A Power Management Framework with Simple DSL for Automatic Power-Performance Optimization on Power-Constrained HPC Systems

Yasutaka Wada^{1(✉)}, Yuan He², Thang Cao³, and Masaaki Kondo³

¹ Meisei University, Tokyo, Japan
yasutaka.wada@meisei-u.ac.jp

² Shenyang University of Technology, Shenyang, Liaoning, China
heyuan@sut.edu.cn

³ The University of Tokyo, Tokyo, Japan
{cao,kondo}@hal.ipc.i.u-tokyo.ac.jp

Abstract. To design exascale HPC systems, power limitation is one of the most crucial and unavoidable issues; and it is also necessary to optimize the power-performance of user applications while keeping the power consumption of the HPC system below a given power budget. For this kind of power-performance optimization for HPC applications, it is indispensable to have enough information and good understanding about both the system specifications (what kind of hardware resources are included in the system, which component can be used as a “power-knob”, how to control the power-knob, etc.) and user applications (which part of the application is CPU-intensive, memory-intensive, and so on). Because this situation forces both the users and administrators of power-constrained HPC systems pay much effort and cost, it has been highly demanded to realize a simple framework to automate a power-performance optimization process, and to provide a simple user interface to the framework. To tackle these concerns, we propose and implement a versatile framework to help carry out power management and performance optimization on power-constrained HPC systems. In this framework, we also propose a simple DSL as an interface to utilize the framework. We believe this is a key to effectively utilize HPC systems under the limited power budget.

Keywords: HPC · Performance · Power · Optimization

1 Introduction

The need for high performance computing (HPC) in modern society never recedes as more and more HPC applications are highly involved in every aspect of our daily life. To achieve exascale performance, there are many technical challenges waiting to be addressed, ranging from the underlying device technology to exploiting parallelism in application codes. Numerous reports including Exascale

Study from DARPA [2] and Top Ten Exascale Research Challenges from DOE [12] have identified power consumption as one of the major constraints in scaling the performance of HPC systems. In order to bridge the gap between required and current power/energy efficiency, one of the most important research issues is developing a power management framework which allows power to be more efficiently consumed and distributed.

Power management is a complicated process involving the collection and analysis of statistics from both hardware and software, power allocation and control by available power-knobs, code instrumentation/optimization, and so on. For large scale HPC systems, it would be more complex since handling these tasks at scale is not easy. So far, these tasks are mostly carried out in a discrete and hand-tuned way for a specific hardware or software component. This fact causes several problems and limitations.

First, the lack of cooperation/automation makes power management very difficult and time-consuming. It is desirable that power management process is able to be carried out under a common convention with little effort from users and system administrators. Second, though there are many existing tools to be potentially used for power optimization, each of them is usually designed for different purpose. For example, PDT and TAU are used for application analysis, RAPL is dedicated to power monitoring and capping while cpufreq targets mainly for performance/power tuning [10, 18, 23]. It is necessary to have a holistic way to use them together. Third, different HPC systems have different capabilities for power management, resulting in system or hardware-specific ad-hoc solutions. Of course, these are not portable. Therefore, it is crucial to provide a generalized interface for portability and extensibility. Fourth, power management sometimes needs a collaborative effort from both users and administrators. Many existing solutions fail to draw a clear border between them.

To address these problems, some of power management related tools and APIs such as GeoPM [5] and PowerAPI [8] have been developed. These tools/APIs are user-centric and need hand-tuning efforts for efficient power management. Hence, we design and implement a versatile power management framework targeting at power constrained large scale HPC systems. We try to provide a standard utility to people of different roles when managing/using such systems. Through this framework, we provide an extensible hardware/software interface between existing/future power-knobs and related tools/APIs so that system administrators can help supercomputer users make full use of valuable power budget. Since the framework defines clear roles for people participating and software components, power management and control can be carried out securely. The framework contains a very simple domain-specific language (DSL) which serves as a front-end to many other utilities and tools. This enables users to create, automate, port and re-use power management solutions.

This paper makes the following contributions:

- We design and implement a versatile framework to meet the necessity of performance optimization and power management for power-constrained large scale HPC systems.

- By showing three case studies, we demonstrate how this framework works and how it is used to carry out certain power management tasks under different power control strategies.
- We also prove the effectiveness of this framework through these three case studies.

The rest of this paper is organized as follows. Section 2 covers the background and related work, while Sect. 3 presents details of the power management framework and its components. Section 4 is about the DSL. In Sect. 5, we demonstrate this framework through a few case studies and make discussions on the results. Finally, Sect. 6 concludes this paper.

2 Related Work

To make full use of a given power budget on power-constrained HPC systems, many aspects such as system configurations and application characteristics are required to be considered when applying power-performance optimization for applications. This section covers related research efforts for some of these aspects.

2.1 Power-Performance Optimization

Power-performance optimization methodologies typically aim at maximizing application performance while satisfying given constraints (power budget, energy consumption, application execution deadline, etc.). Most of them are mainly based on “power-shifting” among hardware components [9, 13, 19] or among applications/jobs [3, 17, 20, 24, 26]. Usually we have a large number of jobs running on a HPC system, and hence optimizing power-performance in both intra- and inter-application cases is important.

Intra-application Optimization. Some power-performance optimization methods focus on the optimization of an application by accommodating power budgets from one hardware component to another.

For example, Miwa et al. proposed power-shifting method to shift the power budget from network resources among nodes to CPUs in order to utilize unused power created when network equipment with the Energy Efficient Ethernet Standard [7] are deployed for the interconnections among nodes [13]. Gholkar et al. proposed a technique to optimize the number of processors to be used in a job under given power budgets [6]. Inadomi et al. proposed power-performance optimization method which considers manufacturing variations in a large system [9, 19].

Though these power-performance optimization methods succeeded to improve the execution performance of applications, each method employs its own optimization process and it is required to develop a unified way or framework to apply the optimizations.

Inter-application Optimization. In most HPC systems, users submit a large number of jobs, and the system scheduler allocates some of the jobs considering the number of nodes needed by each job, estimated execution time, and so on. In addition to the intra-application power-performance optimization, it is important to optimally allocate power budgets among running jobs.

Cao et al. developed a demand-aware power management approach [24]. With this approach, job scheduler allocates power budget and hardware resources to run jobs considering their demands and the total system power consumption. They also developed an approach for cooling-aware job scheduling and node allocations [3]. The aim of this approach is to shift power from cooling facilities to compute nodes as much as possible by keeping low inlet temperature. Sakamoto et al. developed extensible power-aware resource management based on slurm resource manager [20]. By using this resource management framework, job scheduler can correct and utilize the power consumption data of each application and node. Patki et al. proposed a resource management system which makes it possible to apply back-fill job scheduling considering system power utilization [17]. Chasapis et al. proposed a runtime optimization method which changes concurrency levels and socket assignment considering manufacturing variability of chips and the relationships among chips in NUMA nodes [4]. Wallace et al. proposed “data-driven” job scheduling strategy [26] which observes the power profile of each job and use it at runtime to decide power budget distribution among jobs running on the system which has limited power budget.

2.2 Interfaces to Control Power-Knobs

To realize power-performance optimization, user applications need to access and control power-knobs and obtain power consumption values for various hardware. These power-knobs may come from different vendors so we need a unified interface to access them.

Recent Intel processors are equipped with RAPL interface to apply power-capping to CPU and DRAM [10]. RAPL allows us to monitor and restrict the power consumption of both CPU and DRAM. Also, NVML [15] provides the API to manage both power consumption and operating frequency of NVIDIA’s GPUs. PAPI (Performance API) is now developed with functionalities to access NVML and RAPL interfaces [27]. With these functionalities, user applications are allowed to have access to these interfaces in the same way as to other hardware performance counters if the system administrator gives the permission to them. Though PAPI supports various processor architectures, it mainly provides us interfaces to monitor hardware counter information and is not for controlling the hardware. Power API has also been developed to provide standardized interfaces to measure/control power on HPC systems [8]. It provides us unified interfaces and abstraction layers to users, system administrators, applications, and so on.

However, users are still required to insert API calls into the application source code to realize power-performance optimized applications. In addition, each system has its own power-performance characteristics, and this makes the

optimization process much harder and requires users to spend large cost and effort. To alleviate the difficulties, it is desirable to develop a simple and easy-to-use interface to control and monitor power-knobs. This simple interface should play an important function for the unified framework.

2.3 Performance Profiling and Analysis Tools

Not only optimizing applications, but also gathering performance data of applications is essential. Detailed analysis information of a user application is required to apply power-performance optimization more effectively.

Until now, many performance analysis tools have been developed. TAU [21] is one of such performance analysis tools, with which one can collect application profiling information and visualize the data with it. Most of these tools are basically developed to give us a simple and easy way to collect application profiling data, and recently they can help collect power consumption data together with performance data by using PAPI and similar libraries. SPECpower [11] consists of a set of benchmark programs and provides instructions to measure the performance and power consumption of computer systems. SPECpower can be used to collect detailed data for comparing different systems.

However, it is difficult to utilize such information for power-performance optimization as in practice, many kinds of user applications run on many different systems. In addition, most of the performance analysis tools and methodologies assume that the user optimizes their applications manually according to the information obtained through such tools.

2.4 Domain Specific Language to Describe System Performance and Configuration

To make it easy to develop power-performance optimized applications, using a DSL is already considered. ASPEN [22] is a DSL to describe both system architecture and application characteristics in order to model the performance of an application on a system. ASPEN is good at estimating performance of the system described with it, but the power consumption is just an estimation based on given parameters. To realize power-performance optimization on real systems, it is required to know the actual power profile on target systems.

2.5 Towards a Versatile Power Management Framework

Most of the works above, related to power-performance optimization/management on power-constrained HPC systems, could help us carry out good power management. However, we still need to pay much cost and effort to apply them to many kinds of user applications, and it is difficult to make all users know well about the system they want to use, in addition to their application characteristics. Furthermore, if the system is replaced, they are forced to optimize their applications again while getting knowledge about the new system.

In this paper, we aim to provide a simple and easy way to realize power-performance managed/optimized application by using a versatile power management framework which has access to all kinds of information from the systems, users, and applications. This framework applies power-performance optimization to the application automatically, and can reduce users' burden drastically.

3 The Proposed Power Management Framework

The main objective of the framework proposed in this paper is to make the power management and power-performance optimizations processes more facilitating and flexible for both users and system administrators. In this framework, we assume a standard HPC system with its hierarchical structure as shown in Fig. 1. The target system consists of multiple compute nodes, interconnection network, and the storage subsystem. Each node consists of multiple processors, DRAM modules, and accelerators like GPUs. Each processor has multiple cores. We assume some of the hardware components have power-knobs to control power consumption of the executing programs, but their availability to the users depends on the control permission or the operational policy specified by the system administrator.

3.1 Overview and Power Management Workflow Control

To optimize power-performance efficiency and to manage power consumption of HPC applications to be executed, we have to take care of many things including: (1) what kinds of hardware components the system has and how much power is consumed in them, (2) what kinds of power-knobs are available and how to control them, (3) how the applications behave at runtime, and (4) what is the relationship between performance and power consumption of the application.

Fig. 1. The outline of a target HPC system

Based on these information, (5) we have to design a power-performance optimization algorithm. One of the most burdensome tasks is (6) to assemble and utilize existing tool-sets for collecting the necessary information and actually controlling power-knobs.

It requires the user to pay much cost and time to consider these tasks. Our framework is designed to provide or to support the following functionalities which help users and administrators carry out power management/control effectively without taking care of the above mentioned issues:

- Analyzing source code and applying automatic instrumentation
- Measuring and controlling application power consumption and performance
- Optimizing an application under given power budget
- Specifying and defining the target machine specification
- Calibrating hardware power consumption of the system

The outline of the framework is presented in Fig. 2. One of the benefit of the framework is the fact that workflow of power-performance optimization and control can be specified in higher abstraction level. Details of how to use libraries for controlling power-knobs, how to profile and analyze the application code, and how to instrument power management pragmas in the code or the job submission script are hidden from users. Moreover, the framework provides high modularity and flexibility so that libraries or tool-set used in the framework and also power optimization algorithms are customizable.

In order to provide customizability and flexibility, we developed a simple DSL as the front-end for these supported tools and for selecting the power optimization algorithm. Note that in current version of the framework, we support RAPL/cpufreq utils for accessing power-knobs [9, 19] and TAU/PDT for code

Fig. 2. The overview of the power-performance optimization framework

instrumentation and profiling. However, these are extensible and other tool-sets are easily supported.

This framework requires only two sets of inputs, the DSL code and the user application source code. Based on them, our framework offers a semi-automatic way of power-performance optimizations. Meanwhile, the administrators and users can be free from the effort to understand the inside of the optimization workflow. Once the DSL source code is prepared, the proposed framework provides an easy way to realize optimized execution of user applications.

Note that our framework supports two types of users. One is simply supercomputer “user” and the other is “administrator”. An administrator is able to specify machine configurations, enable/disable power-knobs and calibrate the hardware while a user is not allowed to do so. Switching between them is carried out with the DSL.

3.2 Application Instrumentation for Profiling and Optimization

To realize power-performance profiling and optimized execution of a user application, the application is required to be instrumented with API calls to get profiling data and to control power-knobs. In current version of our workflow, it is assumed that PDT based instrumentation tool [9, 19, 25] is used for automatic instrumentation. For example, the user should instruct the instrumentation tool to insert API calls before/after each function call, parallel loop, and so on to control the power-knobs. This process can also be carried out with the DSL.

Figure 3 shows an example of the automatic instrumentation by the framework. As shown on the left in Fig. 3, our automatic instrumentation tool assumes that its input is a source code written with MPI. The instrumentation tool detects the entry/exit points of each function in the source code and inserts API calls. The tool also inserts an initialization API call just after “MPI.Init()”. As the result, as shown on the right in Fig. 3, a source code with API calls (“PomPP_Init()” for initialization, and “PomPP_Start_Section()” and “PomPP_Stop_Section()” for power-knob control) is generated.

Beside the framework, the user can insert the API calls into the application code by themselves. However, it would be a troublesome task since the user needs a sanity check for start-stop relationship of all the API calls throughout the whole source code. If the control flow of the application code is complex, this is not easy. For example, in Fig. 3, “func1” has two return statements, and API calls indicating the exit point of the section have to be inserted for both of them.

For the automatic code instrumentation, we include TAU based profiling tool [21] in the framework and it allows selective instrumentation. With the selective instrumentation capability, users can specify which functions should be instrumented for power-knob control based on its execution time and so on.

As shown in Fig. 2, we assume to use the same execution binary for both profiling and optimization run to reduce users’ labor. To realize this, the library is developed to enable both profiling and power-knob control with the same API call [9, 19]. The library can change its behavior based on the user setting via environment variables.

Fig. 3. An example of the automatic instrumentation for both profiling and power-knob control

3.3 Power Control and Application Optimization

In the current implementation of the framework, we assume to decide power-capping and power distribution in advance statically. For the optimization, the user is asked to run at least two scripts generated by the DSL described in Sect. 4.

The first one is the script to generate power-performance profiling data for the application. This script runs the instrumented application several times and generates power consumption profiling data of the available power-knobs with several settings of them.

The second one is the script for optimized application run. In this script, power-knob settings are decided in advance based on the information given by the hardware calibration data, the profiling data of the target application, and other given constraints (power budget, allowed slowdown in the execution time, and so on). This decision is written to a file, and is referred to find out when and how the power-knobs should be controlled while running the application.

In our current implementation, it is assumed that the script or program, which is used to make a decision for the optimized power capping values, is prepared by the system administrator in advance, because the administrator is responsible to decide which power-knobs to be opened for user applications and what kind of power-performance models are desirable.

3.4 Machine Specification and Setting

A main feature of this framework is to help the system administrators set/modify the configurations of various HPC systems. Through the DSL source, the administrator can provide the system configuration and available power-knobs to the framework. Users of the system should have the permission to access the power-knobs as if they are allowed by the administrator.

3.5 Hardware Calibration

Precise power-performance control is required to realize overprovisioned HPC system because power budget is usually strictly constrained for safe operation of the system. To realize the precise control, actual power consumption information of each component in the system is necessary. In addition, we may need the information on how different the power consumption of each component is. As Inadomi et al. mentioned, because of manufacturing variations, each component in the system has its own characteristic even when we compare the same products [9]. Hardware calibration is very useful in a large scale system as even components with identical performance specifications actually have different power characteristics. In most cases, such hardware configuration and calibration processes are only required once per system right after the system is installed.

Therefore, the proposed framework provides the scripts for the hardware calibration based on the information given by the administrators. The scripts run some microbenchmarks and collect the power-performance relationship information for each component. With this information and the profiling data of user applications, our framework decides how much power budgets should be allocated for each power-knob.

4 DSL to Control Power Management Workflow

As a front-end to our framework, we have developed a simple DSL to provide a uniform gateway to tools and utilities in the framework. It helps to create power management algorithms and processes. The DSL interpreter is developed based on ANTLR v4 [1, 16] with very simple semantics.

Using this DSL leads to several advantages. First, this simple DSL provides a unified way to access to various functionalities supported by this framework. Second, system configurations, optimization processes and algorithms are composed in this DSL such that they can be easily reused and extended. Third, given the code written in this DSL, automation is possible which dramatically improves productivity.

4.1 Semantics of the DSL

Source code written in our DSL is composed of a basic element which is called the “statement”. Each statement has a command, which is used to specify an action.

For example, Listing 1, 2 and 3 illustrate statements manipulating objects defined in this DSL. Listing 1 shows how system configuration is set and Listing 2 shows how to use an application as the microbenchmark to calibrate the hardware. Listing 3 is about how a job is submitted with a socket power cap of 70 W.

Listing 1. DSL Code Snippets for System Configuration

```

1 CREATE MACHINE M
2 ADD M POMPP_NPKGS_PER_NODE 2
3 ADD M POMPP_NCORES_PER_PKG 12
4 ADD M POMPP_TOTAL_NODES 965
5 ADD M POMPP_MAX_FREQ 16
6 ADD M POMPP_MIN_FREQ 12
7 ADD M POMPP_PKG_TDP 130.0
8 ADD M POMPP_DRAM_TDP 62.0
9 ADD M POMPP_PKG_MIN 64.0
10 ADD M POMPP_DRAM_MIN 30.0
11 ADD M POMPP_MODULE_MIN 46.0
12 SWITCH M

```

Listing 2. DSL Code Snippets for Hardware Calibration

```

1 CREATE JOB EP_C
2 ADD EP_C EXEC_PATH <absolute path to the executable>
3 ADD EP_C JOB_TYPE CALIBRATION
4 ADD EP_C PVT_PATH <absolute path to the power variation table>
5 SUBMIT EP_C

```

Listing 3. DSL Code Snippets to Submit a Job with a Specified Power Cap

```

1 CREATE JOB EP_G
2 ADD EP_G EXEC_PATH <absolute path to the executable>
3 ADD EP_G JOB_TYPE GENERAL
4 ADD EP_G MODULE_POWER 70
5 ADD EP_G CONTROL_MODE RAPL
6 SUBMIT EP_G

```

So far, commands supported by this DSL are “CREATE”, “DELETE”, “ADD”, “REMOVE”, “GET”, “SET”, “LIST”, “INSERT”, “SWITCH” and “SUBMIT”. When initializing objects, “CREATE” is used so it is followed by a type and an object name while “ADD” is used to add attribute and it is followed by an attribute and its value. “GET” and “SET” are used to retrieve and modify attributes of an object while “REMOVE” is used to remove an attribute of an object and “DELETE” is used to delete an object. In addition, “LIST” is used to list objects created; “INSERT” is used for manual instrumentation; and “SUBMIT” means to submit a job to the HPC system. Finally, “SWITCH” is used to switch between “user” and “administrator” or used to switch to different sets of machine configurations. In the scope of this DSL, an administrator is also a user, but with more commands/capabilities available. For example, an administrator can submit a job just like a user but a user is not allowed to calibrate the hardware. Therefore, when switching to the “administrator” role, a password is required to prevent an administrator from misuses of privileged commands. All commands are summarized in Table 1.

Supported types in this “DSL” include “MACHINE”, “JOB” and “MODEL”. “MACHINE” is used to represent set of system configurations, while “JOB”

Table 1. Commands in the DSL

CREATE/DELETE	Creating or deleting an object
ADD/REMOVE	Adding or removing an attribute for an object
GET/SET	Setting or getting the value of an attribute
LIST	Listing objects
INSERT	Manual instrumentation to application source
SWITCH	Switching between user/admin or switching to a different machine
SUBMIT	Submitting jobs to the system

is used to represent a job to run on the system and “MODEL” is defined as the relationship between performance and power consumption, which can be used to optimize the execution process of an application to satisfy power or performance requirements according to a mathematical model. Such models are used to generate power caps from the profiling and hardware calibration data.

In addition to commands and types above, arrays and loops are also supported in this DSL. Arrays are used to create objects as a batch and loops simply help improve the productivity of this DSL. Through all these supported features, this DSL stands between the user/system administrator and our framework and allows the framework to be accessed in a more unified manner and also realizes more complex power management tasks.

4.2 Implementation of the DSL Interpreter

The DSL interpreter is designed and implemented with ANTLR v4. During interpretation, various DSL statements are translated into shell scripts and application instrumentation for different purposes such as hardware calibration, application profiling, job submission, specifying power control in the application, interfaces to other tools and so on. This interpretation process is uniform for different systems but different hardware configurations may lead to variations in the results.

Along with any created instances of a defined type in this DSL, there is also an XML file created to store their attributes. For example, an instance of the type “job” will have an accompanying XML file which stores its attributes such as its name, path, executable, power caps and so on.

5 Case Studies and Evaluation

In this section, we provide three case studies to demonstrate some of the functionalities of our framework. All these case studies are firstly programmed with the DSL and then interpreted on a gateway node of an HPC system with its specifications shown in Table 2.

Table 2. Evaluation environment and parameters

Number of nodes	16
Processor	Intel Xeon E5-2680
Number of sockets per node	2
Number of cores per socket	8
Memory size per node	128 GB
Interconnect	Infiniband FDR
OS	Red Hat Linux Enterprise with kernel 2.6.32
Compiler	FUJITSU Software Technical Computing Suite
MPI	Open MPI 1.6.3
Applications	EP and IS (Class D) from the NPB Suite

Fig. 4. Power profiles of EP and IS

Listing 4. DSL Source for Profiling

```

1 CREATE JOB EP_P
2 ADD EP_P EXEC_PATH <absolute path to the executable>
3 ADD EP_P JOB_TYPE PROFILING
4 ADD EP_P PVT_PATH <absolute path to the power variation table>
5 ADD EP_P ITERATIONS 5
6 ADD EP_P PROFILE_NAME EP
7 SUBMIT EP_P

```

In these case studies, we employed RAPL interface [10] as the available power-knob, and considered only CPU power to be controlled through the RAPL under the assumption that DRAM power consumption has strong correlations with CPU performance and power. We used two applications (EP and IS) from the NPB benchmark suite [14] to carry out these case studies. To understand their performance and power characteristics, profiling is necessary and the results

are shown in Fig. 4 with an interval of 100 ms. The profiling processes are also specified with our DSL as in Listing 4. How power capping should be applied during the profiling process depends on the system and the power-performance model to be used, and our framework can be easily extended to follow them.

5.1 Case Study 1: Peak Power Demand as the Power Cap for Applications

The first case study shows how a maximum power demand of an application is used as the power cap for the application. This case study requires our framework to insert power-knob control API calls to the user application. Such API calls are inserted into the application source file, and help both the profiling process and capping tasks.

Listing 5. DSL Source for Case Study 1 (with Peak Power Demand)

```
1 CREATE MODEL MAX
2 ADD MAX MODEL_PATH <absolute path to the model script to find out max power>
3
4 CREATE JOB EP_MAX
5 ADD EP_MAX EXEC_PATH <absolute path to the executable>
6 ADD EP_MAX JOB_TYPE OPTIMIZATION
7 ADD EP_MAX CONTROL_MODE RAPL
8 ADD EP_MAX PROFILE_NAME EP
9 ADD EP_MAX MODEL_TO_USE MAX
10 SUBMIT EP_MAX
```

Fig. 5. Power performance optimization results under case study 1 (with peak power demand)

For this case study, first we profile a target application without any power cap to get its general power profile, and then search for its peak power consumption in the profile. After finding the peak power consumption, we then launch a production run with it as the power cap so that the application does not suffer from any performance loss under the guarantee that the power consumption will not exceed given power budget. The DSL source (for EP) for this case study is shown in Listing 5.

Figure 5 presents the power profiles of two applications under the peak power demand. As expected, there is no performance loss observed in this case study.

5.2 Case Study 2: Average Power Demand as the Power Cap for Applications

The second case study is used to prove how the average power demand of an application is obtained through profiling and used as the power cap for the application. Such power caps can help save the power and may lead to more energy-efficient runs. Saved power can be distributed to other jobs simultaneously running on the same system by the system software like a job scheduler. The DSL source (for EP) for this case study is shown in Listing 6.

In this case study, first we run the target application without any power capping to get its general power profile like Case Study 1, and then obtained average power consumption through a simple calculation. We set the power caps with this average value for a power optimized run.

Figure 6 presents the power profiles of two applications under the average power demand. For each application, there is a performance loss compared to Case Study 1, but the power consumption is much less.

Listing 6. DSL Source for Case Study 2 (with Average Power Demand)

```

1 CREATE MODEL AVERAGE
2 ADD AVERAGE MODEL_PATH <absolute path to the model script to find out the average power>
3
4 CREATE JOB EP_AVE
5 ADD EP_AVE EXEC_PATH <absolute path to the executable>
6 ADD EP_AVE JOB_TYPE OPTIMIZATION
7 ADD EP_AVE CONTROL_MODE RAPL
8 ADD EP_AVE PROFILE_NAME EP
9 ADD EP_AVE MODEL_TO_USE AVERAGE
10 SUBMIT EP_AVE

```

5.3 Case Study 3: Power Cap to Satisfy a User-Defined Deadline While Minimizing Power Consumption

The third case study is used to show how a linear performance/power model of the application is constructed through profiling and how we use this model to derive the power cap according to user's performance demand for the application.

Fig. 6. Power performance optimization results under case study 2 (with average power demand)

Listing 7. DSL Source for Case Study 3 (within a Slowdown of 2)

```

1 CREATE MODEL LINEAR
2 ADD LINEAR MODEL_PATH <absolute path to the script of a linear model between performance
  and power>
3
4 CREATE JOB EP_LINEAR
5 ADD EP_LINEAR EXEC_PATH <absolute path to the executable>
6 ADD EP_LINEAR JOB_TYPE OPTIMIZATION
7 ADD EP_LINEAR CONTROL_MODE RAPL
8 ADD EP_LINEAR PROFILE_NAME EP
9 ADD EP_LINEAR MODEL_TO_USE LINEAR
10 ADD EP_LINEAR PERFORMANCE_TARGET 0.5
11 SUBMIT EP_LINEAR

```

In addition to the power profiles shown in Fig. 4, four extra rounds of profiling are required for this case study. The first two extra rounds are launched with the peak power demand to find the shortest runtime. Then through the third extra round of profiling where we set the power caps to a very small value (10 W/Socket), we found that the minimum amount of power needed to run both applications properly is around 30 W. We then set the power cap to 30 W per socket and profile the forth extra round to find the runtime of the applications. Using these profiled data, we can construct a linear performance/power model for each application as shown in Fig. 7.

Using the models shown in Fig. 7, performance demand can be set from the users when they submit their jobs through the DSL code. For example, if the user allows the runtime to be doubled, the corresponding power caps can be found from these two models as 59 W and 34 W for EP and IS, respectively. The DSL source (for EP) for this case study is shown in Listing 7.

Fig. 7. The linear performance/power models for EP and IS

Fig. 8. Power performance optimization results under case study 3 (within a slowdown of 2)

Figure 8 presents the power profiles of the two applications under power caps obtained from the models to allow the elapsed time to be shorter than twice the runtime under the peak power demand. Obviously, these two models are not accurate enough so that both applications are slowed down for less than two times (1.20x and 1.53x, respectively). Regardless of the accuracy of such models, at least user’s performance demand is satisfied while the allocated power is dramatically cut.

6 Conclusions

We have demonstrated a versatile power management framework for power-constrained HPC systems to tackle the problem of power limitation. With this framework, HPC system administrators can easily specify and calibrate their system hardware. Meanwhile, it is also helpful for tasks such as how the user applications should be tuned to maximize the performance or to cut the power demand.

To verify the validity and usefulness of our framework, we tested it with several case studies. In these case studies, we applied power management to two selected applications and showed how a simple power model with linear relationship between the CPU performance and power consumption can be constructed and used to derive the power cap. These case studies simply proved that our framework can provide the users an easy way to apply power optimization and management to their applications.

In our future work, we plan to evaluate the proposed framework with other power and performance optimization policies/algorithms, and to improve it with more functionalities such as cooperation with system software, job schedulers and other external tools to enrich its functionalities.

Acknowledgment. This work is supported by the Japan Science and Technology Agency (JST) CREST program for the research project named “Power Management Framework for Post-Petascale Supercomputers”. We are also grateful to the Research Institute for Information Technology of Kyushu University for providing us the resources and to all project members for their valuable comments and cooperation.

References

1. ANTLR. <http://www.antlr.org/>
2. Bergman, K., et al.: Exascale computing study: Technology challenges in achieving exascale systems (2008)
3. Cao, T., Huang, W., He, Y., Kondo, M.: Cooling-aware job scheduling and node allocation for overprovisioned HPC systems. In: Proceedings of the 31st IEEE International Parallel and Distributed Processing Symposium (IPDPS 2017), pp. 728–737, May 2017
4. Chasapis, D., Casas, M., Moretó, M., Schulz, M., Ayguadé, E., Labarta, J., Valero, M.: Runtime-guided mitigation of manufacturing variability in power-constrained multi-socket NUMA nodes. In: Proceedings of the 2016 International Conference on Supercomputing (ICS 2016), pp. 5:1–5:12, June 2016
5. GeoPM. <https://github.com/geopm>
6. Gholkar, N., Mueller, F., Rountree, B.: Power tuning HPC jobs on power-constrained systems. In: Proceedings of the 2016 International Conference on Parallel Architectures and Compilation (PACT 2016), pp. 179–191, September 2016
7. IEEE Std 802.3az-2010 (2010). <https://standards.ieee.org/findstds/standard/802.3az-2010.html>
8. Laros III, J.H., DeBonis, D., Grant, R., Kelly, S.M., Levenhagen, M., Olivier, S., Pedretti, K.: High performance computing - power application programming interface specification version 1.3, May 2016

9. Inadomi, Y., Patki, T., Inoue, K., Aoyagi, M., Rountree, B., Schulz, M., Lowenthal, D., Wada, Y., Fukazawa, K., Ueda, M., Kondo, M., Miyoshi, I.: Analyzing and mitigating the impact of manufacturing variability in power-constrained supercomputing. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC15), pp. 78:1–78:12, November 2015
10. Intel Corporation: Intel® 64 and IA-32 architectures software developer’s manual, September 2016
11. Lange, K.D.: Identifying shades of green: the SPECpower benchmarks. *Computer* **42**(3), 95–97 March, 2009
12. Lucas, R., et al.: Top ten exascale research challenges (2014)
13. Miwa, S., Nakamura, H.: Profile-based power shifting in interconnection networks with on/off links. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC15), pp. 37:1–37:11, November 2015
14. NAS parallel benchmarks 3.3. <http://www.nas.nasa.gov/>
15. NVIDIA Corporation: NVML reference manual (2015)
16. Parr, T.: The Definitive ANTLR 4 Reference, 2nd edn. Pragmatic Bookshelf, Dallas (2013)
17. Patki, T., Lowenthal, D.K., Sasidharan, A., Maiterth, M., Rountree, B.L., Schulz, M., de Supinski, B.R.: Practical resource management in power-constrained, high performance computing. In: Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing, pp. 121–132, June 2015
18. PDT. <https://www.cs.uoregon.edu/research/pdt/home.php>
19. PomPP Library and Tools. <https://github.com/pompp/pompp.tools>
20. Sakamoto, R., Cao, T., Kondo, M., Inoue, K., Ueda, M., Patki, T., Ellsworth, D.A., Rountree, B., Schulz, M.: Production hardware overprovisioning: real-world performance optimization using an extensible power-aware resource management framework. In: Proceedings of the 31st IEEE International Parallel and Distributed Processing Symposium (IPDPS 2017), pp. 957–966, May 2017
21. Shende, S.S., Malony, A.D.: The TAU parallel performance system. *Int. J. High Perform. Comput. Appl.* **20**(2), 287–331 (2006)
22. Spafford, K.L., Vetter, J.S.: Automated design space exploration with aspen. *Sci. Program.* 7:1–7:10 (2015)
23. TAU. <https://www.cs.uoregon.edu/research/tau/home.php>
24. Cao, T., Thang, C., He, Y., Kondo, M.: Demand-aware power management for power-constrained HPC systems. In: Proceedings of 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2016), pp. 21–31, May 2016
25. Wada, Y., He, Y., Thang, C., Kondo, M.: The PomPP framework: from simple DSL to sophisticated power management for HPC systems. In: HPC Asia 2018 Poster Session, January 2018
26. Wallace, S., Yang, X., Vishwanath, V., Allcock, W.E., Coghlan, S., Papka, M.E., Lan, Z.: A data driven scheduling approach for power management on HPC systems. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC16), pp. 56:1–56:11, November 2016
27. Weaver, V.M., Johnson, M., Kasichayanula, K., Ralph, J., Luszczek, P., Terpstra, D., Moore, S.: Measuring energy and power with PAPI. In: Proceedings of the 41st International Conference on Parallel Processing Workshops (ICPPW-2012), pp. 262–268, September 2012

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.