The PomPP Framework: From Simple DSL to Sophisticated Power Management for HPC Systems

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

Yuan He
The University of Tokyo
Tokyo, Japan
he@hal.ipc.i.u-tokyo.ac.jp

Thang Cao
The University of Tokyo
Tokyo, Japan
cao@hal.ipc.i.u-tokyo.ac.jp

Masaaki Kondo
The University of Tokyo
Tokyo, Japan
kondo@hal.ipc.i.u-tokyo.ac.jp

ABSTRACT

Effectively utilizing the limited power budget is one of the most important issues when realizing HPC systems beyond petascale performance and this remains a difficult task from every possible angle. On the end user's perspective, optimizing the performance and power consumption of HPC applications requires a good understanding of the system specifications in addition to the characteristics of applications. For system administrators, they are required to provide the system specifications and define the interfaces to monitor and tune certain components while keeping in mind both stability and security of the system. In addition, automation of such tasks is also very critical in terms of productivity. To tackle these concerns, we propose and implement a simple framework to carry out power management and performance optimization of HPC systems.

KEYWORDS

HPC, Performance, Power, Optimization

1 INTRODUCTION

Numerous reports including the Exascale Study from DARPA [2] and the Top Ten Exascale Research Challenges from DOE [7] have identified power consumption as the major constraint to further scale the performance of HPC systems under existing designs. In order to bridge the power/energy efficiency gap, power management is one of the most important approaches, allowing power to be more efficiently consumed and distributed.

Power management is a complex process involving the collection and analysis of statistics from both hardware and software, power allocation with available performance and power-knobs, code instrumentation/optimization and so on. For large scale systems, it gets more complex since handling these tasks under the pressure of scalability and size is not easy. So far, for large scale systems, these tasks are mostly carried out in a discrete and manual manner for specific hardware and software. Therefore, many problems and limitations arise:

- how to apply power management/optimization method in a short time
- how to utilize existing power management tools
- how to adjust the power optimization for various systems

To address some of these problems, efforts like GeoPM and PowerAPI are developed [3, 4]. However, GeoPM is more user-centric while PowerAPI is too abstract that it still requires further effort for production use. Hence, we design and implement a versatile power management framework targeting at power constrained large scale HPC systems. Our objective is to provide a standard utility to people of different roles when managing/using such systems. Through this framework, we provide the following contributions:

- an extensible hardware/software interface to existing/future power-knobs and tools
- security within the system can be guaranteed with clearly defined roles for users and administrators
- a front-end to many utilities and tools in the framework is provided through a simple domain-specific language (DSL) included in this framework

These enable users to create, automate, port and re-use power management solutions.

2 THE PROPOSED POWER MANAGEMENT FRAMEWORK

In the proposed framework, we assume an HPC system with its hierarchical structure. As a total system, it consists of multiple computing nodes, interconnect connecting the nodes, and the storage subsystem shared among nodes. Each node consists of multiple processors, DRAM modules, and accelerators like GPUs. Each processor has one ore more cores. Each component in this system can be a power-knob, but its availability to the users depends on the capability of the hardware and control permission specified by the system administrator.

2.1 Framework Overview

To apply power-performance optimization and power management to an HPC application, our framework has been designed to support the following functionalities:

- Specifying the target machine configurations
- Calibrating hardware power consumption
- Measuring and controlling application power consumption
- Analyzing and instrumenting source code
- Optimizing application under specified power budget and/or performance-power models

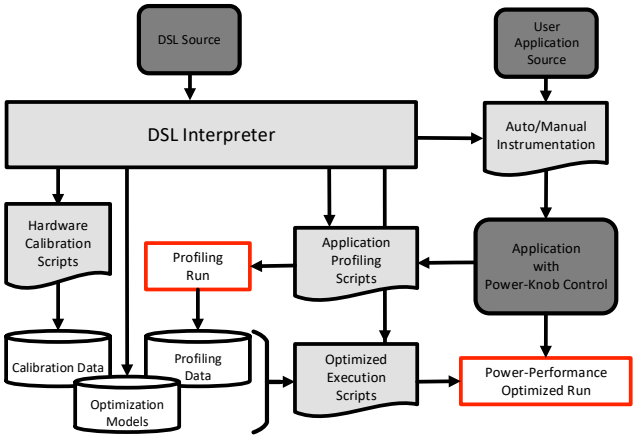

Figure 1: The Overview of the Performance and Power Optimization Framework

These functionalities are supported through certain performance and power tuning libraries to access power-knobs such as RAPL[6] and cpufrequtils, and instrumentation tools implemented based on TAU and PDT [5, 10–12]. In order to make the framework with these libraries/tools more user-friendly and productive, we developed a simple DSL as the front-end of this framework.

Figure 1 shows the overview of the proposed framework. The framework requires two sets of input, the DSL source and the user application source code. Based on them, our framework provides power-performance optimizations for the application. Meanwhile, the administrators and users can be free from the effort to understand the details of optimization workflow. Once the DSL source codes are prepared, the proposed toolchain provides easy way to realize optimized execution of user applications.

2.2 Machine Specification and Setting

The first 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.

2.3 Hardware Calibration

Precise power-performance control is required to realize overprovisioned HPC system because power budget is usually tight constraint for safe operation of the system. Also, As Inadomi et al. mentioned, because of manufacturing variations, each component in the system has its own characteristics even when we compare the same products [5].

Therefore, the proposed framework provides scripts for hardware calibration based on 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 the user applications, our framework decides how much power budget should be allocated with each power-knob.

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
```

2.4 Applications Instrumentation and Power-Performance Optimization

To realize power-performance profiling and optimized execution of a user application, the application is required to be instrumented with API call to get profiling data and to control power-knobs. In this toolchain, it is assumed that PDT based instrumentation tool [5, 10–12] is used for automatic instrumentation.

In the current implementation of the framework, we assume to statically decide power-capping and power distribution in advance. For this optimization, the user is asked to run at least two scripts generated by the DSL. The first one is the script to generate power-performance profiling data for the application, and the second one is the script for optimized application run. Power-performance profiling data generated by the first one is used to generate power-knob settings for optimized execution under the given power budget. It is also assumed that these scripts or program are prepared by the system administrator in advance to decide which power-knobs to be opened for user applications and what kind of power-performance models are desirable.

2.5 Simple DSL as a Front-End

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, reuse, and extend power management/optimization algorithms and processes. Then, once the code written in this DSL is given, automation is possible which dramatically improves productivity. interpreter is developed based on ANTLR v4 [1, 9] with very simple semantics.

Source code written in our DSL is composed of a basic element which is called the “statement”. A statement has two or three parts including a command, a type (if an object/attribute is not defined yet) and an object/attribute name. For example, Listing 1, Listing 2 and Listing 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 microbench to calibrate the hardware. Listing 3 is about how a regular job is submitted with a module power cap of 70W.

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
```

Table 1: Evaluation Environment

Number of Nodes	16
Processor	Intel Xeon E5-2680 (8 cores) 2 sockets per node
Memory Size	128GB per node
Interconnect	Infiniband FDR
OS	RHEL with kernel 2.6.32
Compiler	FUJITSU Software Technical Computing Suite
MPI	Open MPI 1.6.3

3 EVALUATION WITH A CASE STUDY

In this section, we provide a case study to demonstrate some of the functionalities of our framework.

3.1 Evaluation Settings

The case study is programmed with the DSL at first and then interpreted on a gateway node of an HPC system with its specifications shown in Table 1.

In this evaluation, we employed RAPL interface as the available power-knob, and considered only CPU power to be controlled through RAPL under the assumption that DRAM power consumption has strong correlations with CPU performance and power. We used two applications (EP and IS with Class D dataset) from the NPB benchmark suite [8] to carry out these case studies. To understand their performance and power characteristics, profiling is necessary and the results are shown in Figure 2 with an interval of 100ms. The profiling processes are also specified with our DSL.

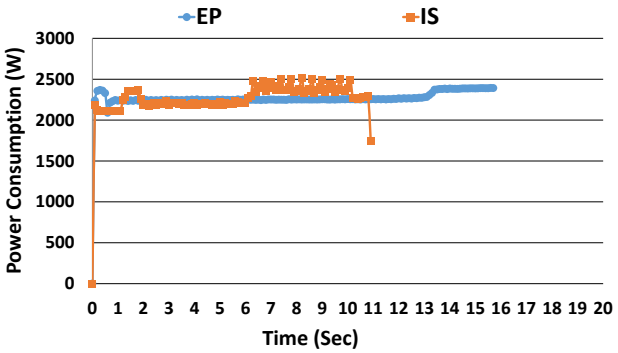

Figure 2: Power Profiles of EP and IS

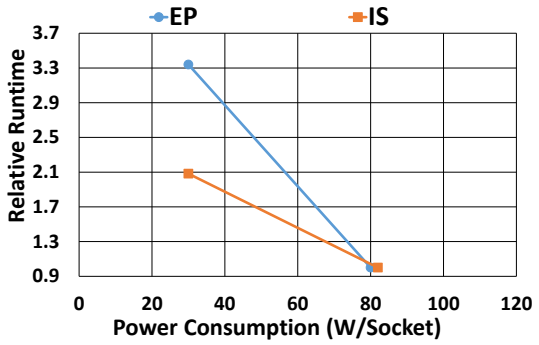

Figure 3: The Linear Power-Performance Models for EP and IS

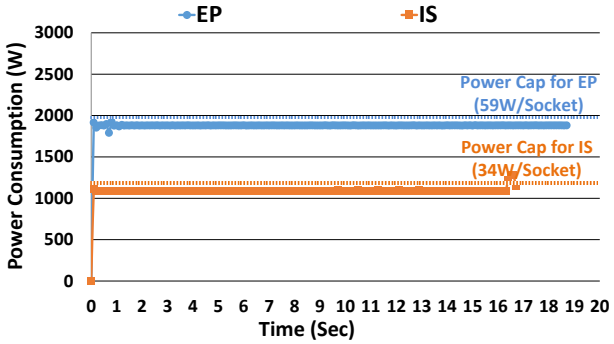

Figure 4: Power Performance Optimization Results (within a Slowdown of 2)

3.2 Case Study: Power Cap to Satisfy A User-Defined Deadline for the Application

In this case study, 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. In addition to the power profiles shown in Figure 2, four extra rounds of profiling are required for this case study to construct the linear performance/power model for each application shown in Figure 3.

Using the models shown in Figure 3, 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 59W and 34W for EP and IS, respectively.

Figure 4 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. Though the slowdown for each application is less than two (1.20x and 1.53x, respectively) because of the model accuracy, user's performance demand is satisfied and allocated power are dramatically cut.

4 CONCLUSIONS

We have demonstrated a 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.

In the case study, we apply power management to two applications and show 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. 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 improve it with more functionalities such as cooperation with system software, job schedulers and other external tools to enrich its functionalities.

ACKNOWLEDGMENTS

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.

REFERENCES

[1] ANTLR. [n. d.]. http://www.antlr.org/. ([n. d.]).
[2] Keren Bergman et al. 2008. ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems. (2008).
[3] GeoPM. [n. d.]. https://github.com/geopm. ([n. d.]).
[4] James H. Laros III, David DeBonis, Ryan Grant, Suzanne M. Kelly, Michael Levenhagen, Stephen Olivier, and Kevin Pedretti. 2016. High Performance Computing - Power Application Programming Interface Specification Version 1.3. (May 2016).
[5] Yuichi Inadomi, Tapasya Patki, Koji Inoue, Mutsumi Aoyagi, Barry Rountree, Martin Schulz, David K. Lowenthal, Yasutaka Wada, Keiichiro Fukazawa, Masatsugu Ueda, Masaaki Kondo, and Ikuo Miyoshi. 2015. 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)*. 78:1–78:12.
[6] Intel Corporation. 2016. Intel® 64 and IA-32 Architectures Software Developer's Manual. (September 2016).
[7] Robert Lucas et al. 2014. Top Ten Exascale Research Challenges. (2014).
[8] NAS parallel benchmarks 3.3. [n. d.]. http://www.nas.nasa.gov/. ([n. d.]).
[9] Terence Parr. 2013. *The Definitive ANTLR 4 Reference (Second Edition)*. Pragmatic Bookshelf.
[10] PDT. [n. d.]. https://www.cs.uoregon.edu/research/pdt/home.php. ([n. d.]).
[11] PomPP Library and Tools. [n. d.]. https://github.com/pompp/pompp_tools. ([n. d.]).
[12] TAU. [n. d.]. https://www.cs.uoregon.edu/research/tau/home.php. ([n. d.]).