Power Capping of CPU-GPU Heterogeneous Systems using Power and Performance Models

Kazuki Tsuzuku and Toshio Endo
Tokyo Institute of Technology, Tokyo, Japan

Keywords: GPGPU, Power Capping, DVFS, Power Model, Performance Model.

Abstract: Recent high performance computing (HPC) systems and supercomputers are built under strict power budgets and the limitation will be even severer. Thus power control is becoming more important, especially on the systems with accelerators such as GPUs, whose power consumption changes largely according to the characteristics of application programs. In this paper, we propose an efficient power capping technique for compute nodes with accelerators that supports dynamic voltage frequency scaling (DVFS). We adopt a hybrid approach that consists of a static method and a dynamic method. By using a static method based on our power and performance model, we obtain optimal frequencies of GPUs and CPUs for the given application. Additionally, while the application is running, we adjust GPU frequency dynamically based on real-time power consumption. Through the performance evaluation on a compute node with a NVIDIA GPU, we demonstrate that our hybrid method successfully control the power consumption under a given power constraint better than simple methods, without aggravating energy-to-solution.

1 INTRODUCTION

The issue of power/energy consumption of HPC systems and supercomputers has been and will be an important research topic. For example, Tianhe-2[1], the current fastest supercomputer with performance of 33.86 PFLOPS consumes the power of 17.6 MW during Linpack benchmark. A realistic power budget for an exascale system is considered as 20 MW, which requires an energy efficiency of 50 GFLOPS/W(Lucas, 2014). Therefore, exascale systems, which are expected to appear around 2020, require 25 times higher energy efficiency than the current fastest supercomputer.

Recently, in order to improve energy efficiency of HPC systems, accelerators including GPUs or Xeon Phi have been attracted attention. For example, Tianhe-2 and TSUBAME supercomputer in Tokyo Institute of Technology(Matsuoka, 2011) are equipped with accelerators, in addition to general purpose CPUs. In spite of better efficiency, however, the fluctuation of power consumption tend to be larger with accelerators. While keeping better efficiency, the peak power consumption of the system should be capped by the power budget determined by the building or organization.

This paper describes an efficient power capping method for computing nodes with accelerators, assuming the existence of *dynamic voltage frequency scaling* (DVFS) mechanism of modern CPUs and accelerators. Here we should note that a naive usage of DVFS may degrade the application performance and sometimes harmful for energy optimization. Instead, our goal is to minimize energy consumption during the application run, while conforming a given power constraint. Towards this goal, there are several issues:

- Most of modern PCs and servers already use DVFS for power saving by controlling the frequency according to the node load. Also some advanced servers support power capping by comparing the realtime power consumption and the cap value (Gandhi et al., 2009). However, it is unclear whether this method can optimize energy efficiency of HPC systems, where the average load is much higher than client PCs.
- For energy optimization, we need to take the application characteristics into account, such as GPU/CPU usage, arithmetic intensity and memory access frequency.
- Recent Intel CPUs are equipped with Running Average Power Limit (RAPL) technique, which

[1] http://www.top500.org

Tsuzuku K. and Endo T..
Power Capping of CPU-GPU Heterogeneous Systems using Power and Performance Models.
DOI: 10.5220/0005445102260233
In *Proceedings of the 4th International Conference on Smart Cities and Green ICT Systems* (SMARTGREENS-2015), pages 226-233
ISBN: 978-989-758-105-2

supports not only DVFS but power capping of CPUs. However, RAPL itself does not cap the power consumption of the entire node.

Towards the above-mentioned goal, this paper proposes a *hybrid* power capping method of a static method and a dynamic method. When the application starts, we determine the initial frequency *statically* based on our power and performance model. Then during the application is running, we *dynamically* change the frequencies based on monitored power consumption. This dynamic phase is introduced to recover the excess of power, which may be caused by model errors. Through the experiments using a compute node with a NVIDIA GPU, we demonstrate that neither the static approach nor the dynamic approach can satisfy the goal solely, and combining the two is essential.

2 BACKGROUND

The control of power consumption of supercomputers, which may reach the order of megawatts, is becoming an important issue towards the protection of the environment and cost reduction for energy.

Power capping technique is even more important, especially for the systems with accelerators, whose power fluctuation is larger. Towards power capping for systems, this paper focus on power capping and energy saving on a compute node equipped with CPUs and GPUs.

In previous HPC systems, it was more difficult to obtain power consumption of each node due to lack of smart power sensors or meters. Thus in order to estimate power consumption of applications on such nodes, statistical power models based on performance counters have been constructed(Nagasaka et al., 2010).

More recently, detailed monitoring of node power consumption is much easier due to spread of power sensors in computer systems. The real time power consumption can be obtained with interfaces such as RAPL for Intel CPUs and NVIDIA Management Library (NVML) for NVIDIA GPUs. Such interfaces are used both for power monitoring and control; the GPU clock speed can be configured by using the NVML library.

Not only for power in processor level, with the spread of inexpensive smart meters, HPC community has started to develop the specification of power monitor/control API of HPC systems(Laros, 2014). As an example of a working system, TSUBAME-KFC(Endo et al., 2014), ranked as No.1 in the world in the GREEN500 List[2], has a detailed monitoring system, which can monitor not only CPUs and GPUs power but also AC power of each node by intervals of a second.

In response to the spread of monitoring systems, we design our power capping method that uses real time monitoring as described in the next section.

3 PROPOSED POWER CAPPING METHODS

In this section, we discuss two simple power-capping methods, a dynamic method and a static method. And then we combine them into a hybrid method. Our goal is to keep power consumption lower than a given power budget during the execution of user applications. Generally it is hard to avoid instantaneous excess of power; instead, we minimize the duration when the power consumption is exceeding the limit (hereafter, *excess duration*). Our goal also includes the optimization of energy consumption, while keeping the excess duration minimum.

We currently focus on power capping of a single node equipped with an NVIDIA GPU accelerator. Our power capping method is designed to support applications that have various characteristics, however, we assume that a single application is running on a node at a time. Also currently we have the following assumptions on the application; the application uses mainly GPUs for its computation, and a single CPU core is mainly used for initialization of the application, controlling the GPU. The application uses GPU kernel functions that have similar characteristics to each other; thus GPU power consumptions when kernel functions are running do not change drastically.

3.1 Dynamic Power Capping Method

Here we describe the power capping technique using dynamic changing of clock speeds of GPUs. With this method, we continuously monitor power consumption of the node, and if power consumption reaches or exceeds the power limit, we decrease GPU clock speeds. If the power consumption is much lower than the limit contrarily, we increase the speed.

While this dynamic method is simple, we still have to take care of control amount and control interval. As for the control amount, we adopt a simple method to change the clock by only a single step once. As for the interval of control, there is a tradeoff; if the interval is too long, we may miss the sudden increase

[2]http://www.green500.org

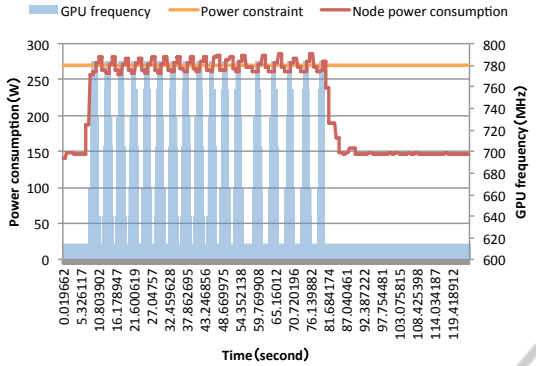


Figure 1: Change of the power consumption when the control interval is 0.2 seconds.

of power consumption and extend the excess duration. On the other hand, too short interval may lead vibration of clock speeds as follows.

For the experiments, we have implemented a daemon program that continuously monitors the node power and changes the GPU clock speed as described. Figure1 shows the results when the control interval is 0.2 seconds. The graph shows changes of the GPU clock speed and power consumption of the node. We observe a fast and large (the clock varies from minimum speed to maximum speed) fluctuation of the clock speed, which leads a frequent excess of the power consumption. The reason of this phenomena is as follows; due to the delay of power sensor, the control daemon observes old power values. If the clock interval is shorter than the delay, the daemon changes the clock speed before the previous change affects power consumption. As a result, excess duration gets larger.

During preliminary experiments, we adopted 5.0 seconds as the control frequency. However, this is fairly long, and may incur slow adaptation to an appropriate clock. Thus initial setting of clock speed is important for our objectives.

3.2 Static Power Capping Method

This section explains a model-based power capping method, which determines appropriate CPUs and GPUs clock speeds based on a power and performance model. We model relationship of clock speeds and the power consumption and execution time for given applications. Using the model, we can determine the clock speeds that achieve our goals as follows. We compute the power consumption and energy consumption during the execution of the given application execution for all combinations of GPU clocks and CPU clocks. Among the results of all combinations, we select the best clocks, which does not cause

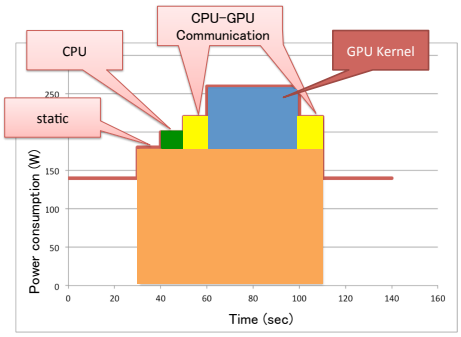


Figure 2: Our model of power consumption during GPU application execution.

exceeding the power limit, while minimizing the energy consumption.

3.2.1 Modeling Execution of GPU Applications

In our model, the execution of a GPU application is divided into the following categories as shown in Figure 2: the duration when GPU kernels are running, the duration when CPU are running, the duration for communication between CPU and GPU, and the duration when both CPU and GPU are idle. Thus E, the energy consumption of a node during application execution is represented by the following equation.

$$E = P_{static} * T_{all} + P_{Comm} * T_{Comm} + P_{CPU} * T_{CPU} + P_{GPU} * T_{GPU} \quad (1)$$

where variables are explained in Table 1.

For simplicity, our current model depends on the following assumptions. First, the GPU kernels in an application are uniform and the power consumption P_{GPU} is constant if the clock speed is constant. Similarly, P_{CPU} is constant. Secondly, GPU kernels, CPU computation and communication between CPU and GPU do not overlap with each other.

Table 1: variables of equation.

P_{static}	Static node power consumption
P_{Comm}	Increased node power consumption for communication between CPU and GPU
P_{CPU}	Increased node power consumption for CPUs
P_{GPU}	Increased node power consumption for GPU kernels
T_{all}	Overall execution time
T_{Comm}	Total execution time of communication
T_{CPU}	Total execution time of CPU
T_{GPU}	Total execution time of GPU kernels

The values of variables in Table 1 are affected by characteristics of applications, characteristics of ar-

chitecture, and CPU/GPU clock speeds. Thus we construct a model equation for each variable including these factors. In the following explanation, we pick up several variables.

3.2.2 Power Model

Here we mainly pick up P_{GPU}, GPU power consumption while GPU kernels are running, which accounts for a large percentage of overall power consumption. We model P_{GPU} as follows.

$$P_{GPU}(f) = \alpha_{kernel} * A_{GPU} * V(f)^2 * f + \beta_{kernel} * B_{GPU} + C_{GPU} * V(f) \quad (2)$$

The model of P_{GPU} consists of three terms. The first corresponds to the power consumed by the GPU cores for computation. It is proportional to the GPU clock frequency f and to the square of the voltage $V(f)$. Since the voltage is automatically determined by NVML according to f [3], we let $V(f)$ be a function of f. The power consumption of cores is also affected both by characteristics of GPU architecture and those of kernel functions of the application. In order to express them, we introduce an architecture parameter A_{GPU} and an application parameter α_{kernel}, where $0 \leq \alpha_{kernel} \leq 1$. Intuitively, A_{GPU} corresponds to capacitance of GPU cores and α_{kernel} corresponds to compute intensity of the kernel.

The second term of the equation corresponds to the power consumed by the GPU device memory, and is derived from an architecture parameter B_{GPU} and an application parameter β_{kernel}, where $0 \leq \beta_{kernel} \leq 1$. Although it would be more precise if it included the frequency and the voltage of device memory, the current GPUs do not provide good control ways for them. Thus we assume they are fixed and already reflected into B_{GPU}. β_{kernel} corresponds to the memory access frequency, and larger β_{kernel} represents memory intensive kernels.

The third term corresponds the static power consumption, and we currently assume it is proportional to the core voltage $V(f)$.

This model is used as follows. As the first step, when the target GPU architecture for modeling is fixed, the architecture parameters, $V(f), A_{GPU}, B_{GPU}, C_{GPU}$ should be obtained. In order to determine $V(f)$, we execute a simple and extremely memory intensive benchmark (we assume it has parameters of $\alpha_{kernel} = 0, \beta_{kernel} = 1$), and measure GPU power consumption P_{GPU} for every supported clock. Since we assume the first term is zero and the second term is constant, we can obtain C_{GPU} and $V(f)$ for every f[4]. Also by observing the constant factor of the power, we can obtain B_{GPU}. Similarly, by measuring GPU power with a highly compute intensive kernel, with $\alpha_{kernel} = 1, \beta_{kernel} = 0$, we can derive A_{GPU}. Now we have obtained architecture parameters for the given GPU architecture.

The second step is to obtain application parameters. In order to support various types of applications, this step should use simpler methods than in the first step; we execute test-runs and power measurements twice per application as follows. For a given application, we execute the first test-run at the maximum GPU clock to obtain GPU power consumption $P_{GPU}(f_{max})$. Also the second test-run is done at the minimum GPU clock to obtain $P_{GPU}(f_{min})$. With these values and architecture parameters obtained above, we can calculate α and β by solving simple simultaneous equations.

Using architecture parameters and application parameters, now we can estimate the GPU power consumption at arbitrary GPU clock speeds.

With similar discussion, we estimate P_{CPU} at arbitrary CPU clock speeds. As for other power parameters P_{static} and P_{Comm}, we assume that they depend only on architecture, and are independent from application characteristics and clock speeds. Thus we obtain them by preliminary power measurement.

3.2.3 Performance Model

Here we pick up $T_{GPU}(f)$, the total execution time of GPU kernels during the application execution. Our estimation is based on the measured values in preliminary measurements; we obtain $T_{GPU}(f_{min})$ at the minimum clock, and $T_{GPU}(f_{max})$ at the maximum clock. From these two values, we estimate $T_{GPU}(f)$ for arbitrary f as follows [5].

$$T_{GPU}(f) = max(T_{GPU}(f_{min}) * f_{min}/f, T_{GPU}(f_{max})) \quad (3)$$

Similarly, we estimate T_{CPU} at arbitrary CPU clock speeds. T_{Comm}, which is roughly independent from clock speeds, is obtained during the preliminary measurement of the application.

3.3 Hybrid Power Capping Methods

For the dynamic method in Section 3.1, we observed that a longer control frequency is better for stable control. With this choice, however, it may take a long

[3]The relationship between f and voltage on NVIDIA GPUs is not open information

[4]Here we assume that $V(f_{min}) = 1$ without loss of generality.

[5]we omit the detail for this for want of space

time to settle at the appropriate clock after application execution starts. On the other hand, the static method can find the appropriate clocks before execution, however, it may suffer from errors in the model or fluctuation of power consumption.

Based on this discussion, we propose to combine the two. Before the application execution, we determine the initial GPU/CPU clock speeds based on the static method. During the execution, we continuously controls the clock speeds by using the dynamic method.

We describe two variants of the hybrid method, whose difference appears in the usage of the dynamic method.

Hybrid1. During the execution, we simply use the dynamic method.

Hybrid2. During the execution, we also control the clock speed dynamically, however, it does not exceed the initial clock obtained using the model.

The performance of this hybrid method is discussed in the following section in detail.

4 Evaluation

4.1 Methodology

In the experiment, we apply the proposed power capping methods while a GPU application is running under a given power constraint. We evaluate average power consumption of the node, excess duration and the energy consumption among the execution. We use a computing node with two CPUs and two GPUs (Table 2), though each application uses a single CPU core and a GPU.

Our power capping method has been implemented in a daemon process, which continuously monitors power consumption and configures GPU/CPU clock speeds. We measure the real time power consumption of the node with "OMRON RC3008", which is a portable power distribution unit equipped with smart power meters. While we measure the power consumption at every 15 miliseconds, we control the clock speeds per 5.0 seconds.

We evaluate three application benchmarks shown in Table 3. We compare power capping methods shown in Table 4. The hybrid method includes two variants described in the previous section. In some experiments, we also show the cases without power control, "static max" and "static min".

Table 2: CPUs and GPUs in the compute node for evaluation.

CPU	Intel(R) Xeon(R) CPU E5-2660
GPU	NVIDIA K20Xm
CPU Frequencies(MHz)	1200,1300,1400,1500 1600,1700,1800,1900 2000,2100,2200,2201
GPU Frequencies(MHz)	614,640,666,705 732,758,784

Table 3: GPU application benchmarks.

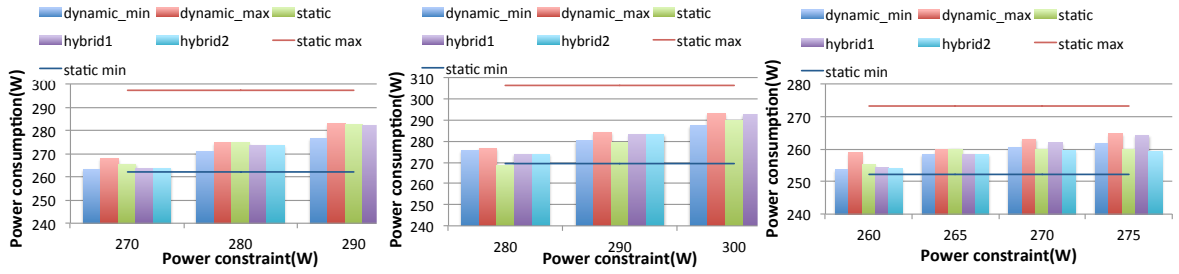
diffusion	thermal diffusion simulation
matmul	matrix multiplication
gpustream	stream benchmark, measuring memory bandwidth for GPU

4.2 Evaluation of Power Consumption and Excess Duration

Figure 3 shows average power consumption during execution of each application. We observe the average power consumption is under the constraint in all cases.

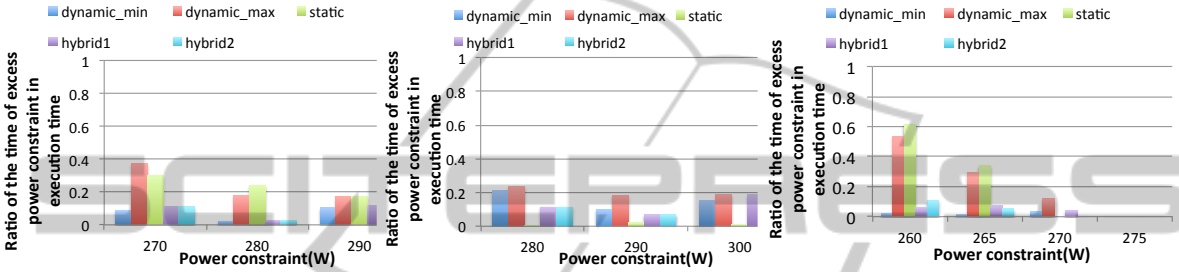
Not only capping average consumption, also restricting power fluctuation is important. Figure 4 shows excess durations, the durations when the power consumption is exceeding the given constraint. In the graphs, the durations are normalized to the application execution time. We observe that excess durations tend to be larger with the static method and the dynamic max, especially in the case of gpustream with the constraint of 260W.

In order to analyze this case, we show the dynamic change of power and clock speeds in figure 5. In the graph of "dynamic max", we see that the application starts with the maximum clock, which leads the excess of power, and it takes about 30 seconds until the clock becomes optimal setting (the minimum clock in this case). Thus the excess duration gets larger. With "static", the model suggested 640MHz as the appropriate clock. However, we see that the actual power consumption sometimes exceeds the constraint, and the excess duration reaches about 30% due to the lack of feedback. On the other hand, with "hybrid" method, we can reduce the excess duration. Compared with "dynamic max", the excess duration is even shorter, since the initial clock, which has been determined with the model, is closer to the appropriate clock than the maximum clock is.



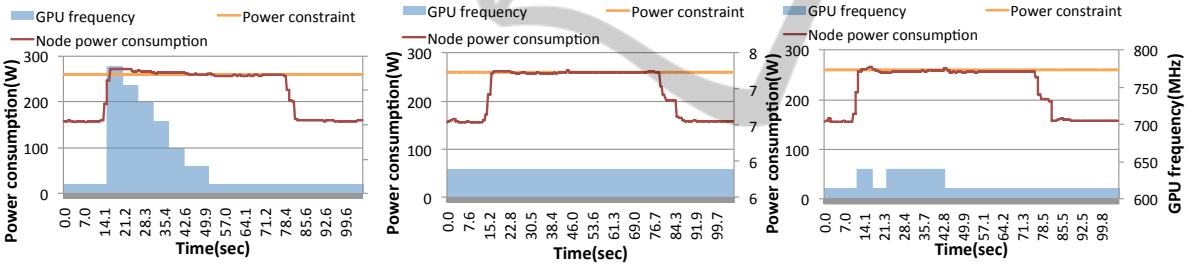
[1]diffusion [2]matmul [3]gpustream

Figure 3: Average power consumption for each power constraint.



[1]diffusion [2]matmul [3]gpustream

Figure 4: Time of excess power constraint.



[1]dynamic max [2]static [3]hybrid1

Figure 5: Change of the power consumption on gpustream with each power capping technique.

4.3 Evaluation of Energy Consumption

When power capping is achieved well, it is also desirable to restrict the energy consumption during the application execution. Figure 6 compares energy consumption of GPU applications.

With diffusion and matmul applications, which are compute-intensive, we observe that energy consumption tends to be smaller with the power constraint gets larger (relaxed). This indicates that the improvement of speed performance outweighs the increase of power consumption. The selection of the power capping method affects the energy for less than 3%.

On the other hand, the memory intensive application, gpustream, demonstrates a different tendency. Changing the power constraint from 270W to 275W increases the energy consumption, if we adopt dynamic or hybrid1 methods. This is due to the following reason. The speed performance of memory intensive programs is hardly affected by the clock speed. Thus using lower clock is not harmful for such programs. However, when dynamic and hybrid1 methods notice that the current power consumption is much lower than power constraint, they continuously raise the clock, which does not improve performance. On the other hand, the hybrid2 method successfully avoids this useless rise of clock by introducing the upper bound of the clock.

5 RELATED WORK

DVFS has been the most popular method for power control not only for CPUs but GPUs. The effects of DVFS on power consumption and performance of

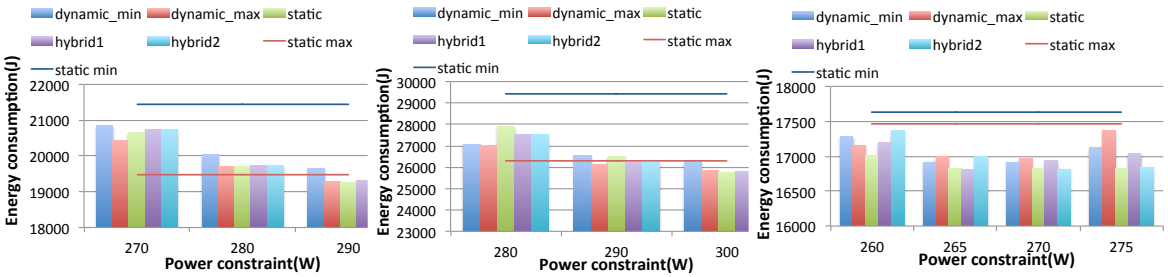


Figure 6: Energy consumption.

Table 4: Compared power capping methods.

dynamic min	The dynamic method. Initial frequency is the minimum one
dynamic max	The dynamic method. Initial frequency is the maximum one
static	The static method. The frequency is determined by the model
hybrid1	The hybrid method. Initial frequency is determined by the model
hybrid2	The hybrid method. Initial frequency and configurable maximum frequency are determined by model
static max	The frequency is fixed at the maximum one
static min	The frequency is fixed at the minimum one

GPUs have been studied (Jiao et al., 2010; Mei et al., 2013). Recently Burschen et al (Burtscher et al., 2014) have investigated one of issues in controlling current GPUs, the time lag of NVML, and proposed a technique to recreate power consumption correctly. While these efforts do not include algorithms to control power consumption directly, we will improve our models by harnessing their knowledge.

There are several projects for estimating power consumption of computer nodes. Some of them have proposed power model of GPU kernels, which give the relation between power consumption and performance counters(Nagasaka et al., 2010; Song et al., 2013). On the other hand, our hybrid method assumes that real time power monitoring is commodity; we can feedback the measured values.

Komoda et al. (Komoda et al., 2013) have proposed a power capping technique using both DVFS and task mapping to cap the power consumption of the systems with GPUs. Unlike our approach, they assume that users describe applications to enable to change their load balance between CPU and GPU. In contrast, our focus is to control power consumption during execution of arbitrary GPU applications.

Shröne et al. have discussed measurement and analysis methods for the energy efficiency of HPC systems(Schöne et al., 2014). Their discussion includes using power model in order to find a CPU clock frequency that is optimal for better energy efficiency. Our static method has a similar purpose, however, we combine a dynamic method and a static method in order to alleviate the effect of errors in the model. Our future plan includes an extension to the whole system, by combining our method with power aware job scheduling, as they suggest.

6 CONCLUSION AND FUTURE WORK

We have proposed an efficient power capping method for compute nodes equipped with accelerators. To avoid the increase of energy consumption under a given power constraint, we adopt a hybrid approach of a model-based static method and a dynamic controlling method. We have shown that the proposed technique successfully reduces the excess of power consumption.

Our future work includes extension of our method for more CPU intensive applications, or applications that consist of several phases, each of which shows different characteristics in power and performance. Also we will extend the methods to the whole HPC system towards the next-gen energy efficient exascale systems.

ACKNOWLEDGEMENTS

This research is mainly supported by Japanese

MEXT, "Ultra Green Supercomputing and Cloud Infrastructure Technology Advancement" and JST-CREST, "Software Technology that Deals with Deeper Memory Hierarchy in Post-petascale Era".

REFERENCES

Burtscher, M., Zecena, I., and Zong, Z. (2014). Measuring gpu power with the k20 built-in sensor. In *Proceedings of Workshop on General Purpose Processing Using GPUs*, pages 28:28–28:;36. ACM.

Endo, T., Nukada, A., and Matsuoka, S. (2014). Tsubame-kfc : a modern liquid submersion cooling prototype towards exascale becoming the greenest supercomputer in the world. In ICPADS2014, editor, *The 20th IEEE International Conference on Parallel and Distributed Systems*.

Gandhi, A., Harchol-Balter, M., Das, R., and Lefurgy, C. (2009). Optimal power allocation in server farms. In *Proceedings of the Eleventh International Joint Conference on Measurement and Modeling of Computer Systems*, pages 157–168. ACM.

Jiao, Y., Lin, H., Balaji, P., and W., F. (2010). Power and performance characterization of computational kernels on the gpu. In *Proceedings of the 2010 IEEE/ACM Int'L Conference on Green Computing and Communications & Int'L Conference on Cyber, Physical and Social Computing*, pages 221–228. IEEE.

Komoda, T., Hayashi, S., Nakada, T., Miwa, S., and Nakamura, H. (2013). Power capping of cpu-gpu heterogeneous systems through coordinating dvfs and task mapping. In *Computer Design (ICCD), IEEE 31st International Conference*, pages 349–356. IEEE.

Laros, J., e. a. (2014). High performance computing - power aoolication programming interface specification version 1.0. In *SANDIA REPORT*, volume SAND2014-17061.

Lucas, R. e. a. (2014). Top ten exascale research challenges. In *DOE ASCAC Subcommitte Report*.

Matsuoka, S. (2011). Tsubame 2.0 begins - the long road from tsubame1.0 to 2.0(part two). In e Science Journal, editor, *Global Scientific Information and Computing Center*, volume 3, pages 2–8. Tokyo Institute of Technology.

Mei, X., Yung, L. S., Zhao, K., and Chu, X. (2013). A measurement study of gpu dvfs on energy conservation. In *Proceedings of the Workshop on Power-Aware Computing and Systems*, pages 10:1–10:5. ACM.

Nagasaka, H., Maruyama, N., Nukada, A., Endo, T., and Matsuoka, S. (2010). Statistical power modeling of gpu kernels using performance counters. In *Green Computing Conference, 2010 International*, pages 115–122. IEEE.

Patki, T., Lowenthal, D. K., Rountree, B., Schulz, M., and de Supinski, B. R. (2013). Exploring hardware overprovisioning in power-constrained, high performance computing. In *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing*, ICS'13, pages 173–182. ACM.

Schöne, R., Treibig, J., Dolz, M. F., Guillen, C., Navarrete, C., Knobloch, M., and Rountree, B. (2014). Tools and methods for measuring and tuning the energy efficiency of hpc systems. In *Scientific Programming*, pages 273–283. IOS Press.

Song, S., Su, C., Rountree, B., and Cameron, K. (2013). A simplified and accurate model of power-performance efficiency on emergent gpu architectures. In *Parallel & Distributed Processing (IPDPS)*, pages 673–686. IEEE.