Power Capping of CPU-GPU Heterogeneous Systems through Coordinating DVFS and Task Mapping

Toshiya Komoda*, Shingo Hayashi*, Takashi Nakada*, Shinobu Miwa*, Hiroshi Nakamura*
* *Graduate School of Information Science and Technology, The University of Tokyo*
7-3-1 Hongo, Bunkyo-ku, Tokyo 113-8656, Japan
Email: komoda,hayashi,nakada,miwa,nakamura@hal.ipc.i.u-tokyo.ac.jp

***Abstract*—Future computer systems are built under much stringent power budget due to the limitation of power delivery and cooling systems. To this end, sophisticated power management techniques are required. Power capping is a technique to limit the power consumption of a system to the predetermined level, and has been extensively studied in homogeneous systems. However, few studies about the power capping of CPU-GPU heterogeneous systems have been done yet.**

In this paper, we propose an efficient power capping technique through coordinating DVFS and task mapping in a single computing node equipped with GPUs. In CPU-GPU heterogeneous systems, settings of the device frequencies have to be considered with task mapping between the CPUs and the GPUs because the frequency scaling can incurs load imbalance between them. To guide the settings of DVFS and task mapping for avoiding power violation and the load imbalance, we develop new empirical models of the performance and the maximum power consumption of a CPU-GPU heterogeneous system. The models enable us to set near-optimal settings of the device frequencies and the task mapping in advance of the application execution. We evaluate the proposed technique with five data-parallel applications on a machine equipped with a single CPU and a single GPU. The experimental result shows that the performance achieved by the proposed power capping technique is comparable to the ideal one.

***Keywords*-GPGPU; Power Capping; DVFS; Task Mapping;**

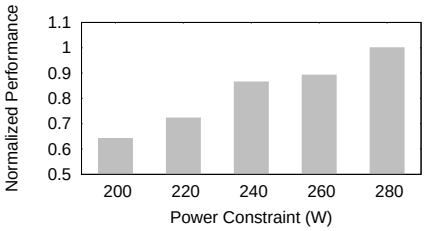
I. Introduction

Future computer systems are built under much stringent power budget due to the limitation of power delivery and cooling systems. Therefore, many researchers have investigated power management techniques to optimize the energy usage in a large scale system [6], [7], [10], [11]. Power capping, which enables us to dynamically limit the peak power consumption of components or computing nodes, is one of the most fundamental technologies of these power management [7]. Meanwhile, in order to maximize energy efficiency, heterogeneous systems equipped with GPUs have become popular, especially in systems for HPC [1], [15], [19]. Developing power capping technique is also important for these heterogeneous systems.

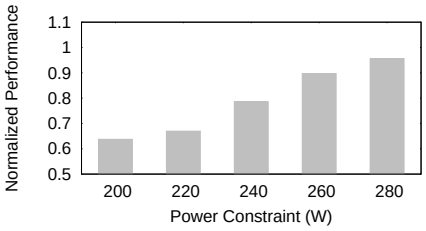
Although many power capping techniques based on DVFS have been proposed for homogeneous systems, these are not directly applicable to heterogeneous systems. In the applications running on heterogeneous systems with GPUs, hybrid computing is often used, which means that parallel tasks are distributed to both of CPUs and GPUs and that the tasks are executed in parallel [9], [12], [17], [18]. When we directly apply conventional power capping techniques with DVFS to these applications, load imbalance occurs among CPUs and GPUs because the amounts of the performance fluctuations by DVFS are quite different between CPUs and GPUs. The load imbalance degrades the application performance significantly. Thus, we cannot separate the problem of frequency settings of CPUs and GPUs from the load balancing problem among CPUs and GPUs.

In Figure 1, we show the potential benefit of coordinating DVFS and task mapping under the power constraints. The bars show the normalized performance of parallel applications running on a heterogeneous system with a single CPU and a single GPU. To keep the power consumption under the power constraint, we use DVFS. The performance is normalized to the performance with ideal settings of DVFS and task mapping under the power constraint, in which we can avoid load imbalance caused by DVFS through adjusting task mapping. The detailed experimental setup is described in section V. In the power capping only with DVFS, lack of mechanism to adjust load balancing results in significant performance degradation under the power constraint. In summary, there is a big room to improve efficiency of power capping for heterogeneous parallel applications through coordinating DVFS and task mapping.

In this paper, we propose an efficient power capping technique through coordinating DVFS and task mapping in a CPU-GPU heterogeneous system. Performance and power consumption of the heterogeneous systems are complicated functions of the CPU frequency, the GPU frequency and the task mapping. To explore the parameter space without runtime overhead, we build a proactive power capping technique in which the settings of DVFS and task mapping are determined in advance of the execution. In order to guide the settings of DVFS and task mapping, the proposed technique utilizes new empirical models of the performance and the power of the target heterogeneous system. With a small number of profiles, the models enable us to get the optimal set of parameters under a given power constraint. The contributions of this paper are listed below.



(a) KMEANS



(b) SGEMM

Figure 1. Performance degradation by power capping only with DVFS. (normalized to the ideal settings of device frequencies and task mapping under each power constraint.)

- As far as we know, this is the first work on the power capping of CPU-GPU heterogeneous systems through coordination of DVFS and task mapping.
- We build empirical models to precisely predict the performance and the maximum power consumption with the given settings of the CPU frequency, the GPU frequency, and the task mapping.
- We show the effectiveness of the proposed technique through evaluating five GPGPU applications in a platform equipped with a single GPU under five different power constraints. The experimental result shows that we can achieve more than 93% of performance compared to the ideal one in 24 cases out of the 25 cases.

The paper is organized as follows. Section 2 explains the background. Section 3 describes the overview of the proposed technique. The proposed models of performance and power are detailed in Section 4. We evaluate the proposed technique in Section 5. We refer the related work in Section 6, then summarize this paper in Section 7.

II. Background

A. Power Management in Computer Systems

Power capping is a fundamental technique to manage the power consumption in computer systems. It eliminates the burst of power consumption in components or computing nodes and keeps the power consumption under the given power constraint. At the component level, power capping techniques for CPUs [6] or system memories [10] have been already proposed. On a different level, power capping for a single computing node is important for power limited large-scale systems [11]. Based on power capping techniques, we can optimize the available power capacity among servers according to online activity of running applications [7].

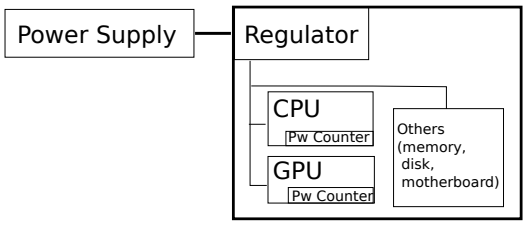


Figure 2. A heterogeneous system with GPUs.

B. Heterogeneous Systems with GPUs

In this paper, we consider a power capping technique for a single computing node. Figure 2 illustrates the assumed machine architecture in this paper. The architecture consists of a single CPU and a single GPU. Recent commercial CPUs and GPUs have already supported dynamic voltage and frequency scaling (DVFS) technology for their power management. Also, power consumption of CPUs and GPUs can be measured by reading special hardware counters integrated into the devices. Software interfaces for the counters are provided by the vendors.

Previous research reported that motherboards, hard disks and other I/O components exhibit relatively small fluctuations in power consumption over a variety of workloads [7]. Since CPUs and GPUs are the main sources of dynamic fluctuation of power consumption of a computing node, we regard them as the targets of the power management.

C. Parallel Computation with CPUs and GPUs

Hybrid data-parallel computation with both CPUs and GPUs increases both performance and energy efficiency of a heterogeneous computing node [14]. With recent portable GPGPU programming environments, such as OpenCL [3] or OpenACC [2], many researchers have proposed compilers and runtime systems to make full use of CPUs and GPUs in data parallel processing [9], [12], [16], [17], [18].

Figure 3 illustrates a simple example of the parallel computation with a single CPU and a single GPU. In Figure 3, CPU executes tasks related to a sub-array with the 0th element to the 29th element of an array. On the other hand, GPU executes tasks related to the rest of the elements in the array which are from the 30th to the 99th element. In this example, the percentage of the CPU tasks is calculated as $30/100 = 0.3$ and that of the GPU tasks is calculated as $70/100 = 0.7$.

Since data-parallel tasks can be decomposed into independent tasks which are related to individual data elements, we can flexibly control the percentage of tasks for each device. Previous research shows that compilers or runtime systems can adjust the task mapping to balance the load between CPUs and GPUs. The optimal percentages of CPU tasks and

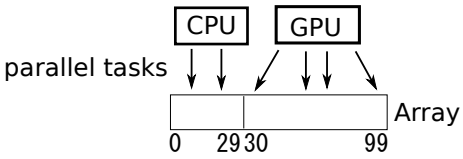


Figure 3. Hybrid computation with a CPU and a GPU.

GPU tasks are dependent on applications because relative speedup by using GPUs is significantly different between applications.

III. Proposed Power Capping Technique

A. Approach

As shown in Figure 1, it is important to coordinate DVFS and task mapping in a CPU-GPU heterogeneous system. To do so, one approach is to integrate adaptive task mapping into the existing feedback controller of DVFS. However, in this approach, a large number of parameter adjustments are required at runtime in order to explore the large parameter space. Even worse, in the current CPU-GPU architecture, adaptive task mapping requires additional data transfers between the CPU memory and the GPU memory. It incurs significant performance overhead.

To avoid the issue, we will investigate a proactive method in which we explore the parameter space in advance of the execution. The behavior of a typical GPGPU application does not change drastically during the executions because many GPGPU applications use iterative algorithms [18]. The above fact tells us that proactive methods are promising to coordinate DVFS and task mapping in a CPU-GPU heterogeneous system.

B. Profile-Based Power Capping

In the proposed power capping technique, we determine the frequencies of the CPU and the GPU, and the percentages of tasks mapped to the CPU and the GPU at the beginning of the application execution. Figure 4 shows the overview of the proposed power capping technique. To enable us to set the parameters in advance of the application execution, we take profile information by executing the application only with the CPU and only with the GPU. Based on the profiles, empirical models predict the execution time and the maximum power consumption in a hybrid execution by utilizing both the CPU and the GPU. The models enable us to examine all possible settings of DVFS and task mapping without the exhaustive profiling of the all possible parameter settings. With the optimal parameter predicted by the model, we execute the application.

It is possible that the system exceeds the power constraint with the pre-determined parameters due to the prediction error of maximum power consumption. Although such a case is rare because the proposed empirical models are sufficiently precise, as evaluated in Section V, we can adjust the CPU frequency with the existing power capping technique at runtime to prevent the power violation. In Figure 4, the *Reactive DVFS controller* is responsible for this. In this case, individual adjustment of the CPU frequency does not incur large performance degradation because the value of the power consumption has already been close to the power constraint and the adjustment can be done in a very small range of frequencies.

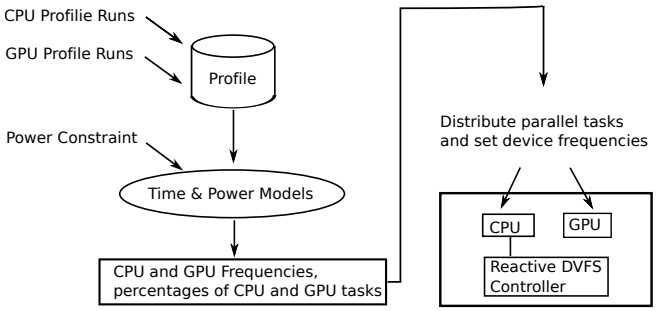


Figure 4. An overview of the profile-based power capping.

IV. Empirical Models

A. Model Parameters and Profiling Information

In Table I, we summarize the model parameters and profile information of the execution only with the CPU and the GPU. The control parameters include the frequency of the CPU (f_{cpu}), the frequency of the GPU (f_{gpu}) and the percentages of CPU tasks (r_{cpu}) and GPU tasks (r_{gpu}). The model outputs the normalized execution time of applications (t_{total}) and the maximum power consumption (p_{node}) with the given parameter set: $(f_{cpu}, f_{gpu}, r_{gpu}, r_{cpu})$. Note that the sum of percentages of CPU tasks and GPU tasks must be 100: i.e. $r_{cpu} + r_{gpu} = 100$.

In the profiling runs, we set the percentages of CPU and GPU tasks to $r_{cpu} = 100, r_{gpu} = 0$ (CPU-only Profile) and $r_{cpu} = 0, r_{gpu} = 100$ (GPU-only Profile). With the settings of task mapping, we profile the execution time and maximum power consumption for all possible set of device frequencies. The profile information includes the power consumption of the CPU, the GPU, and the computing node. Also, we collect the execution time of each parallel kernel function during the execution. We profile the execution time normalized to the execution time of the execution only with the GPU at the highest frequencies. Thus, the model is applicable to executions with different data whose sizes are different from that of the data used in the profiling runs.

We can take the profiling information when we install applications in the system. The system must support measuring the power consumption of the computing node, the power consumption of the CPU and GPU.

Predicted Values	
t_{total}	Total Execution time
p_{node}	Power Consumption of the Node
Control Parameters	**Description**
f_{cpu}	Frequency of the CPU
f_{gpu}	Frequency of the GPU
r_{cpu}	percentage of CPU tasks
r_{gpu}	percentage of GPU tasks
CPU(GPU)-only Profile	
$K_i^{c(g)}$	Execution Time of ith Parallel Kernel
$P_{node}^{c(g)}$	Power of the Node
$P_{cpu}^{c(g)}$	Power of the CPU
$P_{gpu}^{c(g)}$	Power of the GPU

Table I
MODEL PARAMETERS AND PROFILE INFORMATION.

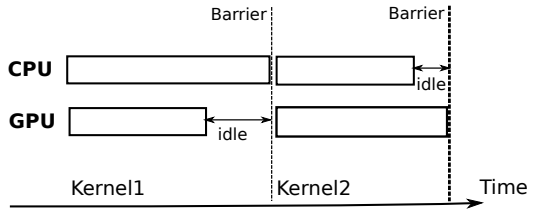


Figure 5. Fine-grained synchronization between the CPU and the GPU.

B. Execution Time Model

The total execution time of the application can be represented by the sum of the execution time of the parallel kernel functions as the following equation:

$$t_{total} = \sum_i K_i(f_{cpu}, f_{gpu}, r_{cpu}, r_{gpu}).$$

The execution time of the i-th kernel is represented as $K_i(f_{cpu}, f_{gpu}, r_{cpu}, r_{gpu})$.

To estimate the execution time of the i-th kernel (K_i) in hybrid computations, in which both the r_{cpu} and the r_{gpu} are not zero, we make the following two assumptions: 1. the actual execution time on the CPU or the GPU is proportional to the amount of mapped tasks, and 2. global barrier is required after the execution of each parallel kernel. The global barrier is illustrated in Figure 5. In this situation, the faster device must wait for the other device at the synchronization point. Using the two assumptions and the profile information, we can estimate the execution time of the i-th kernel with the following equation:

$$K_i = max\left\{K_i^c(f_{cpu}, f_{gpu}) \times \frac{r_{cpu}}{100}, K_i^g(f_{cpu}, f_{gpu}) \times \frac{r_{gpu}}{100}\right\}.$$

C. Power Model

Total node power $p_{node}(f_{cpu}, f_{gpu}, r_{cpu}, r_{gpu})$ can be decomposed into three terms as the following equation:

$$\begin{aligned} p_{node} = \quad & p_{idle}(f_{cpu}, f_{gpu}) \\ & + p_{cpu}(f_{cpu}, f_{gpu}, r_{cpu}) + p_{gpu}(f_{cpu}, f_{gpu}, r_{gpu}). \end{aligned}$$

The term p_{idle} represents idle power consumption, which depends on the frequencies of the CPU and the GPU but is independent of the task mapping. It includes static power of the system. The terms p_{cpu} and p_{gpu} represent power consumptions in the CPU and the GPU, which depend on all the control parameters ($f_{cpu}, f_{gpu}, r_{cpu}, r_{gpu}$).

1) Idle Power: We estimate the idle power p_{idle} by using the CPU-only profiles. Simply, we estimate it by subtracting the dynamic power consumption of the CPU from the total power consumption of the node as the following equation:

$$\begin{aligned} p_{idle} = \quad & P_{node}^c(f_{cpu}, f_{gpu}) \\ & - \left\{P_{cpu}^c(f_{cpu}, f_{gpu}) - P_{cpu}^g(f_{cpu}, f_{gpu})\right\}. \end{aligned}$$

The dynamic power consumption of the CPU is approximated by the expression $P_{cpu}^c - P_{cpu}^g$. The value of p_{idle} includes the static power of CPUs, the static power of GPUs, and the power consumption of the all other components (mother board, hard disk etc.).

2) Dynamic Power: The CPU and the GPU are not always in busy because fine-grained synchronization is required as illustrated in Figure 5. Because of the idle period in the devices, p_{cpu} and p_{gpu} fluctuate according to the activity of the CPU and the GPU with the given task mapping. To model the phenomenon, we assume that the power consumptions are proportional to the fraction of busy time to the total execution time. This relationship is expressed in the following equation:

$$\begin{aligned} p_{cpu} = \quad & \alpha_{cpu} \times \left\{P_{cpu}^c(f_{cpu}, f_{gpu}) - P_{cpu}^g(f_{cpu}, f_{gpu})\right\}, \\ p_{gpu} = \quad & \alpha_{gpu} \times \left\{P_{gpu}^g(f_{cpu}, f_{gpu}) - P_{gpu}^c(f_{cpu}, f_{gpu})\right\}. \end{aligned}$$

The term α_{cpu} and the term α_{gpu} represent the fraction of busy time in the CPU and that in the GPU respectively. In the equations, the expression $P_{cpu}^c - P_{cpu}^g$ and the expression $P_{gpu}^g - P_{gpu}^c$ are used as the approximations of the maximum amount of the dynamic power consumption in the CPU and that in the GPU respectively.

To estimate the fraction of busy time α_{cpu} and α_{gpu} from the profile information, we define the busy time of each device during execution as the sum of the actual execution time of each kernel function. The definition is shown in the following equation:

$$t_{c(g)pu} = \sum_i K_i^{c(g)}(f_{cpu}, f_{gpu}) \times \frac{r_{c(g)pu}}{100}.$$

The busy times are denoted by t_{cpu} and t_{gpu}. Using them, we estimate the fraction of busy time of the CPU α_{gpu} and that of the GPU α_{gpu} as the following equations:

$$\alpha_{c(g)pu} = \frac{t_{c(g)pu}(f_{cpu}, f_{gpu}, r_{c(g)pu})}{t_{total}(f_{cpu}, f_{gpu}, r_{c(g)pu})}.$$

D. Parameter Selection

After taking the profiling information, we examine the all possible set of parameters with the empirical models and build the optimal parameter tables which stores the list of the optimal parameter sets for every value of the power constraint. We have to build the tables only once. Plus, it does not take much time to build it even if we explore the optimal parameter sets with exhaustive search algorithms because we can quickly estimate the execution time and the maximum power consumption for any parameter set with the model. At the beginning of the application execution, the runtime system looks up the table to get the optimal parameter set under the given power constraint.

V. Evaluation

A. Methodology

In the experiment, we use a computing node with a single multi-core CPU and a single GPU. The details of the platform are shown in Table II. We measure the total power consumption of the machine at every one second with "watts up? .net" [4]. To measure power consumed in the CPU and the GPU in profiling executions, we use RAPL [6] API to read the hardware counters related to the device power consumption. We use Linux "cpufreq-set" command to change the CPU frequencies and use "nvidia-smi" command to change the GPU frequencies.

We evaluate five GPGPU applications selected from rodinia benchmark suite [5] and from BLAS library. The applications are summarized in Table III. Compilers or runtime libraries with the capability of scheduling tasks between CPUs and GPUs have not been publicly available yet. Therefore, we manually implemented heterogeneous task mapping mechanism for each evaluated application. They are compiled with gcc 4.4 and with nvcc 5.0. We use the "-O3" flag for the compiler optimization. We enlarge the provided data sets and use them for evaluations.

In the developed hybrid applications, we use the existing parallel codes of rodinia benchmark suite and BLAS library. The parallel codes use OpenMP for CPU side computations and they use CUDA for GPU side computations. In the hybrid *SGEMM* implementation, we use Intel MKL for the CPU side computation and NVIDIA CUBLAS for the GPU side computation. In the hybrid executions, the data elements are transferred only when they are really necessary on the GPU at the given task mapping. In the experiment, we assume that we can change the percentages of CPU and GPU tasks by 10% in a single step.

The reactive DVFS controller for a CPU, which eliminates the possibility of power violation caused by the errors of power prediction, has not been integrated into the evaluation system. Instead of using the adaptive controller, we add a margin in the estimated power consumption and guarantee that the maximum power consumption of the system is under the power constraint.

CPU	Intel Core i7 x1 (6core)
GPU	Nvidia Tesla K20c x1
CPU Frequencies(GHz)	1.2 - 3.2 (0.2 GHz step)
GPU Frequencies(MHz)	614, 640, 666, 705, 754
OS	Linux

Table II
MACHINE SETUPS FOR THE EVALUATION.

Application	Description	Input Description
BFS	Graph Traversal	5M node random graph
HOTSPOT	Heat Simulation	(1024, 1024) Grid
KMEANS	Clustering	494020 data items
PF	Model Estimation	50K particles
SGEMM	BLAS Library	(2k, 2k) matrices x100

Table III
DESCRIPTION OF THE BENCHMARKS.

B. Model Verification

In Figure 6, and 7, we plot the errors in the execution time model and those in the power model. The GPU frequency is fixed at 614MHz, The x-axis denotes the percentage of GPU tasks (r_{gpu}). We show the result of the percentage of GPU tasks from 50% to 100% because the optimal percentages of GPU tasks are between the range for all the evaluated applications. The y-axis denotes the CPU frequency (f_{cpu}). The z-axis denotes the errors in the proposed model. The execution time is normalized to the execution time of the execution only with the GPU at the highest frequency.

We can see that the predicted values are closely fitted to the actual measurement. The average errors for 50%-90% GPU tasks are shown in Table IV. The average error of the execution time is up to 14%. The average errors of the maximum power consumption are up to 5.1 Watt.

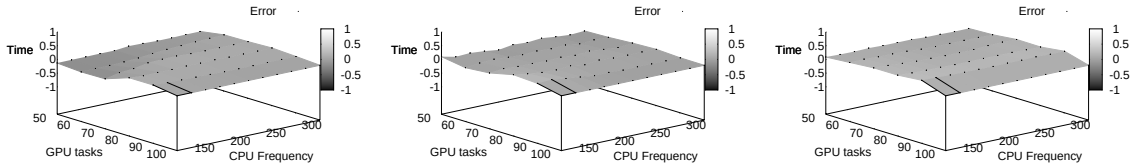
C. Performance Under the Power Constraint

Figure 8 shows the normalized performance with the proposed power capping techniques. The x-axis denotes the power constraint. The y-axis denotes the performance normalized to the performance of the execution with ideal settings of DVFS and task mapping under the given power constraint. We compare power capping techniques with fixed task mappings to the proposed one. They are listed below.

- cpu-dvfs: The percentage of GPU tasks is fixed at 0%.

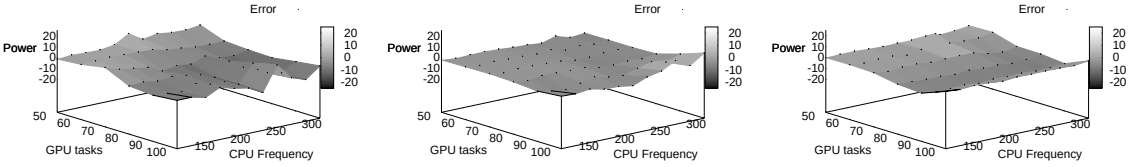
Application	Ave. Time(%)	Ave. Power(W)
BFS	9.18	4.95
HOTSPOT	5.97	3.65
KMEANS	10.41	5.10
PF	14.07	3.49
SGEMM	10.67	1.49

Table IV
THE AVERAGE ERROR OF THE MODEL. (THE PERCENTAGE OF GPU TASKS: 50% - 90%.)



(a) BFS (b) HOTSPOT (c) PF

Figure 6. The errors in the estimated execution time: GPU 614MHz.



(a) BFS (b) HOTSPOT (c) PF

Figure 7. The errors in the estimated power (watt): GPU 614MHz.

- gpu-dvfs: The percentage of GPU tasks is fixed at 100%.
- dvfs: The percentage of GPU tasks is fixed at the optimal percentage under no power constraint.

For *cpu-dvfs*, *gpu-dvfs* and *dvfs*, we assume that we can set the optimal device frequencies under the power constraint at the beginning of the execution.

As mentioned in Section III, the initial parameters set by the proposed technique potentially causes power violation due to the error of prediction. Although we have not yet integrated a feedback power controller into our evaluation systems, in the experiment, the proposed technique could always set the parameters which keep the power consumption of the node under the power constraint.

In the region with strict power constraint (200W-240W), the CPU and the GPU must run with lower frequency levels. Because the range of frequency scaling in the CPU is larger than that of the GPU, the performance degradation caused by DVFS tends to be larger in the CPU. This causes load imbalancing between the CPU and the GPU in *dvfs*, which directly results in large performance degradation. On the other hand, the performance of *gpu-dvfs* is relatively constant throughout the various power constraints. However, *gpu-dvfs* cannot achieve the optimal performance in any cases. Plus, *gpu-dvfs* fails to run under the most strict power constraint (200W) in *BFS*, *HOTSPOT* and *PF*. With the proposed technique, we can migrate parallel tasks between the CPU and the GPU with considering the DVFS effects on the performance of the CPU and the GPU. Thus, we can eliminate inefficient execution caused by load imbalance. In summary, the proposed power capping technique achieves more than 93% of performance compared to the ideal performance under the given power constraint in 24 cases out of the 25 cases.

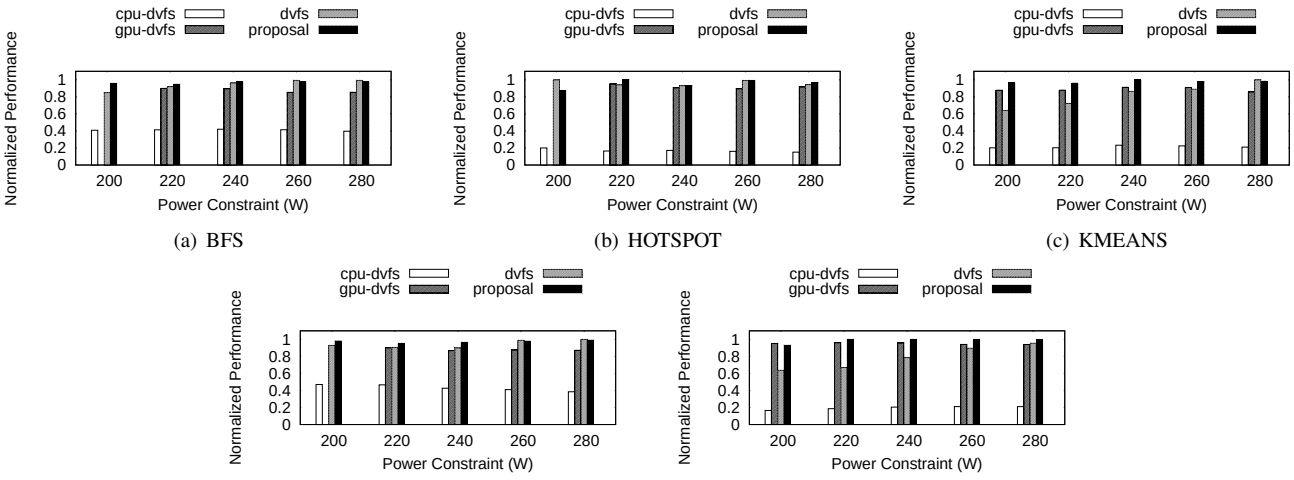
Power Constraint (W)	Proposal	Ideal
200	(1.4, 705, 90)	(1.2, 758.0, 100.0)
220	(1.6, 758, 90)	(1.8, 758.0, 90.0)
240	(2.2, 758, 90)	(1.8, 758.0, 90.0)
260	(2.6, 758, 90)	(2.8, 666.0, 80.0)
280	(3.0, 705, 80)	(3.0, 758.0, 80.0)

Table V
THE SELECTED PARAMETERS FOR KMEANS. (CPU FREQUENCY IN GHZ, GPU FREQUENCY IN MHZ, PERCENTAGE OF GPU TASKS IN %).

Table V shows the parameters selected in the proposed technique and the ideal settings for *KMEANS*. Although the selected parameters are not the same between the proposed technique and the ideal settings, the trends in the parameter selections are similar. As the power constraint gets stricter, we should move the tasks to the GPU while we keep the GPU frequency high. Also, Figure 9 shows the performance with the proposal under the power constraints normalized to the maximum performance without any power constraint.

D. Using Different Size for Profiling

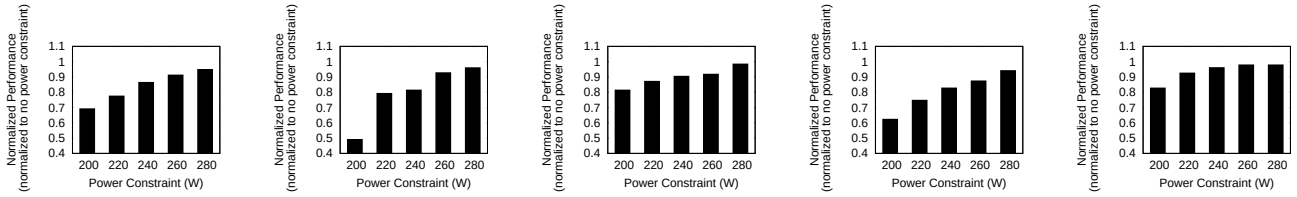
Since the proposed models use the normalized execution times to select the device frequencies and the percentage of CPU and GPU tasks, we can use the same profiles for the execution with data whose size is different from that of the data used in the profiling runs. We prepare different input data sets whose sizes are half of the sizes of the corresponding input data listed in Table III. We use the smaller data sets for profiling runs and use the data sets listed in Table III for actual evaluation. We show the result



(a) BFS (b) HOTSPOT (c) KMEANS

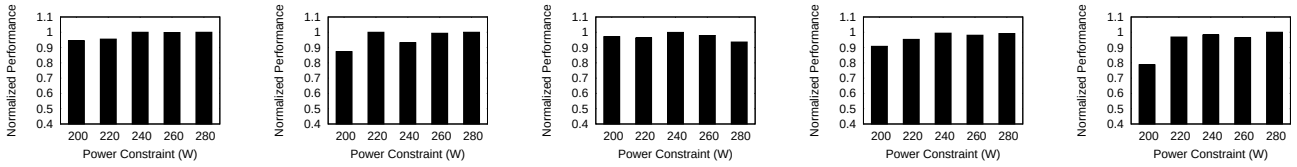
(d) PF (e) SGEMM

Figure 8. Normalized performance under the power constraint. (normalized to the ideal performance for each power constraint.)



(a) BFS (b) HOTSPOT (c) KMEANS (d) PF (e) SGEMM

Figure 9. Performance normalized to the maximum performance without power constraint.



(a) BFS (b) HOTSPOT (c) KMEANS (d) PF (e) SGEMM

Figure 10. Normalized performance with the profiles of different data size. (normalized to the ideal performance for each power constraint.)

in Figure 10. The x-axis and y-axis are the same in Figure 8. Due to the additional errors in profile data, in this case, we have to add 6W in the power estimation as a margin in order to run applications under the power constraint. Figure 10 shows that the proposed technique can achieve near optimal performance even when the size of input data is different from the data used in the profiling runs.

VI. Related Work

Power capping techniques have been proposed for various layers of computing systems [7], [11]. Fan et al [7] evaluates the effectiveness of cluster level power capping with commercial workloads in data centers. David [6] have proposed a power capping technique for memory modules, and Kai et al [13] have proposed a power capping technique for many-core CPUs running mixed groups of multi-threaded applications. However, no previous work has investigated a power capping for CPU-GPU heterogeneous systems.

A lot of researchers have proposed compilers or runtime systems to schedule data parallel tasks among CPUs and GPUs in order to maximize the performance of GPGPU applications [8], [9], [12], [17], [18]. Luk et al. [12] have proposed a dynamic compiler for CPU-GPU heterogeneous systems. The compiler maps parallel tasks among CPUs and GPUs based on an empirical performance model. Scogland et al. [18] have integrated heterogeneous task scheduling into OpenMP compilers with accelerator extensions. Grewe et al. [9] have proposed a static technique based on machine learning in order to determine the optimal percentages of CPU and GPU tasks. On the other hand, several domain specific platforms for GPUs adopt the capability to schedule parallel tasks among CPUs and GPUs [8], [17]. Since these

techniques do not consider power consumption of CPU-GPU hybrid executions, we cannot directly apply these techniques to the system under the power constraint. GreenGPU [14] is a system which can apply both DVFS and task mapping between CPUs and GPUs in order to maximize the energy efficiency. However, they use DVFS and task mapping individually, so their method cannot set optimal parameters of DVFS and task mapping. Plus, their technique cannot be used as a power capping technique because they do not provide any mechanism to keep a power constraint.

VII. Conclusion and Future Work

In this paper, we propose an efficient power capping technique through coordinating DVFS and task mapping in a CPU-GPU heterogeneous system. To avoid the power violation and the load imbalance between the CPUs and the GPUs, we build empirical models to predict the execution time and the power consumption of a heterogeneous system. With a small number of profiles and the model, the proposed technique enables us to determine the optimal set of device frequencies and task mapping at the beginning of the execution. The experimental result shows that the proposed power capping technique can achieve more than 93% of performance compared to the ideal one in 24 cases out of the 25 cases. Our future work is to extend the proposal for on-chip CPU-GPU architectures and multi-node heterogeneous systems, and to extend the proposal for the environment where multiple parallel applications run.

Acknowledgment

This work was supported by JST, CREST and the Grant-in-Aid for JSPS Fellow (23 8062).

References

[1] Amazon EC2. http://aws.amazon.com/ec2/.

[2] OpenACC Home. http://www.openacc-standard.org/.

[3] OpenCL - The open standard for parallel programming of heterogeneous systems. http://www.khronos.org/opencl/.

[4] watts Up Pro. https://www.wattsupmeters.com/secure/products.php?pn=0.

[5] S. Che, M. Boyer, J. Meng, D. Tarjan, J. Sheaffer, S.-H. Lee, and K. Skadron. Rodinia: A benchmark suite for heterogeneous computing. In *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC), 2009.*, pages 44–54, 2009.

[6] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le. RAPL: Memory power estimation and capping. In *Proceedings of the ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED), 2010*, pages 189–194, 2010.

[7] X. Fan, W.-D. Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. *SIGARCH Comput. Archit. News*, 35(2):13–23, June 2007.

[8] A. Gharaibeh, L. Beltrão Costa, E. Santos-Neto, and M. Ripeanu. A yoke of oxen and a thousand chickens for heavy lifting graph processing. In *Proceedings of the 21st international conference on Parallel architectures and compilation techniques*, PACT '12, pages 345–354, 2012.

[9] D. Grewe and M. F. P. O'Boyle. A static task partitioning approach for heterogeneous systems using OpenCL. In *Proceedings of the 20th international conference on compiler construction*, CC'11/ETAPS'11, pages 286–305, 2011.

[10] C. Isci, A. Buyuktosunoglu, C.-Y. Chen, P. Bose, and M. Martonosi. An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO'06*, pages 347–358, 2006.

[11] C. Lefurgy, X. Wang, and M. Ware. Power capping: a prelude to power shifting. *Cluster Computing*, 11(2):183–195, 2008.

[12] C.-K. Luk, S. Hong, and H. Kim. Qilin: exploiting parallelism on heterogeneous multiprocessors with adaptive mapping. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 42, pages 45–55, 2009.

[13] K. Ma, X. Li, M. Chen, and X. Wang. Scalable power control for many-core architectures running multi-threaded applications. In *Proceedings of the 38th annual international symposium on Computer architecture*, ISCA '11, pages 449–460, 2011.

[14] K. Ma, X. Li, W. Chen, C. Zhang, and X. Wang. GreenGPU: A Holistic Approach to Energy Efficiency in GPU-CPU Heterogeneous Architectures. In *Proceedings of the 41st International Conference on Parallel Processing (ICPP)*, pages 48–57, 2012.

[15] W. mei W. Hwu. *GPU Computing Gems Emerald Edition*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2011.

[16] Y. Ogata, T. Endo, N. Maruyama, and S. Matsuoka. An efficient, model-based CPU-GPU heterogeneous FFT library. In *Proceedings of the 22th IEEE International Parallel and Distributed Processing Symposium(IPDPS), 2008.*, pages 1–10, 2008.

[17] V. T. Ravi, W. Ma, D. Chiu, and G. Agrawal. Compiler and runtime support for enabling generalized reduction computations on heterogeneous parallel configurations. In *Proceedings of the 24th ACM International Conference on Supercomputing*, ICS '10, pages 137–146, 2010.

[18] T. Scogland, B. Rountree, W. chun Feng, and B. de Supinski. Heterogeneous Task Scheduling for Accelerated OpenMP. In *Proceedings of the IEEE 26th International Parallel Distributed Processing Symposium (IPDPS), 2012*, pages 144 –155, may 2012.

[19] J. Vetter, R. Glassbrook, J. Dongarra, K. Schwan, B. Loftis, S. McNally, J. Meredith, J. Rogers, P. Roth, K. Spafford, and S. Yalamanchili. Keeneland: Bringing Heterogeneous GPU Computing to the Computational Science Community. *Computing in Science Engineering*, 13(5):90–95, 2011.