Understanding the Impact of Dynamic Power Capping on Application Progress

Srinivasan Ramesh* Swann Perarnau† Sridutt Bhalachandra† Allen D. Malony* Pete Beckman†
*University of Oregon {sramesh, malony}@cs.uoregon.edu
†Argonne National Laboratory {swann, sriduttb}@anl.gov, beckman@mcs.anl.gov

***Abstract*—Electrical power has become an important design constraint in high-performance computing (HPC) systems. On future HPC machines, power is likely to be a budgeted resource and thus managed dynamically. Power management software needs to reliably measure application performance at runtime in order to respond effectively to changes in application behavior. Execution time tells us little about how the *science* in the application is progressing toward an application-defined end goal. To the best of our knowledge, no study has defined or categorized online application progress in the context of power management. Based on semi-structured interviews with HPC application-specialists, we define an *online* notion of progress—an application-specific metric that can be monitored at runtime to provide a sense of the rate at which application *science* is being performed. Using instrumentation, we characterize and categorize the progress of various production scientific applications and benchmarks. We propose a model of the impact of dynamic power capping on application progress. By experimental evaluation, we show that our model accurately captures the general behavior of the progress of different classes of applications under a power cap. We believe that such a model is an important first step toward the design of more dynamic power management policies for HPC systems.**

***Index Terms*—dynamic power management, online performance characterization, application progress, power capping**

I. Introduction

Power has been a critical design constraint at the processor level for a decade now. Power constraints in the processor have resulted in the notion of *dark silicon*—the entire chip cannot be run at full power without overheating. As a result, hardware employs techniques to manage power usage in the background. The Intel Turbo-Boost technology is a prime example of a built-in power management infrastructure on the processor.

Power has now become a critical design constraint at the high-performance computing (HPC) system level. This is due primarily to the fact that IT infrastructure and electrical power needs are becoming economically infeasible. In this context, one can reasonably expect future HPC systems to be *power budgeted*. Consequently, the limited power in such systems needs to be managed effectively.

The broad goal of power management in the context of HPC is to optimize performance under a power budget. The power management system needs to be *dynamic* in order to respond to changes in the system and application behavior. A key requirement of a dynamic control system is to reliably measure the *online performance* of the application. Traditionally, performance has been defined relative to observable metrics about the program's code execution, such as instructions per cycle, execution time, and cache misses. However, this definition of performance fails to capture the application-level concept of "progress", which is a more relevant metric for how well the system is delivering compute value. Thus, in this study, we use the terms *online performance* and *progress* interchangeably.

Dynamic power management requires online performance to be monitored as the application is executing. An application-specific definition of online performance would give the power management infrastructure additional knowledge to make informed decisions regarding the optimal budgeting of power in the system. Specifically, we seek answers to the following questions:

- How do we describe online performance such that it precisely conveys how a system is delivering scientific value, in an application-specific, system-agnostic manner?
- What are the challenges involved in employing online performance for the purpose of generating a performance baseline?
- By employing power capping as a means to limit node power, how do we model its impact on online performance?

To the best of our knowledge, our research is the first extensive study that defines and categorizes online performance (i.e., progress) for production HPC applications in the context of power management. The paper makes the following contributions:

- Based on semi-structured interviews with HPC application scientists, we define, categorize, extract, and characterize progress for a set of production HPC applications.
- We study and model the impact of dynamic power capping schemes on progress for this set of applications.
- We show that our model accurately captures the impact of dynamic power capping for different classes of applications.

The rest of this paper is organized as follows. Section II provides the motivation for characterizing online performance. Section III defines and categorizes online performance for the selected applications in our study. Section IV describes the methodology used to extract progress. Section V describes the dynamic power capping schemes and studies their impact on progress. Section VI models and evaluates the impact of dynamic power capping on progress. Section VII describes the related work. Section VIII summarizes our conclusions and briefly discusses directions for future work.

II. Motivation

Argo is an on-going project of the U.S Department of Energy to design and develop low-level system software for future exascale systems. As a part of the Exascale Computing Project (ECP), Argo aims to adapt, extend, and improve the low-level HPC system software stack, based on the current expectations of exascale architectures and workloads.

On the power management side, the Argo project leverages the hierarchical nature of typical HPC systems to provide a comprehensive approach to dynamic power management [1]. At its core, this approach relies on a hierarchy of control loops that cooperate across system, job, and compute nodes to improve both power consumption and application throughput. At the top, a system controller monitors power across the entire machine and distributes power budgets across the jobs. Inside each job, this power budget is then distributed to nodes, according to application characteristics and node variability. On each node, a daemon is in charge of enforcing the power budget while maximizing performance. The recently started HPC PowerStack community-driven effort is advocating for a similar power management infrastructure.

At the node level, the focus of this paper, a node resource manager (NRM) implements our resource management policies. This component uses direct access to hardware to measure and control power. It is ultimately responsible for the enforcement of a power budget received from higher levels in the hierarchy, while improving application performance. On the control side, various techniques are available to the NRM to enforce a power budget, including dynamic voltage frequency scaling (DVFS), dynamic duty cycle modulation (DDCM), and dynamic hardware power capping methods such as Intel's running average power limit (RAPL).

In order to design efficient power management policies, the NRM must also be able to monitor both power and application performance. We argue in this paper that more efficient and more dynamic control policies could be designed if application performance in particular could be easily monitored *online*. Among the policies we envision are the following:

- In response to an increasing system load, the NRM receives gradually decreasing power budgets and chooses the optimal strategy that respects the power budget with the least impact on performance.
- A large, high-priority job begins executing elsewhere on the system, and the power budget for the currently executing low-priority job is reduced. The NRM responds to this reduced power budget for the low-priority job by implementing a hard, immediate power cap on the node.

Past studies [2]–[5] have focused largely on modeling the impact of power-limiting techniques on execution time or a static application metric output at the end of the run. These studies form the basis for the work we present here. When dealing with the question of dynamic power management, however, they fall short in the following ways:

- They do not describe a way to monitor performance at runtime that can help make dynamic decisions within the power management system.
- Execution time tells us nothing about the online characteristics of the application:
  - It is unclear whether the application performs work at a fixed rate.
  - It is uncertain whether the application suffers from inherent instability in terms of the rate at which work is performed.
  - It misses power management opportunities within fine-grained demarcations such as phases.

Hardware counters that represent instructions per cycle (IPC), floating-point instructions per second (FLOPS), and million instructions per second (MIPS) are popular measures of application performance. IPC, for example, is a good indication of how efficiently the hardware is being utilized. We argue that hardware counters are useful when debugging platform-specific performance problems but are not always appropriate measures of progress.

To illustrate this point, we design a simple MPI code sample shown in Listing 1. The code executes a *do_work()* routine for a *fixed* number of iterations. There are two variations of the *do_work()* routine corresponding to situations where the load is balanced and when it is not. For both variations of the *do_work()* routine, the highest value MPI rank always lies on the critical path. For our code sample, we assign one work unit for every microsecond that a process spends inside *sleep()*. The process with the highest MPI rank is on the critical path and always performs a fixed 1,000,000 units of work.

This application's goal is to execute a fixed number of iterations of some pre defined, arbitrary work. This situation is atypical of many iterative HPC applications. There are at least two ways to define online performance:

- Definition 1: The number of iterations of the outer for-loop executed per second
- Definition 2: The total work units executed in a certain period of time

The former definition of online performance is relevant to situations where additional parallelism does not always mean that the additional work done is used. Consider a graph search problem where multiple processes take different paths to search for an element. The search terminates when one (or more) process finds the element. From an application performance perspective, one cares about how quickly the element is found. The latter definition is the more general case. Additional parallelism means that a larger problem can be solved or a given problem can be solved faster. Table I depicts that the MIPS metric is not correlated with Definition 1 of online performance. Along with the useful work, the MIPS metric captures the *wasted* cycles due to busy waiting spent at the MPI barrier as a result of load imbalance. However, this has no bearing on the online performance of the application, which remains at a constant one iteration per second regardless of the *do_work()* routine used.

Listing 1: MPI workload imbalance

```
/* Leads to load imbalance */
void do_unequal_work(int rank, int size) {
   float sleeptime =
       ((float)rank/size)*1000000.0;
   usleep((int)sleeptime);
}

/* Load is balanced */
void do_equal_work(int rank, int size) {
   usleep(1000000);
}

int main(int argc, char** argv) {
   int i; double start, end;

   MPI_Init(NULL, NULL);

   // Get the rank of the process
   int world_rank, world_size;
   MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
   MPI_Comm_size(MPI_COMM_WORLD, &world_size);

   for(i = 0; i < 5; i++) {
     start = MPI_Wtime();

     do_(un)equal_work(world_rank+1,
         world_size);

     MPI_Barrier(MPI_COMM_WORLD);

     end = MPI_Wtime() - start;

     if(!world_rank) printf("PROGRESS is %f
         iterations per second\n", 1.0/end);
   }

   MPI_Finalize();
}
```

III. Defining online performance

Many scientific applications define a numerical value, called the figure of merit (FOM), that represents the performance or quality of results in a way that relates to the science being carried out. This value is usually generated at the end of the execution. For example, the FOM for many climate-modeling codes is the *simulated years per day*, calculated as the number of simulated years divided by the time taken to run the application (in days). The FOM is almost always derived from the execution time. Note that the FOM may be different from the metric that represents the quality of the scientific result. For example, the FOM for iterative solvers involves the number of iterations executed per second. However, the key measure of the quality of the solution is the *relative residual norm*. This is a measure of error in the system.

TABLE I: Correlation between MIPS and online performance

No. of MPI Processes	*do_work* Routine	Online Performance: Definition 1 (iterations/second)	Online Performance: Definition 2 (work units/second)	MIPS
24	do_equal_work	0.998	4, 800, 000	4,115.5
24	do_unequal_work	0.998	2, 400, 000	79,724.1

TABLE II: Description of applications

Application	Description
QMCPACK	Monte Carlo quantum chemistry code that samples particle positions randomly. Phased application.
OpenMC	Monte Carlo neutron transport code that simulates particle movement inside nuclear reactor. Phased application.
AMG	Iterative solver benchmark that uses algebraic multigrid preconditioning. Only the solve phase is important for performance.
LAMMPS	Molecular dynamics package that uses N-body simulation techniques. No detected phases in the application.
CANDLE	Deep Learning based cancer suite. Benchmark code that uses TensorFlow [6] to solve problems related to precision medicine for cancer.
STREAM	Memory bandwidth benchmark designed to stress-test the memory subsystem.
URBAN	Collection of applications for modeling and simulation of city infrastructure and transport mechanisms. Multiphysics application where individual components run at different timescales.
Nek5000	Computational fluid dynamics library that is a part of larger applications.
HACC	Cosmology application that uses N-body techniques for simulation of galaxies. Many individual components with distinct performance characteristics.

To study online performance, we identify three objectives that are relevant to power management:

- Online performance relates directly to the *science* being performed within the application.
- If the application defines an FOM, the online performance is correlated with this FOM. If not, it is correlated with the execution time.
- The value of the online performance metric is reasonably consistent during execution.

The ECP applications and benchmarks that we consider represent a broad range, from heavily compute to memory-bound (Table II). We set up discussions with application specialists to gather their responses to questions in Table III.

A. Gathering insights from application specialists

Table IV summarizes the responses to the questions in Table III. In the context of defining online performance, QMCPACK [7], OpenMC [8], LAMMPS [9], and STREAM [10] are similar. They are all loop-based applications that have a fixed amount of work to perform, specified by the input. Execution time for these applications can be determined by gathering the input requirements, since the work per loop or timestep iteration is fixed. The online performance can be accurately determined by the number of iterations (or equivalent metric) executed in a given period of time.

We choose to study iterative solvers because they form an important part of many scientific applications. AMG and CANDLE [11] share several commonalities. Online performance can be measured, but the number of iterations cannot be predicted in advance for these loop-based codes. Further, online performance for both these applications does not relate directly to the science being performed. For example, although not currently supported, the training phase of the CANDLE

TABLE III: Questions posed to application specialists

Question Number	Question
1	Is there a well-defined FOM for the application?
2	Can we measure online performance during execution that correlates well with either FOM or the execution time?
3	Does online performance measure progress toward an application-defined scientific goal?
4	Is the execution time accurately predictable based on a performance model of the application?
5	If the application is loop based, is the number of loop iterations decided prior to execution?
6	If application is loop based, do loop iterations proceed in a uniform manner in terms of instructions executed?
7	Does the application have multiple phases or components that are clearly demarcated from a design or performance characteristic standpoint?
8	What system resource is the application limited by?

TABLE IV: Summary of responses

Application	Question							
	1	2	3	4	5	6	7	8
QMCPACK	Y	Y	Y	Y	Y	Y	Y	Compute
OpenMC	N	Y	Y	Y	Y	Y	Y	Memory latency
AMG	Y	Y	N	N	N	Y	N	Memory bandwidth
LAMMPS	Y	Y	Y	Y	Y	Y	Y	Compute
CANDLE	Y	Y	N	Y	Y	Y	Y	Memory bandwidth
STREAM	Y	Y	Y	Y	Y	Y	N	Memory bandwidth
URBAN	N	-	-	-	-	N	Y	Component-dependent
Nek5000	N	-	-	-	Y	N	Y	Compute
HACC	Y	N	-	N	Y	N	Y	Compute

benchmark can be bounded by accuracy. In this case, the number of *epochs* required for training to complete cannot be predicted. However, online performance can be defined as the number of epochs completed in a given period of time.

HACC [12] and Nek5000 [13] are large applications that are also loop based. Specifically, Nek5000 is used as a library in computational fluid dynamics applications. For these applications, however, the number of timesteps per second cannot be used to measure online performance reliably because this metric does not stay uniform during the execution. Moreover, such a high-level metric provides little insight into the progress of the science as the application executes. In the URBAN project, the Nek5000 library is used with Energy Plus (a building energy simulation suite) to study the effect of external temperature changes on the energy-efficiency of buildings. Nek5000 and Energy Plus run at timescales that are orders of magnitudes apart. We could define the online performance of URBAN using an arbitrary metric such as the number of buildings simulated per second. This definition, however, has little meaning in the context of power management because it does not translate to the performance of its key component applications.

B. Categorizing applications

Based on the summary of responses in Table IV, we find that applications can be broadly categorized based on their similarities. Table V summarizes the definition and categorization of online performance for the selected applications.

- Category 1: Most HPC applications, simulations, and benchmarks that have iterative loops fall in this category. For such applications, there is a clear, well-defined measure of online performance that correlates well with the application-specific scientific goal. Usually, this metric also correlates well with the FOM of the application if it is defined.
- Category 2: These are timestep-based applications where the online performance is well defined but does not correlate well with the scientific metrics of interest. For this reason, one cannot determine how far the application has progressed toward its goal.
- Category 3: Applications in this category are characterized by one or both of the following properties:
  - Online performance cannot be monitored reliably.
  - The application is composed of multiple components that limit the usefulness of a single metric.

  This class of applications requires a multifaceted definition or an appropriate composition of the progress of individual components that is out of the scope of this study.

TABLE V: Categorizing applications and defining online performance

Application	Category	Online performance Metric
QMCPACK	1	Blocks per second
OpenMC	1	Particles per second
AMG	2	Conjugate gradient iterations per second
LAMMPS	1	Atom timesteps per second
CANDLE	1/2	Epochs per second (training phase)
STREAM	1	Iterations per second
URBAN	3	N/A
Nek5000	3	N/A
HACC	3	N/A

IV. Extracting and characterizing progress

Now that we have a definition of progress for the applications in our study, we would like to corroborate the findings from our research discussions with experiments that characterizes applications and the behavior of their progress.

A. Characterizing applications

The β metric is a measure of the compute-boundedness of an application introduced by [5]. Its value lies between 0 and 1: a high value indicates that the application is compute-bound. A previous work [14] reports that this value shows less than 5% of variation with CPU frequency. We do not verify this claim here. Rather, we use the execution time at the maximum frequency of 3300 MHz and the execution time at 1600 MHz to calculate this value.

Misses Per Operation (MPO) is another metric used to characterize applications. MPO is frequency independent, making it more reliable than the β metric. We use PAPI [15] to calculate MPO for an application by dividing the total L3 cache misses (`PAPI_L3_TCM`) by the total instructions executed (`PAPI_TOT_INS`). A high MPO suggests that the application is memory bound.

Table VI displays the experimentally observed values for the β and MPO metrics for the applications in our study. We see a

good correlation between the MPO and the β metric for all the applications. Although there exist other ways to characterize applications, we focus on the β metric in particular because our model builds on past work that uses this measure.

B. Extracting progress

Our study involves extracting and monitoring the progress of parallel applications running on a single node. Each of the applications described in this study was instrumented at the source-code level to publish its *online performance* metric (refer to Table V) using the publish-subscribe ZeroMQ [16] sockets.

1) LAMMPS: We use the Lennard-Jones benchmark that is provided as a part of the LAMMPS package. Pure MPI is used to parallelize the application using 24 processes on the node and MPI process pinning is enabled. The application simulates a fixed number of 40,000 atoms. LAMMPS runs an outer timestep loop that encloses the main parallel section of the program. We measure the amount of time required to execute one timestep in the `VERLET` run function. This number is multiplied by the number of atoms simulated to report progress as a single value for the application. Note that the rate at which progress is reported depends entirely on the rate of execution of the timestep. For the single node setup, progress is reported roughly 20 times a second. These values are collected and averaged once every second.

2) AMG: AMG is a solver benchmark that uses iterative solvers from the HYPRE [17] library. We use the AMG preconditioner with solver 3: GMRES with diagonal scaling along with *pooldist* ID 1 for our experiments. Pure MPI is used to parallelize AMG using 24 processes, one per physical core. Process pinning is enabled. The GMRES iterations are managed and implemented within the HYPRE library. We measure progress as the number of GMRES iterations executed in a given period of time. The GMRES iterative loop encloses the performance-critical parallel section of the solver. Progress is reported as a single value for the application. For this setup, progress is reported approximately 3 times a second.

3) QMCPACK: The *performance-NiO* benchmark that is a part of the QMCPACK package has three phases: VMC 1, VMC 2, and DMC. The DMC is particularly important for performance benchmarking of this application. The benchmark is parallelized by using pure OpenMP on the node. For these experiments, 24 pinned OpenMP threads are used. The number of *steps per block* is set to 15, and the *number of blocks* is set to 3,000 for the DMC in order to ensure that the application executes for a long enough period of time. QMCPACK continuously monitors and reports the number of blocks executed. This reporting is done at a level outside the main parallel section of the code. We leverage this reporting framework to monitor and report progress as the number of blocks completed in a given period of time. This number is reported approximately 16 times a second for the DMC.

4) STREAM: OpenMP is used to parallelize STREAM using 24 threads, one per physical CPU core. STREAM executes four operations: copy, scale, add, and triad for a fixed number of iterations specified in the input. The iterative loop is instrumented to report progress as a single value for the application, once per iteration. The loop encloses the OpenMP parallel sections for the three operations. Progress is reported approximately 16 times a second.

TABLE VI: β and MPO metrics for selected applications

Application	β Metric	MPO Metric ($\times 10^{-3}$)
QMCPACK (DMC)	0.84	3.91
OpenMC (Active)	0.93	0.20
AMG	0.52	30.1
LAMMPS	1.00	0.32
STREAM	0.37	50.9

5) OpenMC: OpenMC is a neutron-particle simulation code that has two phases: *active* and *inactive*. The number of particles to simulate along with the number of *batches* to run fully specifies the amount of work in the application. The loop for the batch computation encloses the loop for the particles. The particle loop is parallelized by using 24 OpenMP threads. Progress is reported at the batch-loop level, once after each batch computation completes. Note that progress is reported as one value for the application. We use 10 inactive batches and 300 active batches to simulate 100,000 particles. Progress is reported approximately once every second.

CANDLE is built by using TensorFlow, a Deep Learning suite that reports the number of epochs completed during the training phase. Extracting the online performance for CANDLE as the number of epochs completed in a certain period of time involves instrumenting TensorFlow. We do not present a description for extracting progress from the CANDLE application. We faced significant technical issues in installing and using TensorFlow from the source on our testing platform. Installing TensorFlow from the source is a known difficulty among the data-science community. We thus had to resort to using the pre built TensorFlow binaries for our study. As a result, access to the source that generates the training progress information was not possible. In principle, however, extracting online performance for CANDLE involves the same procedure as that described for other applications in our study.

Although the study that we present here is restricted to a single node, we note that this directly maps to a multi node study without any change. Transposing this notion of progress in order to monitor it at a per-processing-element level is part of future work.

C. Characterizing progress

We characterize the online performance of the selected applications. LAMMPS and STREAM display consistent behavior for their respective online performance metrics. Figure 1 (left) depicts this consistent online behavior of LAMMPS: online performance remains at 1080 atom timesteps per second throughout the execution. On the other hand, the online performance of AMG is inconsistent (Figure 1, center). Online

performance for AMG fluctuates between 2.5 and 3 iterations per second and needs to be averaged out.

For QMCPACK, consider the *performance-NiO* benchmark. Recall that QMCPACK is an application belonging to Category 1. The benchmark has three distinct phases: VMC1, VMC2, and DMC. Each of these phases, however, could have a different number of blocks to compute and distinct performance characteristics. As depicted in Figure 1 (right), when we define and monitor blocks per second at runtime, the phases are clearly distinguishable from one another as they compute blocks at different rates. This information is missed when we consider static definition of performance output at the end of the run. OpenMC displays a similar phased behavior: we do not show OpenMC's online performance due to a lack of space.

V. Power Capping and Application Progress

Dynamic power capping is a requirement for this study for two reasons:

- Application progress is measured at runtime, so static techniques are not sufficient in order to study the impact of power capping on this metric.
- We want to study the effect of possible interactions between the node resource manager and power management software at higher levels within the hierarchy.

We begin this section by describing the software we used in our study.

A. Software

1) Running Average Power Limit: RAPL is Intel technology that allows users to specify power caps on hardware *domains* by using model-specific registers (MSRs). Commonly exposed domains include the *package* and *DRAM* domains. When given a power cap and time window, the processor ensures that the *average* power over the time window is maintained. Rountree et al. [18] offer a good explanation of how RAPL can be used to measure and limit power usage. We use RAPL in this study to implement our dynamic power capping schemes on the *package* domain.

RAPL is a proprietary interface to power management on the chip. Given a power cap, it budgets power between the core (processors, caches) and uncore (off-chip) components that make up the package domain. However, we have access to power usage only at the package level. To the best of our knowledge, no published work accurately describes or models RAPL's internal behavior.

Figure 2 depicts CPU frequencies for two applications—LAMMPS and STREAM—under a RAPL-based power cap. LAMMPS is a compute-bound application, whereas STREAM is a memory-bound benchmark. Under identical power caps, RAPL employs a higher CPU frequency for compute-bound applications and thus distributes more power to the core components. In other words, RAPL performs application-aware power management.

2) libmsr: The *libmsr* library from Lawrence Livermore National Laboratory provides an easy-to-use interface to the model-specific registers on Intel hardware. We use libmsr along with the msr-safe [19] module to implement our dynamic power-capping schemes and monitor power usage without the need for root access.

B. Dynamic power-capping schemes

The dynamic power-capping schemes we present here were implemented by using a *power-policy* tool that was developed as a part of this study. The *power-policy* tool runs as a background daemon on the node. It monitors power usage and applies the selected dynamic power-capping scheme on the package domain once every second. We present below the power-capping schemes that we have designed.

- Linearly decreasing power cap – Initially, the power on the node is uncapped, and a linearly decreasing power cap is applied until a system or user-specified minimum value is reached.
- Step-function power cap – The power cap on the node alternates between an uncapped (or high value) and a low value.
- Jagged-edge function power cap – The power cap on the node linearly decreases from an uncapped level to a low value and then goes back to an uncapped level quickly.

C. Impact of dynamic schemes on online performance

In the interest of space, we present the impact of applying these dynamic power capping schemes on three applications: LAMMPS, QMCPACK (performance-NiO benchmark, DMC), and OpenMC (active). Figure 3 depicts the impact of these three different schemes. These experiments were run on a single node and all the applications were parallelized to utilize all available CPU cores. The following observations can be made from the results of this experiment:

- The online performance of the application follows the power capping function being applied. We found this to be true regardless of the application being studied or the power capping function being applied.

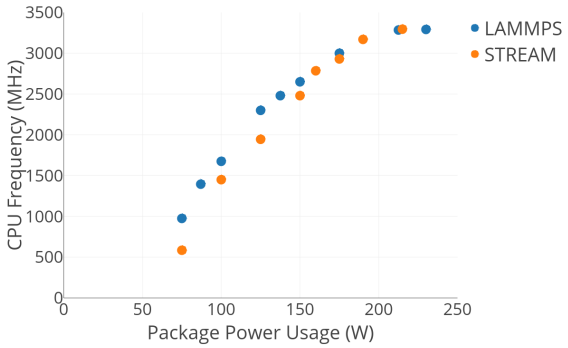


Fig. 2: RAPL: Application aware power management

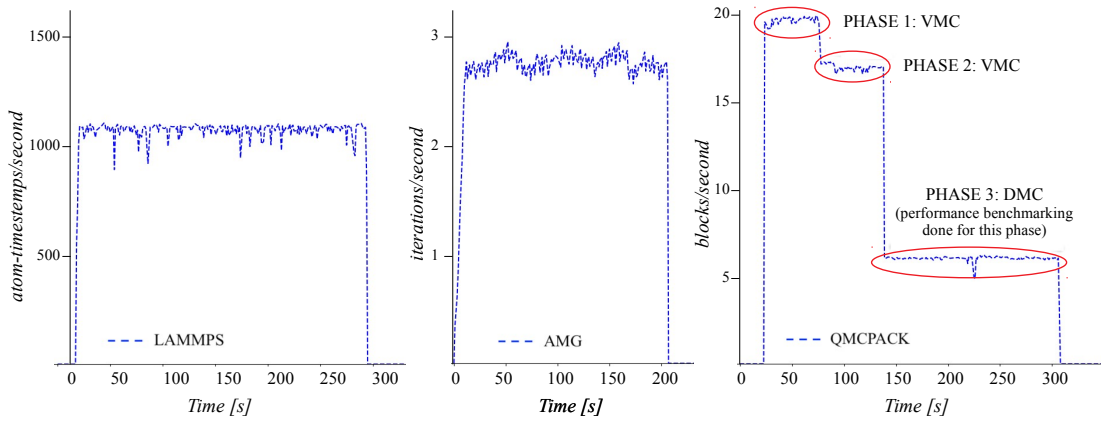


Fig. 1: Characterizing online performance

- Upon careful observation of the collected progress values, we note that the instantaneous value of the progress metric is affected by the instantaneous value of the power cap. Figure 3 supports this observation.

For OpenMC, the progress metric is occassionally reported as zero, as shown in Figure 3. We note that is due to a flaw in the design of the ZeroMQ-based progress monitoring framework and not a characteristic of the application itself.

VI. Modeling and Evaluation

We would like to quantify and model the impact of RAPL-based power-capping on progress in order to do the following:

- Validate and improve upon assumptions about RAPL's behavior for different application characteristics.
- Be able to *predict* the impact of power-capping on progress given a description of application characteristics. This is the first step in the design of power policies at the node level.
- Be able to decide on the exact power budget to be employed given an expectation of online performance.

The impact of DVFS of execution time is discussed in [4] and [14]. We extend these ideas and models in order to study the impact of power capping on application progress. In particular, we are interested in studying the impact of the *package* domain power capping on application progress.

We rely on experimentally observed behavior to make the following assumptions:

- RAPL distributes the available power between the core and uncore in the ratio equal to their compute-boundedness (refer to Figure 2). We use the β metric introduced by [5] to denote this ratio.
- If a power cap that is lesser than the uncapped power usage of the application is applied, the application uses all the power given to it. We have observed this behavior to be true regardless of the application being studied.

A. Model

We begin with the equation denoting the impact of frequency scaling on execution time described in [4].

$$\frac{T(f)}{T(f_{max})} = \beta * (\frac{f_{max}}{f} - 1) + 1 \tag{1}$$

Here f_{max} represents the nominal maximum frequency of the processor, and β is a value betweeen 0 and 1; an ideal compute-bounded application has a β value of 1.

The power used by the *core* component, P_{core} is related to CPU frequency f as follows.

$$Pcore \propto f^{\alpha} \tag{2}$$

Here α is a value that lies between 1 and 3 [20], and $P_{coremax}$ is the *core* power usage level corresponding to the CPU frequency f_{max}. Denoting the progress at a CPU frequency f by $r(f)$, we note that

$$r(f) \propto \frac{1}{T(f)} \tag{3}$$

We now use a change of variable and denote the progress at a *core power usage* level P_{core} as

$$r(P_{core}) = \frac{r(P_{coremax})}{\beta * ((\frac{P_{coremax}}{P_{core}})^{\frac{1}{\alpha}} - 1) + 1} \tag{4}$$

We denote P_{cap} as the power cap applied to the *package* domain and $P_{corecap}$ as the *effective* power budget employed by RAPL on the *core* component under a package domain power cap P_{cap}. Recalling our assumptions, we have

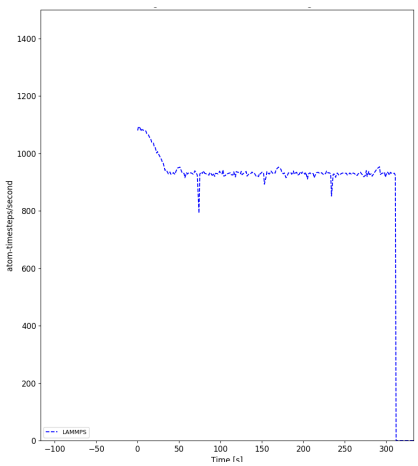
$$P_{corecap} = \beta * P_{cap} \tag{5}$$

and

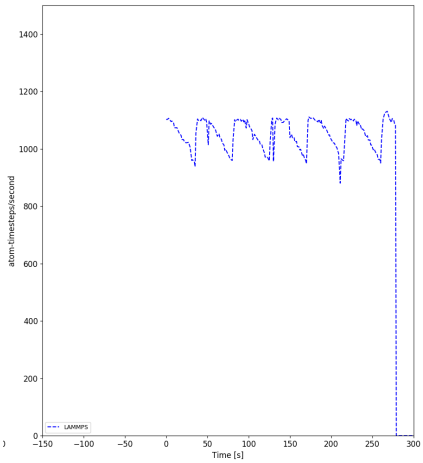
$$P_{core} \approx P_{corecap} \tag{6}$$

We can thus model the *change in progress* with an *effective* power cap on the core, $P_{corecap}$, as follows.

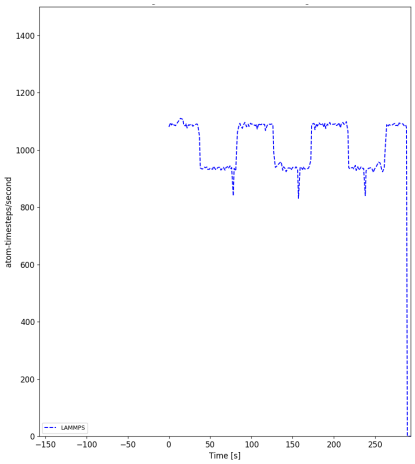
$$\delta_{progress} = r(P_{coremax}) * [1 - \frac{1}{\beta * ((\frac{P_{coremax}}{P_{corecap}})^{\frac{1}{\alpha}} - 1) + 1}] \tag{7}$$



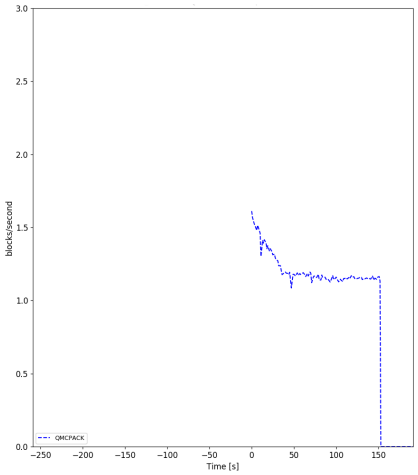
(a) LAMMPS: Linearly decreasing power cap



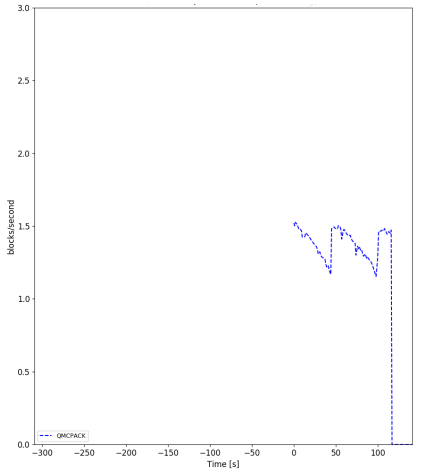
(b) LAMMPS: Jagged-edge power cap



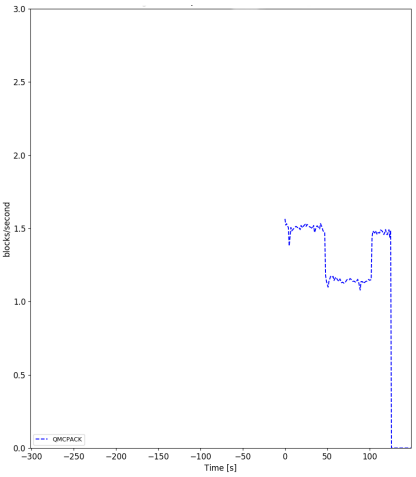
(c) LAMMPS: Step function power cap



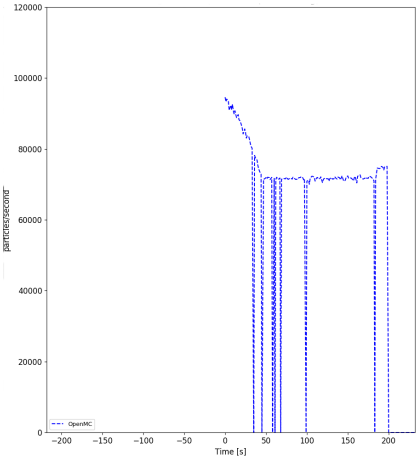
(d) QMCPACK: Linearly decreasing power cap



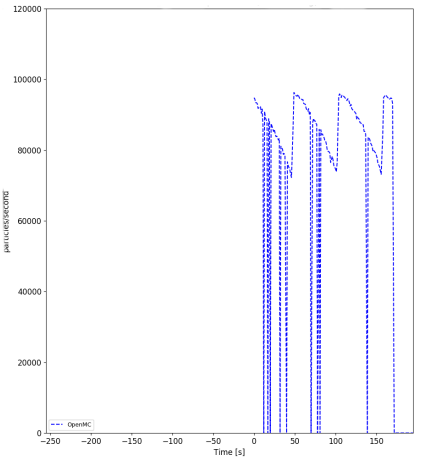
(e) QMCPACK: Jagged-edge power cap



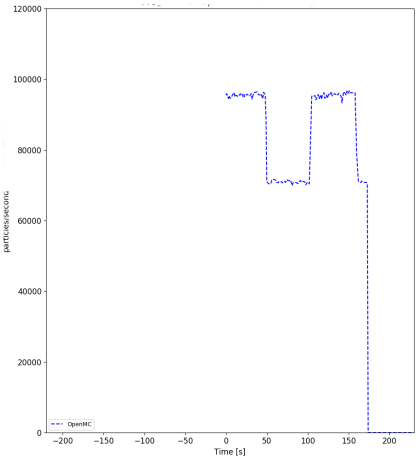
(f) QMCPACK: Step function power cap



(g) OpenMC: Linearly decreasing power cap



(h) OpenMC: Jagged-edge power cap



(i) OpenMC: Step function power cap

Fig. 3: Impact of dynamic power-capping on progress

B. Evaluation

We evaluated the model against the experimentally observed values. We present the results in this section.

1) Experimental Setup: Our experiments were performed on the Chameleon cluster at the Texas Advanced Computing Center. We used a single *compute_skylake* instance for all of our experiments. Each Skylake instance is a dual-socket Intel Xeon Gold 6126 CPU with 12 cores per socket. The Linux kernel version is 3.10.0-862.9.1.el7.x86_64. Each core is hyper-threaded accounting for a total of 48 logical cores per instance. Hyperthreading was turned off for all our experiments. Intel Turbo-Boost was enabled on the instance. The compiler used was gcc-4.8.5, and the MPI version used was mvapich2.2. The application setup is identical to the description in Section 4.

2) Results: Recall the observation (refer to Section 4) that the progress of applications follows the power-capping function being applied. For each of the applications described in this section, we applied the *step-function* dynamic power-capping policy (refer to Section 5) and measured the change in progress when a package power cap was applied. We chose the step-function policy since the power cap (and hence, progress) remains stable for a longer period of time, making it easier to measure the impact on progress. Note that the change in progress is measured when a power cap is applied from an *uncapped* state of execution. The following points are important to note before the results are analyzed:

- For each power cap, five measurements for change in progress were made, and the average value for the change in progress was calculated.
- $P_{corecap}$ is a model-estimated value based on the package power cap.
- α is assumed to have a value of 2 for all model predictions.

As depicted by Figure 4a and Figure 4c the model closely predicts the impact of power-capping on progress for LAMMPS and QMCPACK. For an effective core power cap of 80 W or less, the model predicts the impact on the progress of LAMMPS to within 13.3% of its experimentally observed value. When a more stringent power cap is applied, however, the model underestimates the impact of RAPL-based power-capping for LAMMPS by upto 19%.

For QMCPACK, the model consistently overestimates the impact of power capping: the model has error values ranging from 8.5% for stringent power caps to 53.3% for mid-range power caps. For low power caps, the model performs poorly, overestimating the impact by 250% of the measured value. Figure 4b depicts the measured and model-predicted change in progress with a change in $P_{corecap}$ for AMG. The model, in general, overestimates the impact of RAPL-based power capping on progress. It is not able to account for the "plateaus" in the measured values. The error percentages lie between 13.3% for a power cap of roughly 65 W and 111% for a power cap of 50 W.

For a memory-bound code such as STREAM, the model performs well for power caps of 20 W or less, keeping the error percentage to within 3% of the measured value. It clearly performs badly for stringent power caps. It underestimates the performance impact of RAPL-based power-capping for STREAM by 70.0% for a power cap of 43 W. This is not an entirely surprising result because the model is built upon the assumption that RAPL uses DVFS for power capping. Clearly, RAPL is using additional means to ensure that the power budget is met, and these additional means are not captured by our model.

Because of a lack of space, we present the results of dynamic power capping on the active phase only for the *assembly* example that is used for performance benchmarking on a single node. From Table VI, we see that OpenMC's active phase has a β of 0.94, indicating that it is a CPU-bounded code.

Figure 4e demonstrates that the model is able to closely match the measured values for the change in progress with different power caps. For a power cap in the range of 70 to 140 W, the error percentage varies between a minimum of 3.8% and a maximum of 27.7%. Owing to an unstructured memory access pattern, OpenMC's performance is generally influenced by memory latency. This particular example, however, is sensitive to CPU frequency.

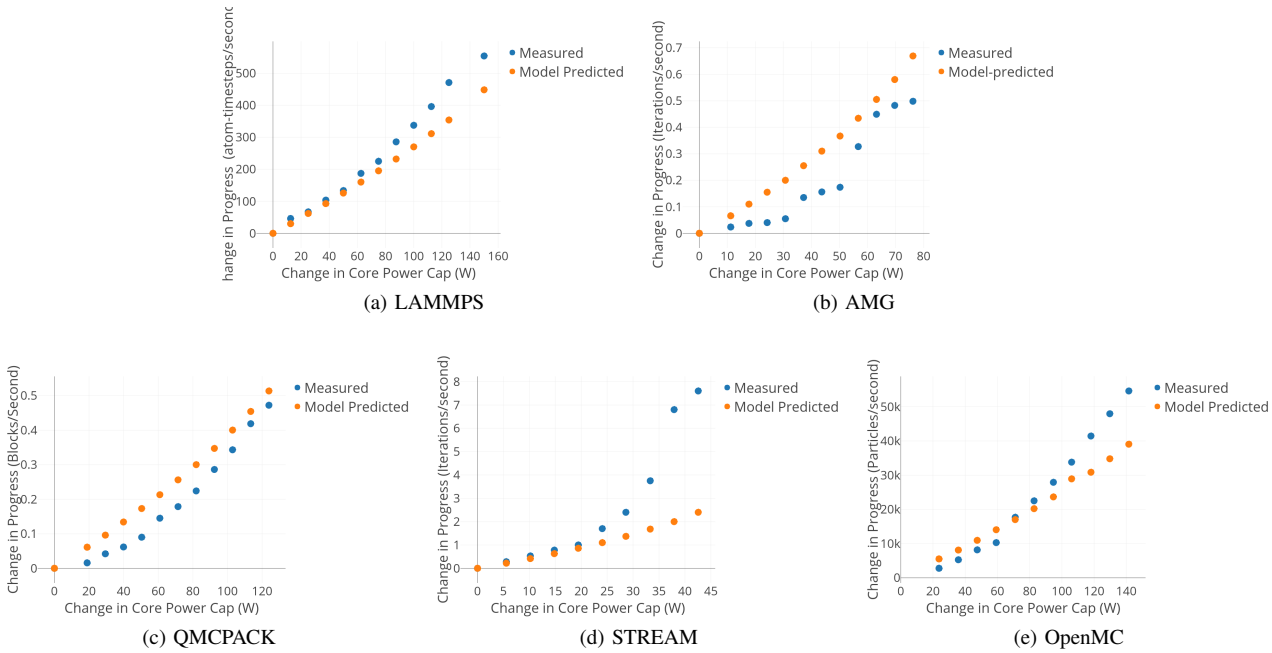
In general, our model performs well for mid-range power caps but performs poorly at the extreme ends of the power cap range. We argue that in production, power-caps are more likely to be in the mid-range, and the model holds merit in this regard. Further, we note that model the performs better overall for CPU-bound code than for memory-bound code.

3) Discussion: The task of managing power efficiently and dynamically on HPC systems demands a high-fidelity measure of online performance of production HPC applications. We have demonstrated how such an online measure of performance can be elicited from semi-structured discussions with application specialists. Online performance is application specific and no single solution exists for defining this quantity. Our categorization of applications helps us get closer to this goal by structuring the information that we have gathered from our discussions.

We have presented a model that attempts to predict the impact of RAPL-based power capping on application progress. Through extensive experimentation, we have shown that this model accurately captures the general behavior of a variety of classes of applications. In evaluating our experimental results, we note potential improvements that could be made to our model's fidelity and our approach to categorization of applications.

We have described applications such as HACC and URBAN for which no single, well-defined, and reliable measure of online performance exists. This situation is atypical of large, multiphysics applications. We can improve upon this by studying individual components separately and modeling progress as a weighted combination of the progress of individual components.

Figure 4d clearly demonstrates that our model fails to predict performance for memory-bound code for stringent



(a) LAMMPS (b) AMG (c) QMCPACK (d) STREAM (e) OpenMC

Fig. 4: Comparison of measured and predicted values for change in progress

power caps. Our model is built on the assumption that RAPL uses DVFS to manage power on the node. Further, we assume that RAPL budgets power between the core and uncore in the ratio of the β metric of the application. In a way, we make an optimistic assumption about RAPL's performance. Figure 5 clearly demonstrates that RAPL is not the best technique to implement power capping for STREAM: DVFS performs better in the range that it is applicable in.

Further, we fix the α value in the model to be 2 for all of our experiments. Our experiments indicate that this value varies between 1 and 4 depending on the range of the power cap being applied. Without having access to fine-grained, reliable core power usage data, we choose to parameterize RAPL to explain its behavior. Our model does not explicitly take into account the other hardware features that RAPL has access to: specifically, DDCM and uncore-DVFS.

VII. Related Work

In this section, we present the related work that influence our research.

A. Defining and characterizing application performance

Traditionally, execution time has been used to define, analyze, and optimize application performance. Interfaces for measuring hardware performance counters such as [15] allow performance introspection at runtime. Numerous tools exist that use such interfaces to provide performance monitoring, introspection, and tuning capabilities. Gustafson and Todi [21] introduce the concept of miniapps to characterize application performance. Carrington et al. [22] study the effectiveness of different metrics in representing application performance. The metrics presented correlate well with the actual application performance, but they give us nothing useful about the science of interest to the application. In other words, hardware counters and other metrics are arbitrary values: scientific information is lost when dealing with defining performance in the context of power management.

B. Modeling power usage and studying impact of power capping on performance

Prior to direct power measurement capabilities becoming mainstream on HPC hardware, several studies have used electrical power meters or hardware counters to provide *average power usage* over a time window. In [23] and [24], the authors present a way to estimate instantaneous (dynamic)

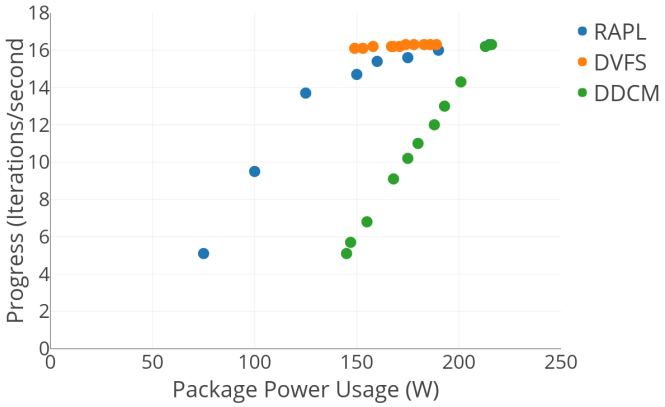


Fig. 5: STREAM: Comparison of different power limiting techniques on progress

power usage using a subset of critical and commonly occurring hardware counters. Powermon [25] operates inside commodity computer systems and analyzes the performance and power consumption tradeoffs in computer applications at a high frequency. Goel and McKee [26] provide a comprehensive analytical model of CPU power usage that includes static and dynamic power usage of core and uncore components.

Haidar et al. [2] study the impact of power capping on application execution time and other application-specific metrics of interest. Bhalachandra et al. [27] show that with power capping, non optimal programs speedup with frequency reduction due to an increase in overall thermal headroom to the critical path. The notion of performance impact in these studies is based on the execution time and not on the actual algorithmic progress discussed in this work.

C. Limiting node power and optimizing performance under a power bound

Petoumenos et al. [28] present a broad survey of a variety of power-capping techniques, both hardware and software schemes. Rountree et al. [18] show that performance variability between compute nodes becomes a highlighted issue in a power-limited HPC environment. A large body of related work seeks to optimize performance (typically execution time) by moving power to nodes on the critical path or by searching for optimal configurations that do not exceed the power budget. Several approaches [29], [30] have explored the benefits of *overprovisioning* HPC systems, limiting power usage in order to add additional hardware and improve application performance at a system level. Ellsworth et al. [31] present an application-agnostic scheme to move power within a power-limited system to where it is most needed. Wang et al. [32] present an application power-aware scheduler based on profiled power usage characteristics. Conductor [33] is a runtime system that finds optimal concurrency and DVFS configurations to adaptively power balance applications at the job level. The use of DDCM to effect energy savings and increase performance in situations where there exists load imbalance in parallel code is explored in [27], [34]. How RAPL affects an application remains to be answered. Thus we focus on modeling the impact of dynamic power capping on application progress.

VIII. Conclusion and Future Work

In this paper, we presented a methodology to monitor and characterize the *online* performance of HPC applications. Based on semi-structured interviews with application specialists, we identified an application-specific notion of *progress* that can be monitored during application execution and that directly relates to both the application performance and its scientific output. Using lightweight instrumentation and simple dynamic power capping schemes, we measured how such a metric is affected by changes in the power caps, and we provided a model of this effect under a RAPL-enforced control scheme. Our model builds on existing studies of the impact of DVFS on execution time and provides a satisfying predictive power of this effect on various classes of applications. We believe that the availability of application progress and this model will provide the key components needed to implement complex dynamic power management policies for future exascale systems.

Nevertheless, our study can be extended in several ways. First, some production applications ended up being too complex to define a single progress metric across the entire workload. In such cases, the resolution of these progress reports or the intrusiveness of the instrumentation might need to be changed. Second, our current model could be improved by dissociating application characteristics such as compute boundedness from the exact control knob being used, by more accurately modeling the relation between power cap and processor behavior. Such improvements could also incorporate online hardware performance monitoring and measurement across a wider range of architecture and hardware power control mechanisms. Also valuable would be a more detailed study of the infrastructure needed for dynamic progress monitoring across large-scale systems and how to combine job wide and node-local progress metrics for power management.

Acknowledgments

We would like to thank Stephanie Brink and Tapasya Patki from LLNL for their help in setting up the libmsr software. Results presented in this paper were obtained by using the Chameleon testbed supported by the National Science Foundation. Argonne National Laboratory's work was supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computer Research, under Contract DE-AC02-06CH11357. This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.

References

[1] D. Ellsworth, T. Patki, S. Perarnau, S. Seo, A. Amer, J. Zounmevo, R. Gupta, K. Yoshii, H. Hoffman, A. Malony *et al.*, "Systemwide power management with Argo," in *IEEE International Parallel and Distributed Processing Symposium Workshops*, 2016.

[2] A. Haidar, H. Jagode, P. Vaccaro, A. YarKhan, S. Tomov, and J. Dongarra, "Investigating power capping toward energy-efficient scientific applications," *Concurrency and Computation: Practice and Experience (Special Issue Paper)*, 2018.

[3] H. Zhang and H. Hoffmann, "Maximizing performance under a power cap: A comparison of hardware, software, and hybrid techniques," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 2, pp. 545–559, 2016.

[4] M. Etinski, J. Corbalán, J. Labarta, and M. Valero, "Understanding the future of energy-performance trade-off via DVFS in HPC environments," *Journal of Parallel and Distributed Computing*, vol. 72, no. 4, pp. 579–590, 2012.

[5] C.-H. Hsu and U. Kremer, "The design, implementation, and evaluation of a compiler algorithm for CPU energy reduction," vol. 38, no. 5, 2003, pp. 38–48.

[6] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "TensorFlow: A system for large-scale machine learning." in *OSDI*, vol. 16, 2016, pp. 265–283.

[7] J. Kim, A. T. Baczewski, T. D. Beaudet, A. Benali, M. C. Bennett, M. A. Berrill, N. S. Blunt, E. J. L. Borda, M. Casula, D. M. Ceperley *et al.*, "QMCPACK: An open source ab initio quantum Monte Carlo package for the electronic structure of atoms, molecules and solids," *Journal of Physics: Condensed Matter*, vol. 30, no. 19, p. 195901, 2018.

[8] P. K. Romano and B. Forget, "The OpenMC monte carlo particle transport code," *Annals of Nuclear Energy*, vol. 51, pp. 274–281, 2013.

[9] S. Plimpton, P. Crozier, and A. Thompson, "LAMMPS large-scale atomic/molecular massively parallel simulator," *Sandia National Laboratories*, 2007.

[10] J. D. McCalpin, "Stream benchmark," *Link: www.cs.virginia.edu/stream/ref.html*.

[11] J. M. Wozniak, R. Jain, P. Balaprakash, J. Ozik, N. Collier, J. Bauer, F. Xia, T. Brettin, R. Stevens, J. Mohd-Yusof *et al.*, "CANDLE/Supervisor: A workflow framework for machine learning applied to cancer research," *BMC Bioinformatics*, 2018.

[12] S. Habib, A. Pope, H. Finkel, N. Frontiere, K. Heitmann, D. Daniel, P. Fasel, V. Morozov, G. Zagaris, T. Peterka *et al.*, "HACC: Simulating sky surveys on state-of-the-art supercomputing architectures," *New Astronomy*, vol. 42, pp. 49–65, 2016.

[13] P. Fischer, J. Kruse, J. Mullen, H. Tufo, J. Lottes, and S. Kerkemeier, "Nek5000: Open source spectral element CFD solver," *Argonne National Laboratory, Mathematics and Computer Science Division, Argonne, IL, see https://nek5000. mcs. anl. gov/index. php/MainPage*, 2008.

[14] V. W. Freeh, D. K. Lowenthal, F. Pan, N. Kappiah, R. Springer, B. L. Rountree, and M. E. Femal, "Analyzing the energy-time tradeoff in high-performance computing applications," *IEEE Transactions on Parallel & Distributed Systems*, no. 6, pp. 835–848, 2007.

[15] P. J. Mucci, S. Browne, C. Deane, and G. Ho, "PAPI: A portable interface to hardware performance counters," in *Proceedings of the Department of Defense HPCMP Users Group Conference*, 1999.

[16] P. Hintjens, *ZeroMQ: messaging for many applications*, 2013.

[17] R. D. Falgout and U. M. Yang, "hypre: A library of high performance preconditioners," in *International Conference on Computational Science*, 2002, pp. 632–641.

[18] B. Rountree, D. H. Ahn, B. R. De Supinski, D. K. Lowenthal, and M. Schulz, "Beyond DVFS: A first look at performance under a hardware-enforced power bound," in *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), IEEE*, 2012.

[19] K. Shoga, B. Rountree, M. Schulz, and J. Shafer, "Whitelisting MSRs with msr-safe," in *3rd Workshop on Exascale Systems Programming Tools, in conjunction with SC14*, 2014.

[20] L. Yu, Z. Zhou, S. Wallace, M. E. Papka, and Z. Lan, "Quantitative modeling of power performance tradeoffs on extreme scale systems," *Journal of Parallel and Distributed Computing*, vol. 84, pp. 1–14, 2015.

[21] J. L. Gustafson and R. Todi, "Conventional benchmarks as a sample of the performance spectrum," in *Proceedings of the Thirty-First Hawaii International Conference on System Sciences*, vol. 7. IEEE, 1998, pp. 514–523.

[22] L. C. Carrington, M. Laurenzano, A. Snavely, R. L. Campbell, and L. P. Davis, "How well can simple metrics represent the performance of hpc applications?" in *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, 2005, p. 48.

[23] R. Rodrigues, A. Annamalai, I. Koren, and S. Kundu, "A study on the use of performance counters to estimate power in microprocessors," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 60, no. 12, pp. 882–886, 2013.

[24] K. Singh, M. Bhadauria, and S. A. McKee, "Real time power estimation and thread scheduling via performance counters," *ACM SIGARCH Computer Architecture News*, vol. 37, no. 2, pp. 46–55, 2009.

[25] D. Bedard, M. Y. Lim, R. Fowler, and A. Porterfield, "Powermon: Fine-grained and integrated power monitoring for commodity computer systems," in *Proceedings of the SoutheastCon*, 2010, pp. 479–484.

[26] B. Goel and S. A. McKee, "A methodology for modeling dynamic and static power consumption for multicore processors," in *IEEE International Parallel and Distributed Processing Symposium*, 2016, pp. 273–282.

[27] S. Bhalachandra, A. Porterfield, and J. F. Prins, "Using dynamic duty cycle modulation to improve energy efficiency in high performance computing," in *IEEE International Parallel and Distributed Processing Symposium Workshop (IPDPSW)*, 2015, pp. 911–918.

[28] P. Petoumenos, L. Mukhanov, Z. Wang, H. Leather, and D. S. Nikolopoulos, "Power capping: What works, what does not," in *IEEE Parallel and Distributed Systems (ICPADS)*, 2015, pp. 525–534.

[29] T. Patki, D. K. Lowenthal, B. Rountree, M. Schulz, and B. R. De Supinski, "Exploring hardware overprovisioning in power-constrained, high performance computing," in *Proceedings of the 27th international ACM conference on International Conference on Supercomputing*, 2013, pp. 173–182.

[30] O. Sarood, A. Langer, A. Gupta, and L. Kale, "Maximizing throughput of overprovisioned HPC data centers under a strict power budget," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2014.

[31] D. A. Ellsworth, A. D. Malony, B. Rountree, and M. Schulz, "POW: System-wide dynamic reallocation of limited power in HPC," in *Intl. Symposium on High-Performance Parallel and Distributed Computing*, 2015.

[32] B. Wang, D. Schmidl, C. Terboven, and M. S. Müller, "Dynamic application-aware power capping," in *Proceedings of International Workshop on Energy Efficient Supercomputing*, 2017.

[33] A. Marathe, P. E. Bailey, D. K. Lowenthal, B. Rountree, M. Schulz, and B. R. de Supinski, "A run-time system for power-constrained HPC applications," 2015.

[34] A. Porterfield, R. Fowler, S. Bhalachandra, B. Rountree, D. Deb, and R. Lewis, "Application runtime variability and power optimization for exascale computers," in *Proceedings of International Workshop on Runtime and Operating Systems for Supercomputers*, 2015.