic is also highly dependent on the architecture even for the same application. For example, on the Intel Sandy Bridge architecture, MPI versions of CoMD, snap, and miniFE spend more than 50% of their execution time at a power consumption level higher than the 80% of the peak power of the given application. In contrast, these same applications spend only 20% of their execution time at a power consumption level higher than 80% of peak on the Intel Xeon Phi. Autonomic programs and schedulers often need to know how much of the time will be spent at k% or higher of the peak power load for efficiently provisioning the power distribution and estimating the cost of power and cooling. This analysis suggests that estimating how much time will be spent in or near peak power requirements is not only application-dependent, but also highly architecture dependent.

Observation 4: We observed that the power consumption behavior changes significantly across application and can change significantly across architectures even for the same application. We also observed that there may be noticeable change in the power consumption characteristics across different implementations (MPI versus shared memory) of the same application. Both the underlying architecture and the application characteristics significantly affect how much time is spent in or near the peak power.

As expected, scientific applications may exhibit distinct power consumption phases. However, we also observe that the power phases for a given application may change significantly

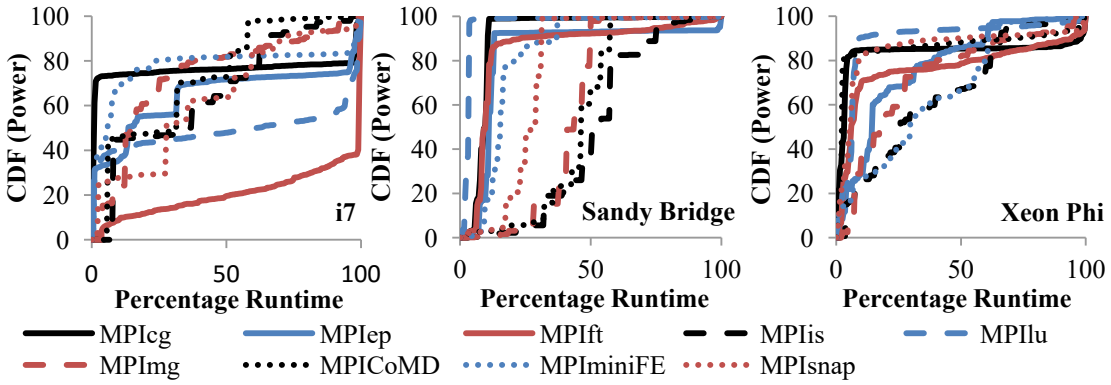

Fig. 5. CDF of power consumption for different applications and architectures.

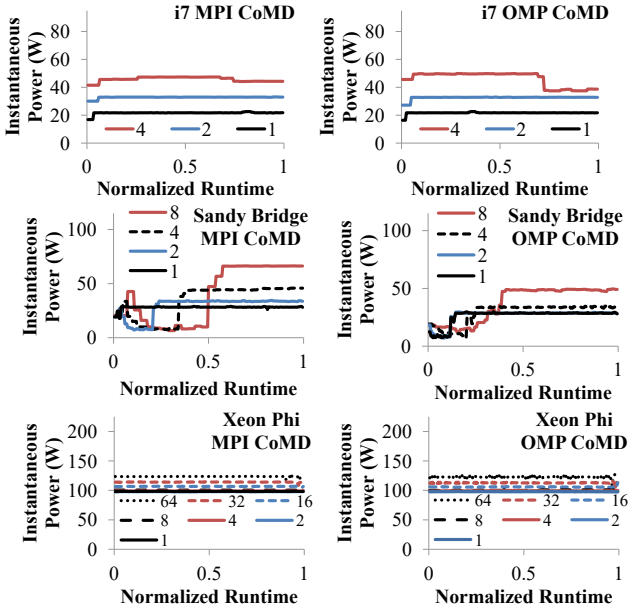

Fig. 6. Power profile of CoMD application for MPI and OpenMP implementations on different architectures.

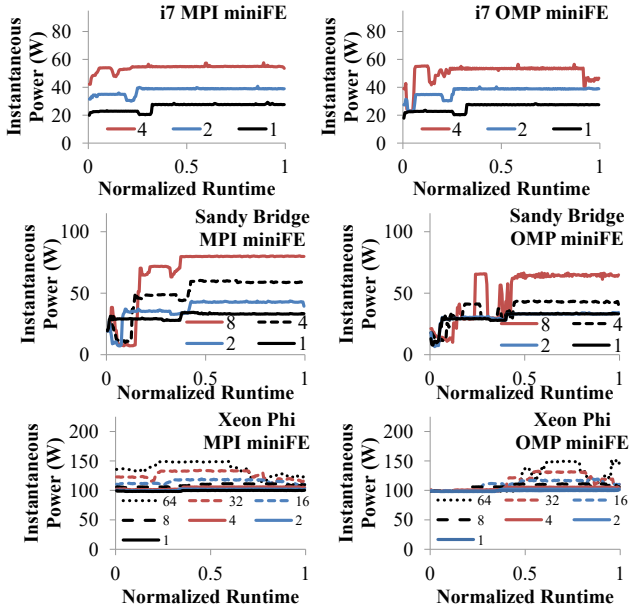

Fig. 7. Power profile of miniFE application for MPI and OpenMP implementations on different architectures.

as we change the underlying architecture. For example, MPI implementation of CoMD has distinctly different power profiles on Intel i7 and Sandy Bridge platforms (Figure 6). Similar observations are true for MPI implementations of miniFE and snap applications. We note that this observation is not limited to the MPI implementation or multi-core platforms only. For example, OpenMP implementation of miniFE exhibits significant differences in the power profile on Intel i7, Sandybridge and Xeon Phi architectures (Figure 7).

Second, we observed that while MPI and OpenMP implementations of CoMD and miniFE applications exhibit similar power profiles on a given architecture, the third application, snap, shows significantly different power profiles between MPI and OpenMP implementations (Figure 8). This observation is pronounced on Intel i7 and Sandy Bridge platforms, albeit not so much on the Intel Xeon Phi platform. We observe that even the peak power differs significantly between MPI and OpenMP implementations. This indicates peak-power requirements are not only dependent on the architecture and underlying algorithm, but also on the platform (programming model) on which they are implemented (message passing versus OpenMP). For the applications we tested, we observed that MPI implementations usually resulted in higher peak power consumption (Figures 6 - 8).

Observation 5: Interestingly, for a given application the power profiles are similar across different numbers of active cores on a fixed architecture given normalized execution time.

This is an important observation that we exploit to estimate peak power across core counts for a given application and architecture pair in (Section VI). When we fix architecture, application, and platform, the power profiles with changing number of active cores display similar phases occurring at similar points in their normalized execution. A clear example of this is the OpenMP implementation of snap in Figure 8(D -

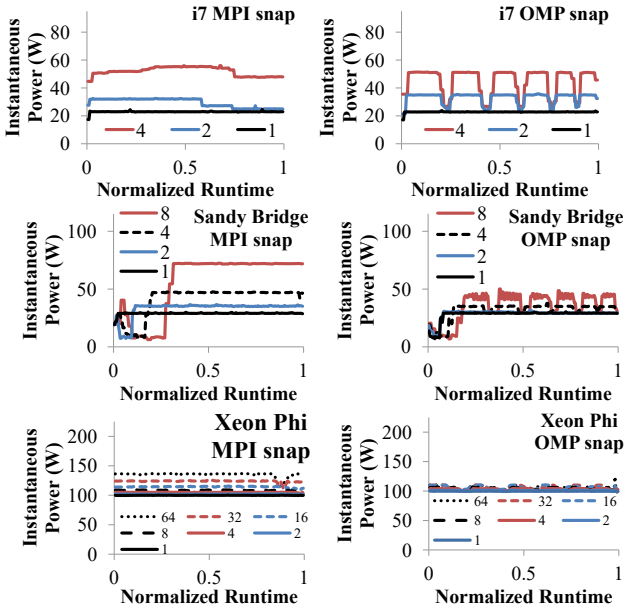

Fig. 8. Power profile of snap application for MPI and OpenMP implementations on different architectures.

F). Regardless of architecture, clear phases can be seen, more distinctly at higher numbers of active cores yet still present at lower core counts. These phases are most distinct at higher core counts on the Intel Sandy Bridge and Intel i7 architectures because the variance in power with added cores is greater with these Turbo Boost capable architectures. However, these phases are still visible on the Intel Xeon Phi.

Power spikes from execution phases often occur early in execution, as can be explicitly seen in the power traces from Intel Sandy Bridge. We tested our workloads to see how far in their execution peak power occurs. In order to do this, we computed the maximum power so far seen in the workload at each timestep, and computed relative difference using this number and the peak power seen over the entire trace. This gave us a trace of peak power estimation error over the entire run of the workload.

We found that in many cases, profiling a workload on any core count for 40% of its execution resulted in peak power error below 5%. We show in Figure 9 the peak power error seen as the time spent profiling a workload increases for the MPI NAS benchmarks on Intel Sandy Bridge. Certain workloads can be profiled for a shorter time to achieve the same peak power error, and other workloads take longer on certain core counts. Overall, we found that peak power generally occurred later at higher core counts.

VI. Adaptive Power Profiling

In this section, we present an algorithm to profile peak power across many core scaling settings. Consider the most widely used approach to profile a workload's power [5], [6]:

1. Choose how much of the workload to execute (i.e., k% of the running time).

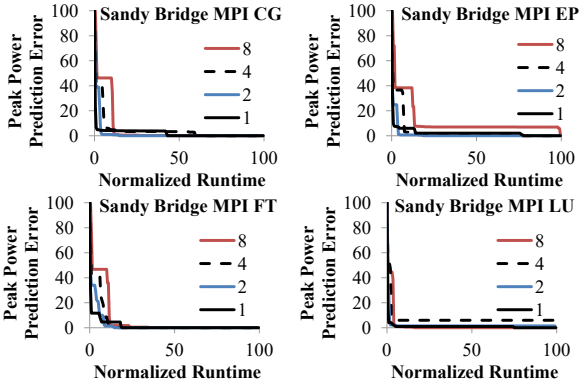

Fig. 9. Peak power estimation error curves

2. Run the workload on the target architecture for k% of the running time.

3. During each run, collect power usage.

We call this approach *k% sampling*. To profile peak power under 5 core scaling settings, k% sampling must run 5k% of the workload. For $k = 5$, the delay to profile is 25% of total running time [5]. The previous sections showed that power phases occur at approximately the same point in a workload's normalized execution regardless of the number of cores used. Our approach uses power phases to reduce profiling time.

Figure 9 plots the absolute difference between peak power observed during execution and peak across the whole workload. The x-axis shows peak power observed at several execution points, ranging from 0.5% to 100% of the execution. Each line will eventually converge to zero when the peak power is observed. The curves converge quickly if peak power occurs early in the execution, e.g., MPI FT. They also converge quickly if other high but not peak phases occur early in execution, e.g., MPI EP. Power phases ensure that the curves look similar across core counts.

Our approach profiles for k% at a single core count (by default the maximum core count). We then construct a single curve from Figure 9. We find the execution point on the x-axis where the error drops below a user provided threshold. This point is ($\hat{k}$%) and $\hat{k}$<k. *We then adapt our profiling to run workload for only $\hat{k}$% on the remaining core scaling settings.* The inputs to our approach are:

1. Sampling duration (k%)

2. Accuracy (i.e., maximum expected error)

3. Active cores for initial profiling run (by default, we assume the initial run will use all cores).

A. Our Profiling Method

Our method collects a power trace from a k% run of the workload at a single core count. This trace is represented by $PP(i)$ where $PP(i)$ is the power observed at timestep i. At each point in this trace, the peak power so far seen $PPmax(i)$ is calculated. We are guaranteed that the most accurate peak power over k% of the normalized execution of the workload

will be found at $PPmax(k)$, where $PP(k)$ is the last reading in the trace. From this, we calculate a trace showing the expected error in estimating peak power $PPEC(i)$. For the requested accuracy of $a\%$, our model uses $PPEC(i)$ to find the first point in the normalized runtime at this core count where an error less than $1-a$ occurred.

$$PPmax(i) = max(PP(1), ..., PP(i)) \quad (1)$$

$$PPEC(i) = \frac{PPmax(k) - PPmax(i)}{PPmax(k)} \quad (2)$$

Algorithm 1: This algorithm is used to find the duration to sample at other core counts.

```
Data: PPEC[], k, a
Result: find sampling duration k̂% to run at other cores
for i ← 0 to k do
    if PPEC[i]<a then
        return i;
    end
end
```

We then collect a power trace from all other core counts for $\hat{k}\%$ normalized execution (Algorithm 1). Since we use the maximum number of cores by default, we conservatively assume that the profiled workloads are perfectly parallelizable. Running time doubles when core scaling halves the number of active cores. On every core scaling setting except the initial, our profiling runs for $\hat{k}\% \times \frac{\text{initialCoreCount}}{\text{currentCoreCount}}$. This time is likely longer than it would be if we knew the execution time on other core counts.

B. Evaluation

In our evaluation, we compare our method to k% profiling and prediction from similar workloads. These experiments used a fixed maximum core count, and assumed that a speedup profile was available so the correlation between k% and an actual running time was not a concern.

First, we compared the profiling duration at k% profiling to the profiling duration used with our max-core first method. For this experiment, we determined the percentage of workload to run by first profiling for k% on the maximum number of active cores, then finding the percent time at which the peak power was found. This percent time was then used as the profiling duration for the other core counts profiled. Our results in Figure 10 show the 25th and 75th percentile of total time used to profile all cores of a given workload. We found that we could reduce the amount of time profiling a workload across all core counts by up to 93% on the Intel Xeon Phi, 60% on the Intel Sandy Bridge, and 73% on the Intel i7. The average time saved by using our max-core first method was 25% on the Xeon Phi, 12% on the Sandy Bridge, and 11% on the i7 architecture.

We also examined the effects of this selective reduction in profiling time on the profile's average inaccuracy (accuracy - 1). We found that on average, our max-core first method produced a profile within 0.3% of the profile produced with k% profiling for the Intel Xeon Phi. This same analysis found average difference of 1.5% on the Intel i7 and 3% on the Intel Sandy Bridge architecture.

We tested the behaviour of our max-core first model while changing the percentage of peak power we located in the profile of the highest core count. When we relaxed this requirement from 0% error in peak power estimation, we found that the resulting profiling time for other core counts still produced inaccuracy less than 3% for 75% of workloads, as shown in Figure 13. However, when this approximation percentage rose above 7% on the Intel i7, we saw the 75th percentile of workloads reach 5% or higher inaccuracy. This approximation dial allows us to trade accuracy for shorter profiling time. Figure 12 show that for our experiment with 100% profiling time requested, our max-core first model acquired these approximate profiles for most workloads using less than 60% of the profiling time requested on the Intel i7 and Sandy Bridge. The Intel Xeon Phi had a wider variation in estimated profiling time. The maximum normalized runtime used to get peak power stayed near 100% across architectures, but median normalized runtime dropped even with a 2% reduction in expected accuracy.

C. Corner Cases

We primarily tested our model using the maximum core count as the profiled core count, but it is possible to use a different core count for this. We experimented with profiling the minimum core count, the median core count, and the maximum core count to completion. We then profiled additional core counts using the percent time at which peak power was found on the fully profiled core count. The resulting peak power inaccuracy was averaged across all core counts for each workload. We show in Figure 14 the median inaccuracy across all workloads for each architecture. We found that the core count which offers the lowest median inaccuracy was the maximum core count, 4 for the Intel i7 and 8 for the Intel Sandy Bridge. The Intel Xeon Phi had a lower average inaccuracy on the minimum core count by 0.06%, but more than twice the number of workloads achieved 0% inaccuracy on the maximum core count compared to the minimum core count. Using the maximum core count, the greatest inaccuracy was 0.56%. Despite the intuition that using the maximum core count to determine how long to profile on other core counts, no core count was clearly preferable to the others in terms of running time.

VII. Related Work

Researchers have studied the impact of core scaling on performance, and power consumption as the number of available cores on the multicore processors increases [27], [20], [11], [32]. For example, [27] proposed a feedback-driven approach to maximize performance by varying the number of threads at runtime based on data synchronization and off-chip bandwidth. Similarly, [20] propose approaches that can choose the optimal

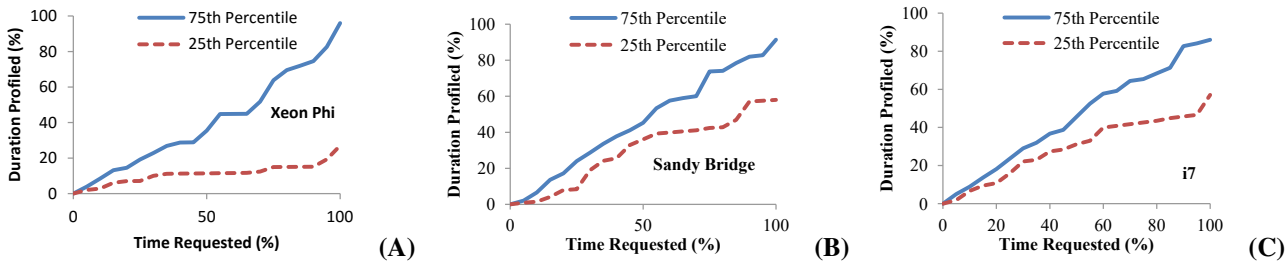

Fig. 10. Duration spent profiling with our method as the requested time to profile increased. (A) Intel Xeon Phi (B) Intel Sandy Bridge (C) Intel i7.

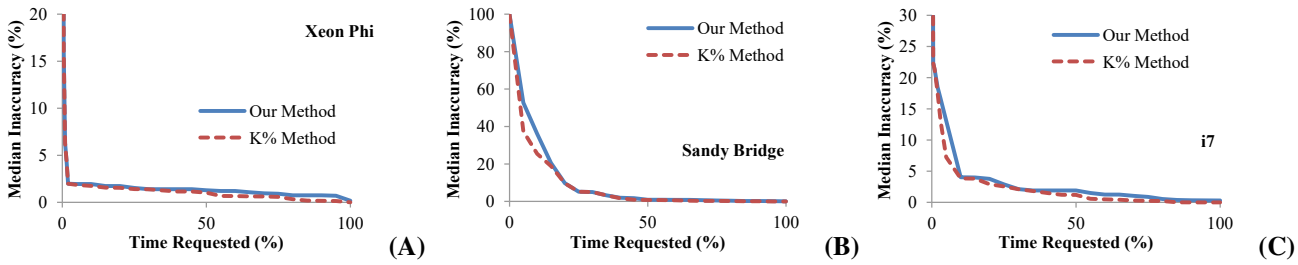

Fig. 11. Median inaccuracy across all benchmarks for our method and k% profiling as the requested time to profile increased. (A) Intel Xeon Phi (B) Intel Sandy Bridge (C) Intel i7.

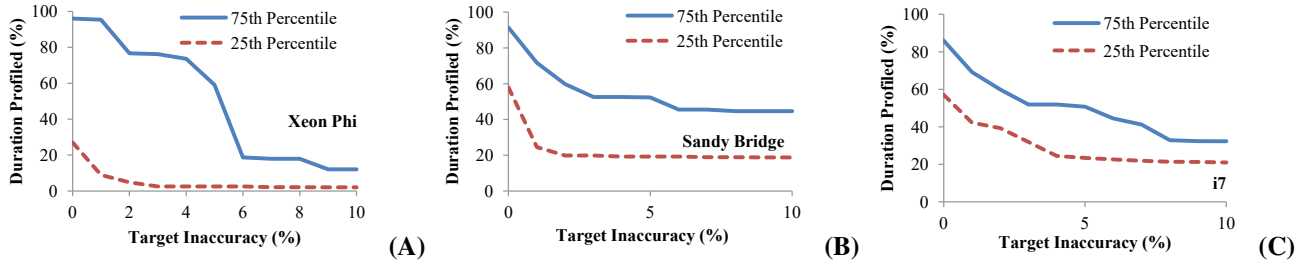

Fig. 12. 75th and 25th percentiles of inaccuracy across all benchmarks for our max-core first method as the approximation of peak power requested from the maximum core profile increases. (A) Intel Xeon Phi (B) Intel Sandy Bridge (C) Intel i7.

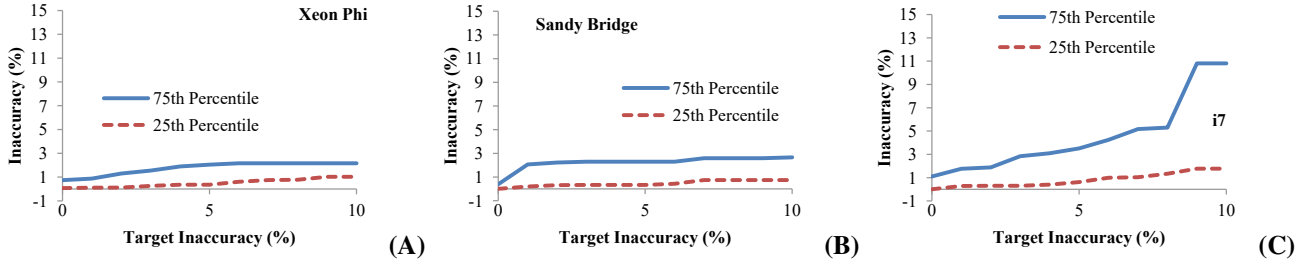

Fig. 13. 75th and 25th percentiles of profiling duration across all benchmarks for our max-core first method as the approximation of peak power requested from the maximum core profile increases. (A) Intel Xeon Phi (B) Intel Sandy Bridge (C) Intel i7.

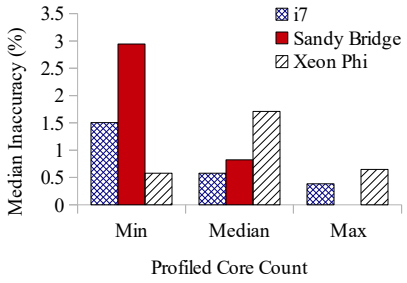

Fig. 14. By architecture, inaccuracy averaged across core counts.

number of threads based on offline and online method. Core scaling is combined with and scaling of resources in [11] with a focus on power-constrained processors. [32] proposed a method to utilize concurrency levels and DVFS to achieve optimal energy efficiency configuration for a workload. The goal is to achieve maximum performance within a given power budget. In contrast to these works, our study is a unique empirical study of the effects of core scaling on peak power.

Peak power has also been explored by several studies, and it can be divided in two broad focuses, (1) Power capping (2) Peak power prediction/estimation. Works such as [19], [9], [8], [24], [14] propose models and dynamic techniques

to keep power consumption under a budget. On the other hand, peak power is relatively less explored by the research community. Performance profiling and performance prediction for better scheduling decisions in data-centers has been explored by [5], [34], [16], [23]. A key challenge is profiling performance, power, and/or answer quality quickly in data centers that support heterogeneous hardware and software. Our study complements these works by proposing a peak power profiling and prediction framework. By combining the observations from peak-power profiling and core scaling, our proposed framework exploits the consistent nature of power usage across workload phases to provide accurate peak power prediction at a low overhead.

VIII. Conclusion

In this study, we analyzed power traces from profiling core scaling on multiple high performance computing benchmarks over 3 multicore architectures. We found that peak power exhibited up to 30% variation between workloads on the same architecture. We saw that a workload which was a good predictor of peak power for a target workload was not guaranteed to be a good predictor of peak power for that same workload on another architecture. We show that profiling for k% of a target workload results in variable accuracy across workloads. However, k% profiling set at 40% resulted in less than 5% inaccuracy for most workloads. Our key insight was that for the same workload on the same architecture, the traces of peak power showed similar phases occurring near the same points in the normalized runtime between different numbers of active cores. We developed an algorithm that used this key insight to reduce the amount of time k% profiling needed to attain approximate peak power profiles across multiple core counts. Using this method, we could save up to 93% profiling time with less than 3% increase in median inaccuracy.

IX. Acknowledgments

This work was also supported by and used the resources of the National Center for Computational Sciences at the Oak Ridge National Laboratory managed by UT Battelle, LLC for the U.S. DOE (contract No. DE-AC05- 00OR22725) and NSF grants CAREER CNS-1350941 and CNS-1320071.

References

[1] Minife summary v 2.0. https://asc.llnl.gov/CORAL-benchmarks/Summaries/MiniFE_Summary_v2.0.pdf, June 2014.

[2] D. Bailey et al. The nas parallel benchmarks. In *RNR Technical Report RNR-94-007*, 1994.

[3] L. Bautista-Gomez et al. Reducing waste in extreme scale systems through introspective analysis. In *IEEE IPDPS 2016*.

[4] R. Bertran, M. Gonzelez, X. Martorell, N. Navarro, and E. Ayguade. A systematic methodology to generate decomposable and responsive power models for cmps. In *IEEE Transactions on Computers*, 2012.

[5] C. Delimitrou and C. Kozyrakis. Paragon: Qos-aware scheduling for heterogeneous datacenters. *ACM SIGARCH Computer Architecture News*, 41(1):77–88, 2013.

[6] C. Delimitrou and C. Kozyrakis. Quasar: Resource-efficient and qos-aware cluster management. In *ASPLOS*, 2014.

[7] H. Esmaeilzadeh, E. Blem, R. Amant, K. Sankaralingam, and D. Burger. Dark silicon and the end of multicore scaling. In *ISCA*, 2011.

[8] M. Etinski et al. Optimizing job performance under a given power constraint in hpc centers. In *IGCC*, 2010.

[9] W. Felter, K. Rajamani, T. Keller, and C. Rusu. A performance-conserving approach for reducing peak power consumption in server systems. In *Proceedings of the 19th annual international conference on Supercomputing*, pages 293–302. ACM, 2005.

[10] X. Fu, X. Wang, and C. Lefurgy. How much power oversubscription is safe and allowed in data centers. In *IEEE ICAC*, 2011.

[11] H. Ghasemi and N. Kim. Rcs: runtime resource and core scaling for power-constrained multi-core processors. In *ACM PACT*, 2014.

[12] S. Gupta et al. Understanding and exploiting spatial properties of system failures on extreme-scale hpc systems. In *Int'l Conference on Dependable Systems and Networks (DSN) 2015*.

[13] J. Hardman. Nas parallel benchmarks. http://www.nas.nasa.gov/publications/npb.html, Mar. 2012.

[14] H. Hoffmann and M. Maggio. Pcp: A generalized approach to optimizing performance under power constraints through resource management. In *International Conference on Autonomic Computing*, 2014.

[15] Intel. Intel xeon phi coprocessor. http://www.colfax-intl.com/nd/downloads/Xeon-Phi-Coprocessor-Datasheet.pdf, Apr. 2014.

[16] J. Kelley, C. Stewart, N. Morris, D. Tiwari, Y. He, and S. Elnikety. Measuring and managing answer quality for online data-intensive services. In *International Conference on Autonomic Computing*, 2015.

[17] M. Kicherer, L. G. Lima, and R. DelValle. Github anyc librapl. https://github.com/anyc/librapl, Apr. 2013.

[18] T. Kidd. Intel xeon phi coprocessor power management configuration: Using the micsmc command-line interface. https://software.intel.com/en-us/blogs/2014/01/31/intel-xeon-phi-coprocessor-power-management-configuration-using-the-micsmc-command, Jan. 2014.

[19] T. Komoda et al. Power capping of cpu-gpu heterogeneous systems through coordinating dvfs and task mapping. In *ICCD, 2013*.

[20] R. Moore and B. Childers. Using utility prediction models to dynamically choose program thread counts. In *IEEE Int. Symp. Performance Analysis of Systems Software*, 2012.

[21] H. Nguyen, Z. Shen, X. Gu, S. Subbiah, and J. Wilkes. Agile: Elastic distributed resource scaling for infrastructure-as-a-service. In *International Conference on Autonomic Computing*, 2013.

[22] B. Nie et al. A large-scale study of soft-errors on gpus in the field. In *IEEE International Symposium on High Performance Computer Architecture (HPCA), 2016*, 2016.

[23] S. Ren, Y. He, S. Elnikety, and K. S. McKinley. Exploiting processor heterogeneity in interactive services. In *ICAC*, pages 45–58, 2013.

[24] O. Sarood, A. Langer, A. Gupta, and L. Kale. Maximizing throughput of overprovisioned hpc data centers under a strict power budget. In *IEEE Supercomputing*, 2014.

[25] R. Schone, D. Hackenberg, and D. Molka. Memory performance at reduced cpu clock speeds: An analysis of current x86_64 processors. In *HOTPOWER*, 2012.

[26] R. Smith. Intel's knights landing co-processor detailed. http://www.anandtech.com, 2014.

[27] M. Suleman, M. Quresh, and Y. Patt. Feedback-driven threading: Power-efficient and high-performance execution of multi-threaded workloads on cmps. In *ACM ASPLOS*, 2008.

[28] K. Tang et al. Power-capping aware checkpointing: On the interplay among power-capping, temperature, reliability, performance, and energy. In *Int'l Conference on Dependable Systems and Networks (DSN) 2016*.

[29] D. Tiwari et al. Reliability lessons learned from gpu experience with the titan supercomputer at oak ridge leadership computing facility. In *Supercomputing (SC) 2015*.

[30] D. Tiwari et al. Understanding gpu errors on large-scale hpc systems and the implications for system design and operation. In *International Symposium on High Performance Computer Architecture (HPCA), 2015*.

[31] J. Treibig. likwid: Lightweight performance tools. https://code.google.com/p/likwid, 2010.

[32] S. Wang et al. Application configuration selection for energy-efficient execution on multicore systems. *Journal of Parallel and Distributed Computing*, 87:43–54, 2016.

[33] A. Wierman, Z. Liu, I. Liu, and H. Mohsenian-Rad. Opportunities and challenges for data center demand response. In *IEEE IGCC*, 2014.

[34] Z. Xu, N. Deng, C. Stewart, and X. Wang. Cadre: Carbon-aware data replication for geo-diverse services. In *International Conference on Autonomic Computing*, 2015.

[35] H. Zhang and H. Hoffmann. Maximizing performance under a power cap: A comparison of hardware, software, and hybrid techniques. In *ASPLOS*, 2016.