Parallel Computing: Technology Trends
I. Foster et al. (Eds.)

doi:10.3233/APC200070

Dynamic Runtime and Energy Optimization for Power-Capped HPC Applications

Bo WANG [a], Christian TERBOVEN [a] and Matthias MÜLLER [a]
[a] *IT Center at RWTH Aachen University, Germany*

Abstract. Future large-scale high-performance computing clusters will face a power wall where the peak power draw of these clusters exceeds the maximal power-supplying capability of the surrounding infrastructure. To use the limited power budget efficiently, we developed a dynamic strategy to tackle execution time imbalance issues through power shifting and frequency limitation. By applying this strategy to NPB OpenMP benchmarks, we succeed in a continuous enforcement of power draw under a specified power cap. At the same time, execution time is reduced by up to 12.8% and the energy to solution is reduced by up to 12.3%, compared to a native power strategy.

Keywords. power capping, performance optimization, energy reduction, power shifting, core frequency limitation

1. Introduction

Large-scale high-performance computing (HPC) clusters face a power wall where their peak power draw exceeds the power supplying capacity of the surrounding infrastructure [1]. The power draw of these clusters has to be limited in order to avoid hardware damage. Under this constraint, utilizing the power budget efficiently and minimizing the execution time of running jobs are required in order to improve the clusters' job throughput.

Execution of a parallel job suffers frequently from imbalance among parallel tasks. In this work, a parallel task can be an MPI process or an OpenMP thread. Each task may reach a global synchronization call, like barrier, with distinct delays. The imbalance can be caused by inherited load imbalance of the job, but can also be caused by power capping at runtime.

Due to variations at manufacturing, processors have distinct power efficiencies, defined as the number of watts needed to execute certain operations. Enforcing a power cap on these processors causes distinct performance loss, like floating-point operation rate [FLOP/s]. The loss happens because of the different amounts of power need be cut to remain under the power cap[2]. Therefore, power-capping causes an additional runtime execution imbalance.

Because of diverse factors causing an execution imbalance, it is difficult to analyze and handle the imbalance issue before the execution. In this work, we developed a dynamic power and frequency management (DPF) method to detect, analyze and handle

runtime imbalances issue. Applying DPF to NPB OpenMP benchmarks, we achieved up to 12.8% performance improvement and saved up to 12.3% energy compared to a native strategy.

The remainder of the paper is structured as follows: in Section 2, we describe related work briefly. In section 3, we illustrate and analyze execution imbalance issues in detail. Based upon the imbalance observations, we introduce the DPF method to tackle these issues in section 4. In the subsequent Section, we evaluate DPF with NPB benchmarks in comparison with a native management strategy. In the last section, we conclude this work.

2. Related work

Several large scale clusters are facing a power wall issue and several works had been contributed to investigate and optimize performance of power-capped applications and clusters[3][2][4][5][6][7].

Routree et al. [2] observed that power capping causes performance imbalance among processors of the same model since processors have distinct power efficiencies. This kind of imbalance causes execution time imbalance of parallel tasks.

The authors of [6] and [7] introduced a static power budgeting method to handle the imbalance issue. They measured and documented the processors' power efficiencies. Then they increased the power budget of less power-efficient hardware to improve the overall performance. However, the static methods are limited in scalability and usability since each processor needs to be measured and characterized accordingly.

[4] and [5] introduced dynamic methods to tackle the disadvantages of the static methods separately. Marathe et al. [5] archived performance optimization through fine-grained management of power budget, thread concurrency, and core clock rate. Gholkar et al. [4] improved performance through careful power shifting. Both works require online power and frequency monitoring.

Our DPF implementation differs from the other dynamic methods[4] and [5]. DPF assumes the power budget for a job is dynamically adjustable at runtime [8]; DPF avoids hardware monitoring as far as possible; DPF manages hardware with limitations instead of direct manipulation on power or frequency.

3. Platform and Power Capping

In this section, we present the hardware platform where our measurements were conducted at first. We present power values and performance characteristics of the processors. In the end, we illustrate scenes of a power-capped cluster where this work contributes to.

3.1. Hardware Platform and Software

Measurements in the following sections were conducted on a computing node which is chosen randomly from the CLAIX-2016 system at RWTH Aachen university. This node possesses two processors of Intel Xeon E5-2650 V4. Each processor has 12 cores

with hyper-threading deactivated, attached with 64 GB DRAM. The processors can be clocked up to 2.5 GHz with activated Turbo Boost.

The thermal designed power (TDP) of each processor is 105 watts while the power can be throttled down to 53 watts according to the runtime average power limitation (RAPL) setting[9]. RAPL is a technology introduced by Intel which enables power measurements. The power values provided by RAPL are well verified in many aspects [10],[11],[12],[13]. On the other hand, RAPL enables power capping where a user can specify a power value and a time window. RAPL enforces that the average power draw of the time window remains under the specified power value, regardless of what kind of operations are being performed.

RAPL manages power of a processor in three domains, the PKG, PP0 and DRAM domain[1] [14]. The PKG domain is in charge of power management of core and uncore area where arithmetic logic units (ALU) and last-level cache are located respectively. The DRAM domains manages power of DRAM. In this work, we only adjust the PKG domain since it has a high power draw compared to the DRAM domain. In addition, power capping the DRAM domain decreases the execution performance seriously since memory bandwidth is the performance bottleneck for many scientific-technical applications.

Compared to the per-processor power capping through RAPL, each core of the processors can be adjusted independently because of the intergrated dynamic voltrage and frequency scaling (DVFS), and fully integrated voltage regulator (FIVIR)[15] technologies. In this work, we limit the maximal frequency using Linux */sys/devices/system/cpu/cpui/cpufreq/scaling_max_freq* interface instead of setting a concrete frequency. Through the frequency limitation, hardware is allowed to choose a concrete frequency to meet a power cap automatically and flexibly.

We employ the NPB OpenMP [16] benchmarks of the size C and a home-grown synthetic benchmark (this benchmark will be introduced in the Section 5.1). The benchmarks in our measurements always occupy all available cores with parallel tasks, i.e., 24 threads.

3.2. Power Efficiency Variation of Hardware

Due to manufacturing variations, processors of the same product line can have diverse power efficiencies. We define power efficiency as the power draw of a processor performing certain operations: the higher the power draw, the less efficient the processor is. Figure 1 illustrates power efficiencies of the two processors of our testbed. The power draw of each benchmark differs. On processor 0, *ft* draws more than 90 watts while *is* consumes 61 watts in average. In particular, processor 0 is about 10% less power efficient than processor 1 (processor 0 consumes 8 watts more than processor 1 in average.).

We observed a similar power variation in many other RWTH HPC cluster's nodes with different degree of variations.

3.3. Performance Variation under Power Capping

The enforcement of a power cap causes performance degradation, i.e. applications will perform slowly. We measured relative execution time extension of NPB benchmarks

[1]The PP0 domain is not supported on our platform.

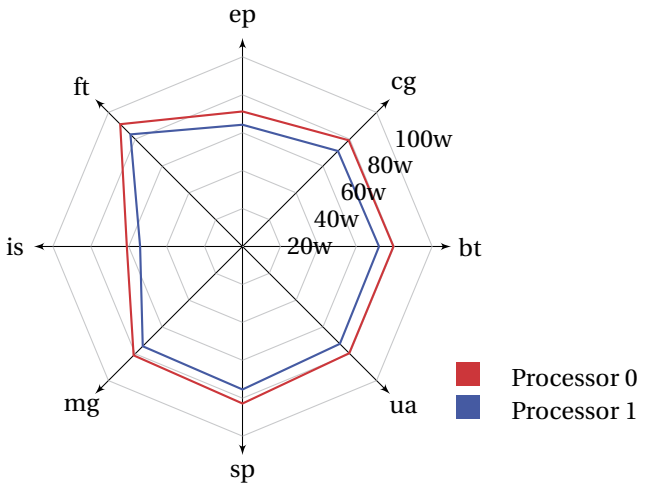

Figure 1. Power draw of two processors

capped at 53 watts compared to executions at TDP. Measurements were conducted on sockets one after another. Results are illustrated in Figure 2.

The amount of performance degradation depends on many factors. It depends on applications' peak power draw. The higher the peak power is, the more power needs to be capped and the more performance will be throttled. Since *ft* has a higher power draw than *is* illustrated in Figure 1, *ft*'s performance degradation is greater than *is*, as illustrated in Figure 2.

The degradation also depends on the performed operations by the applications, like memory-bound or compute-bound operations. *bt* and *sp* have a similar peak power draw in Figure 1, but the performance degradation differs a lot (12% vs. 24% on processor 0) since the measured L3 cache miss rates of *bt* and *sp* are quite different(2E-4 and 1.2E-3 misses per second).

In particular, the degradation depends on the power efficiency of the underlying processors. On a power-inefficient processor, more power needs to be capped compared to on a power-efficient processor. The frequency of the inefficient processor is throttled greatly and the performance degradation is tendentiously high, as illustrated by Figure 2. Processor 0 has more performance loss than processor 1 for all benchmarks.

3.4. Dynamic Power Budgeting on a Power-capped Cluster

Because of infrastructure limitations, power supplying can be insufficient for a cluster running at its peak power draw. On such a cluster, power budgeting methods were developed [8][17] to accelerate executions as far as possible. The methods schedule power to jobs dynamically and according to jobs' states. For an individual job, power budget varies from time to time. In this context, the DPF method needs to track the jobs' power budget and enforces the average power draw under the current budget.

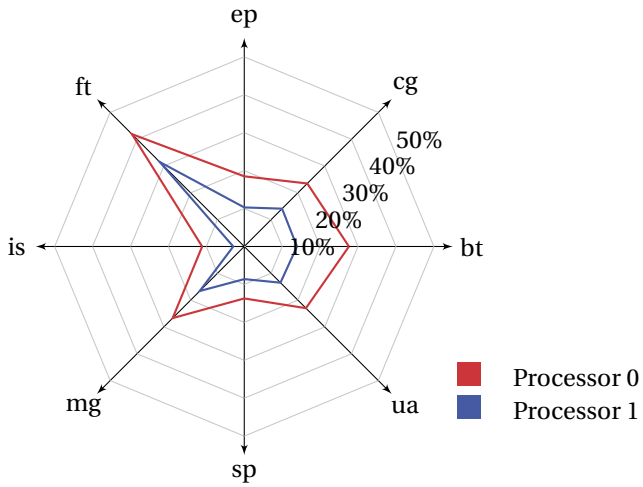

Figure 2. Relative time extension or performance degradation of executions power capped to 53 watts compared to capped to 105 watts

4. Dynamic Resource Management to Eliminate Imbalance

An execution of a parallel job on a power-capped cluster may suffer from an imbalanced execution time among parallel tasks. In this section, we justify and present our dynamic algorithm to tackle the imbalance issue and improve the jobs' performance.

4.1. Why Dynamic Management?

An imbalanced execution of a job can be caused by many factors. The job may have inherited load imbalance, i.e. parallel tasks obtain uneven loads. The load imbalance is individual to an application even to an input dataset.

In addition, the execution imbalance can also be caused by varied performance of power-capped processors as illustrated is Figure 2. The degree of imbalance is determined by the processors and the power cap.

Because of individual imbalances and runtime factors, a static analysis that predicts and handles imbalance before an execution is complex and imprecise. In contrast, a dynamic runtime imbalance tracking and handling are more promising. Assuming that a job consists of iterative executions of parallel regions and the imbalance remains constant among iterations, we track the imbalance of the previous region, calculate and distribute resources in a way to eliminate the imbalance for the next iteration. This dynamic algorithm is a light-weight solution since it does not require any prerequisite knowledge of the hardware or the software.

4.2. Load Imbalance Detection and Elimination

Normally, an HPC application executes one or multiple parallel regions iteratively. A parallel region is defined as a section of execution between two global synchronization calls, such as barriers. The entire execution time T_{app} can be calculated as a sum:

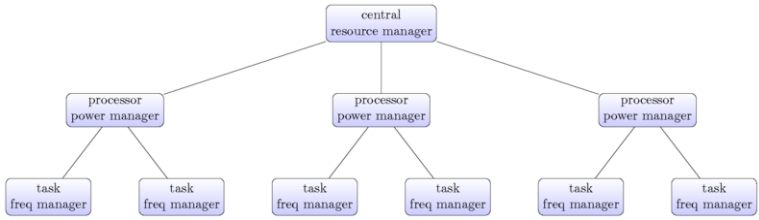

Figure 3. Two-level resource management

$$T_{app} = \sum T_r \tag{1}$$

T_r is the execution time of a parallel region r. Under the assumption that synchronization time is negligible, T_r is determined as

$$T_r = max(T_r^t), \forall t \in parallel\ tasks \tag{2}$$

namely by the slowest (critical) task.

T_{app} can be reduced by accelerating the critical task of each parallel region r through a high power budget or a high core frequency. Since a power-capped application has a limited power budget, the required power budget needs to be moved or shifted from other tasks. If the overall power cap remained and the execution is accelerated, the job's energy consumption will be reduced.

4.3. Power Shifting and Frequency Limitation to Minimize Execution Time

We developed a two-level resource management approach, the DPF, to minimize T_{app}, illustrated by a tree in Figure 3. The tree design is constructed according to the available resource management technologies, RAPL and DVFS. On top of the tree a central resource manager monitors available power-budget for the job. In the middle, a processor manager manages its own power draw using RAPL. At the bottom, a core manager manages its own frequency through DVFS.

The managers are integrated into each parallel task which are bound to physical cores. In general, the task with ID 0 of a job is the *central resource manager*. A single task on each processor manages the processor's power. Besides, each task is a core *freq manager*.

The computation time T^p of a processor within the parallel region r is defined as the maximum computation time of parallel tasks executing on the processor in Eq. (3).

$$T_r^p = max(T_r^j), \forall j\ runs\ on\ processor\ p \tag{3}$$

A critical processor $\overline{p}$ is the processor with the maximal T_r^p. Executions on $\overline{p}$ can be accelerated through enlarging its power budget PB. The required power enlargement FP (free power) needs be collected from other sockets through Eq. (4).

$$FP = \sum (SP,\ if\ T_r^p < \alpha \cdot T_r^{\overline{p}}\ and\ PB^p > MinPB,\ \forall p \in P) \tag{4}$$

SP is a predefined step power (e.g. one watt). $MinPB$ is the minimal power budget inherited in hardware to assure a stable operation. $\alpha \in (0,1)$ is a threshold that determines

when power can be shifted. α is a sensitive parameter: a high α causes corrupted resource adjustment frequently due to the tiny runtime deviation; a low α eliminates improvement potential since the resource rescheduling occurs rarely.

A critical task $\bar{t}$ within a processor is the task with the maximal execution time T_r^t. $\bar{t}$ should always run without any frequency limitation while the frequency of other tasks can be limited. In this way, more power will be allocated to the critical task. The frequency limitation is calculated using Eq. (5):

$$F_{new} = \begin{cases} 0, & \text{if } T_r^t \geq T_r^p \\ F_{current} + 1, & \text{if } T_r^t \leq \beta \cdot T_r^p \\ F_{current}, & \text{otherwise} \end{cases} \tag{5}$$

Here, the frequency setting F is similar to ACPI P-states [18] where the higher F is, the lower the actual frequency of the hardware is. β is a sensitive scaling parameter similar to α in Eq. (4). In the first case where $T_r^t \geq T_r^p$, the frequency is reset to the valid maximum. In the second case, the tasks' frequency is limited to a lower level. Otherwise, the current frequency is retained.

Through an execution, once a parallel regions r is encountered, the central resource manager checks and updates the job's current power budget. If the power budget is changed, the new budget is distributed evenly among processors. Frequency limitation of each core is reset to the valid maximum. Otherwise, power budget and frequency will be recalculated for each processor and for each core.

After a parallel region execution, each task stops its execution time. The time values are collected from *task freq managers* to the *central resource manager*. Algorithm 1 illustrate the implementation.

4.4. Implementation and Overhead

We implemented the DPF for common MPI and OpenMP jobs. To eliminate the expenditure in recognizing parallel regions, we employ OMPT [19] and PMPI [20] tools for automatic detection.

Since a parallel region can be small and the resource-scheduling time overhead compared to the execution time can be high, we introduced three techniques to eliminate the overhead:

- If the execution of an identified region is shorter than 0.01 seconds, no resource recalculation takes place.
- DPF tracks current resource settings. If the newly-calculated settings are identical to the current settings, no resource scheduling takes place.
- Hardware setting occurs locally and in parallel. Each task manages its own core frequency. A single task of each processor enforces the power cap.

In particular, the DPF only tracks time consumption. Monitoring of other hardware settings, like actual power draw and actual frequency, is unnecessary since it is irrelevant to calculate settings of the next step. Using this method, monitoring overheads can be eliminated essentially.

Algorithm 1: Resource management

```
1  Function resource_scheduling()
2      if taskID = centralManagerID then
3          calculate and distribute power budget for each processor;
4      if taskID = processorManagerID then
5          receive and set power cap;
6          calculate and distribute freq;
7      set freq;
8      start time measurement;
9  Function time_collection()
10     stop time measurement;
11     send time to processor manager;
12     if taskID = processorManagerID then
13         receive time;
14         calculate processors' critical time;
15         send processor critical time to the central manager;
16     if taskID = centralManagerID then
17         receive time;
18 Function job_execution()
19     Parallel
20         call resource_scheduling();
21         doing some operations;
22         call time_collection();
```

5. Evaluation

At the beginning, we illustrate how power and clock rate are managed for a synthetic benchmark. Then, we present the overhead introduced by DPF. At the end, we evaluate energy and execution time reduction through DPF compared to a native strategy with NPB benchmarks[2].

5.1. Dynamic Power and Clock Rate Management

As described in previous sections, the dynamic resource manager should a) enforce the power draw under a specified power cap continuously, b) react to a changed power limitation quickly and c) manage resources to eliminate execution imbalance. We verified the management process with a synthetic benchmark on the hardware platform.

The synthetic benchmark executes a parallel region in a loop. The iterations are divided into three phases. In the first phase, power cap for each processor is changed from 80 watts to 70 watts (from 160 watts to 140 watts for two processors). In the second

[2]The EP, UA and LU benchmarks are not suitable for the investigation since they only have very short regions, or a single parallel region or inconsistent execution time.

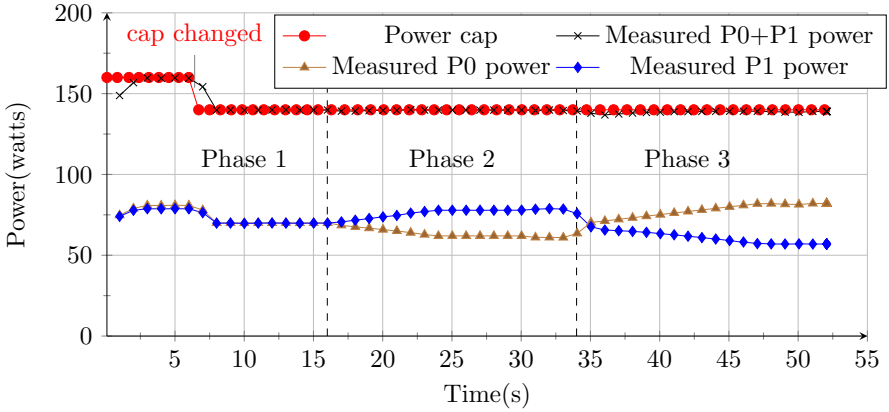

Figure 4. Run-time power monitoring

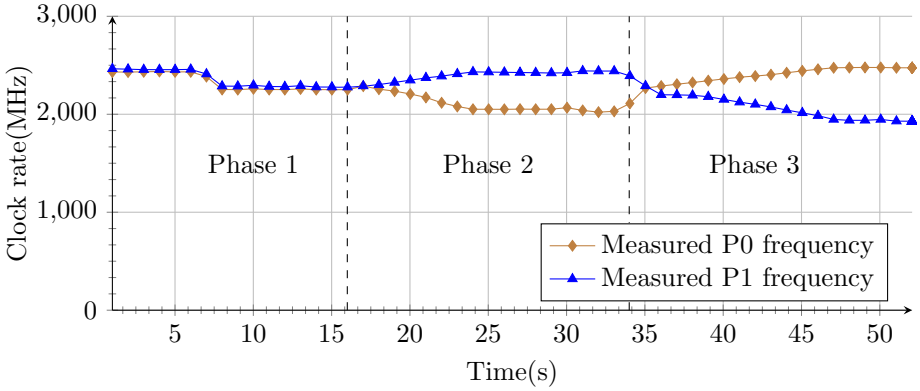

Figure 5. Run-time processor frequency monitoring

phase, tasks on processor 1 obtain 25% more load than the others. In the third phase, load of the two processors is reversed.

During the benchmark execution, we sampled the actual power draw and core frequency at 1 Hz. The results are illustrated in Figure 4 and Figure 5. As expected, the power draw remains permanently under the configured power cap, except the second after the cap is throttled from 80 to 70 watts (the measured power amounts to 77 watts). This violation is due to the interleaving of the sampling and power-adjusting points.

The measured power and frequency of phases 2 and 3 in Figures 4 and 5 illustrate that the resources are scheduled to eliminate load imbalance. In phase 2, more power is shifted to processor 1 whose cores are clocked up. In phase 3, power is shifted back to processor 0 immediately after a new load imbalance is detected.

5.2. Overhead of the Resource Management

Since the time overhead of DPF can be critical for short parallel regions, we evaluate the overhead with NPB OpenMP benchmarks.

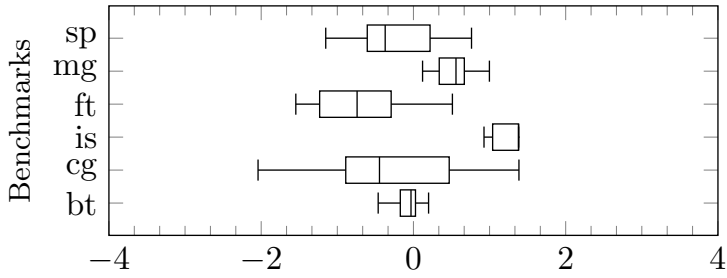

Figure 6. Measured DPF relative overheads in percent

During the measurements, power was capped to TDP (105 watts) on each processor. The actual execution is not power capped at all. However, DPF calculates power and frequency settings continuously and may set the hardware rarely.

We started each benchmark ten times with or without DPF and recorded the execution time. The overheads calculated as

$$Overhead = \frac{T_{DPF}}{T_{NODPF}} - 1$$

is illustrated in Figure 6. T_{DPF} and T_{NODPF} are the execution time with DPF and the average execution time without DPF, respectively.

The executions with DPF have some deviations up to $\pm 2\%$. Regardless of the deviations, the DPF overhead amounts to lower than 2% in the worst case, and in average lower than 1%.

5.3. Execution Time and Energy Consumption Optimization

We attached DPF to NPB OMP benchmarks of size C for validation. Capping to 55, 60, 65 and 70 watts, we measured execution time and energy consumption. Figure 7 illustrates normalized DPF time and energy compared to the values of a native strategy. The native strategy sets equal and constant power caps among processors.

For most benchmarks the achieved execution time improvements are higher than the observable deviation illustrated in Figure 6, except the *cg* at 65 watts and *sp* at 70 watts. The average improvements amount to about 4% for all power caps. In some cases, executions can be accelerated by up to 12.8%. At the same time, energy can be saved by up to 12.31%.

6. Conclusion

Because of the high power draw of an HPC cluster, the power needs be capped to avoid hardware damage in the future. However, power capping causes performance variation among processors due to their distinct power efficiencies.

In this work, we introduced a dynamic method, the DPF, that schedules power and limits frequencies to tackle the performance variation issue. DPF monitors runtime and hardware in a light-weight way. It succeeds in remaining the power draw under a specified power cap. At the same time, it accelerates executions and reduces energy consump-

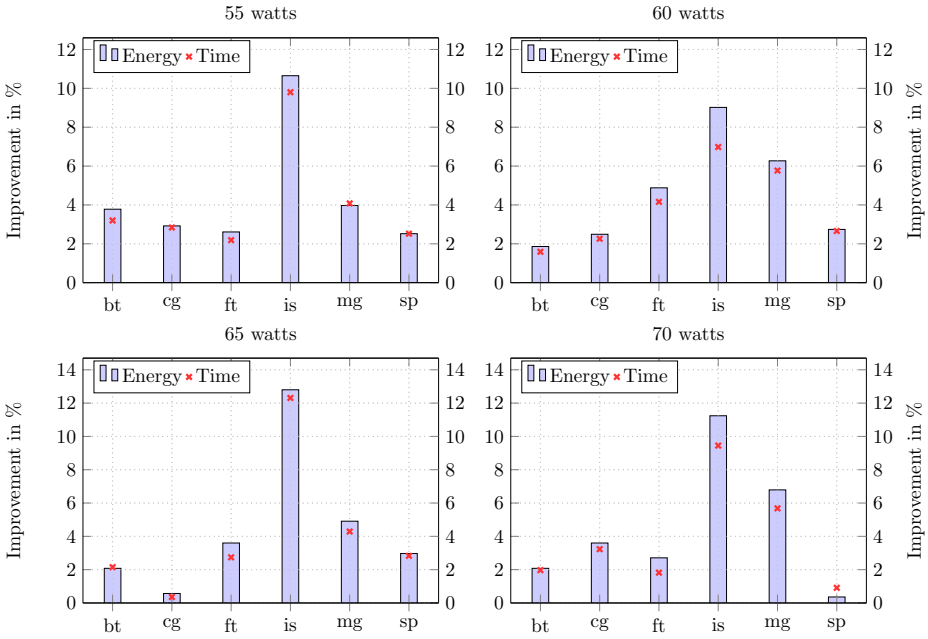

Figure 7. Improvements provided by DPF for executions with NPB OMP benchmarks

tion. For instance, applying DPF on NPB benchmarks executions were accelerated up to 12.8% and the energy consumption was reduced up to 12.3%.

In the future, we will improve the DPF for large-scale executions where hardware variation is more significant and DPF overheads need be reduced much more.

Acknowledgment

(Part of) this work was performed under the POP project and has received funding from the European Union's Horizon 2020 research and innovation programs under grant agreement 824080.

References

[1] Exascale Initiative Steering Committee. A decadal plan for providing exascale applications and technologies for doe mission needs. `https://permalink.lanl.gov/object/tr?what=info:lanl-repo/lareport/LA-UR-11-02200`, 2010.

[2] Barry Rountree, Dong H Ahn, Bronis R De Supinski, David K Lowenthal, and Martin Schulz. Beyond dvfs: A first look at performance under a hardware-enforced power bound. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*, pages 947–953. IEEE, 2012.

[3] Tapasya Patki, David K Lowenthal, Barry Rountree, Martin Schulz, and Bronis 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*, pages 173–182. ACM, 2013.

[4] Neha Gholkar, Frank Mueller, Barry Rountree, and Aniruddha Marathe. Pshifter: Feedback-based dynamic power shifting within hpc jobs for performance. In *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*, pages 106–117. ACM, 2018.

[5] Aniruddha Marathe, Peter E Bailey, David K Lowenthal, Barry Rountree, Martin Schulz, and Bronis R de Supinski. A run-time system for power-constrained hpc applications. In *International conference on high performance computing*, pages 394–408. Springer, 2015.

[6] Neha Gholkar, Frank Mueller, and Barry Rountree. Power tuning hpc jobs on power-constrained systems. In *Proceedings of the 2016 International Conference on Parallel Architectures and Compilation*, pages 179–191. ACM, 2016.

[7] Yuichi Inadomi, Tapasya Patki, Koji Inoue, Mutsumi Aoyagi, Barry Rountree, Martin Schulz, David Lowenthal, Yasutaka Wada, Keiichiro Fukazawa, Masatsugu Ueda, et al. Analyzing and mitigating the impact of manufacturing variability in power-constrained supercomputing. In *SC'15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12. IEEE, 2015.

[8] Bo Wang, Dirk Schmidl, Christian Terboven, and Matthias S Müller. Dynamic application-aware power capping. In *Proceedings of the 5th International Workshop on Energy Efficient Supercomputing*, page 1. ACM, 2017.

[9] Howard David, Eugene Gorbatov, Ulf R Hanebutte, Rahul Khanna, and Christian Le. Rapl: memory power estimation and capping. In *2010 ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED)*, pages 189–194. IEEE, 2010.

[10] George Stantchev, William Dorland, and Nail Gumerov. Fast parallel particle-to-grid interpolation for plasma pic simulations on the gpu. *Journal of Parallel and Distributed Computing*, 68(10):1339–1349, 2008.

[11] Marcus Hähnel, Björn Döbel, Marcus Völp, and Hermann Härtig. Measuring energy consumption for short code paths using rapl. *ACM SIGMETRICS Performance Evaluation Review*, 40(3):13–17, 2012.

[12] Huazhe Zhang and H Hoffman. A quantitative evaluation of the rapl power control system. *Feedback Computing*, 2015.

[13] Daniel Hackenberg, Thomas Ilsche, Robert Schöne, Daniel Molka, Maik Schmidt, and Wolfgang E Nagel. Power measurement techniques on standard compute nodes: A quantitative comparison. In *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 194–204. IEEE, 2013.

[14] Martin Dimitrov. Intel power governor. `https://software.intel.com/en-us/articles/intel-power-governor`, 2012. [Online; accessed 19-July-2019].

[15] Edward A. Burton, G Schrom, Fabrice Paillet, Jonathan Douglas, William J. Lambert, Kaladhar Radhakrishnan, and Michael Hill. Fivr fully integrated voltage regulators on 4th generation intel core socs. pages 432–439, 03 2014.

[16] Nas parallel benchmarks. `https://www.nas.nasa.gov/publications/npb.html`. [Online; accessed 1-July-2019].

[17] Daniel Ellsworth, Tapasya Patki, Martin Schulz, Barry Rountree, and Allen Malony. Simulating power scheduling at scale. In *Proceedings of the 5th International Workshop on Energy Efficient Supercomputing*, page 2. ACM, 2017.

[18] Acpi specification. `https://uefi.org/sites/default/files/resources/ACPI_6_2.pdf`.

[19] Alexandre Eichenberger, John Mellor-Crummey, Martin Schulz, Nawal Copty, John DelSignore, Robert Dietrich, Xu Liu, Eugene Loh, and Daniel Lorenz. Ompt and ompd: Openmp tools application programming interfaces for performance analysis and debugging. In *International Workshop on OpenMP (IWOMP 2013)*, 2013.

[20] Sava Mintchev and Vladimir Getov. Pmpi: High-level message passing in fortran77 and c. In *International Conference on High-Performance Computing and Networking*, pages 601–614. Springer, 1997.