Energy-centric DVFS Controling Method for Multi-core Platforms

Shin-gyu Kim, Chanho Choi, Hyeonsang Eom, Heon Y. Yeom
School of Computer Science and Engineering
Seoul National University, Korea
Email: {sgkim, chchoi, hseom, yeom}@dcslab.snu.ac.kr

Huichung Byun
Samsung Electronics, Korea
kw.byun@samsung.com

***Abstract*—Saving data center energy consumption is a hot issue for the environment and the economy. CPU is the biggest energy consuming component in a server, and it has various energy saving technologies, such as C-state and P-state. Advances in multi-core processors and virtualization technologies have enabled many workloads to be consolidated in a physical server, and energy efficiency can be degraded due to the contention in shared resources. We observed that energy efficiency can be improved further by adjusting CPU frequency according to the degree of contention in shared resources, and the most energy-efficient CPU frequency is quite different for each situation. In this paper, we propose an energy-centric DVFS controlling method (*eDVFS*), which aims to minimize total energy consumption. Our experimental results show that *eDVFS* method is more energy-efficient and faster than current "on-demand" CPU governor.**

I. Introduction

Today, reducing data center energy consumption is a critical issue for the environment and its operational cost. Environmental activist group Greenpeace reported that data centers in the US consumed close to 3% of the national power supply, and the global demand for electricity from data centers was close to the equivalent of the entire electricity demand of the UK in 2007. Moreover, this demand is projected to triple or quadruple by 2020. [1]

There has been a lot of research efforts to reduce energy consumption in many aspect of data centers. It includes reducing idle server power [2], server power budgeting [3], [4] and the use of renewable energy [5]. Actually, these work achieved significant improvements in general cases. Now, we focus on virtualized environments where multiple independent virtual machines (VMs) are consolidated in one physical server. The VM consolidation implies that multiple VMs contend for shared resources on each server. Although current virtualization technologies provide resource isolation, they do not ensure performance isolation. It is already well known that shared resource contention can result in performance degradation of VMs [6], [7], [8].

Most data center energy is consumed by computing servers, and CPU is the biggest energy consuming component in the server. Modern CPUs have various energy saving technologies, such as C-state, P-state and T-state. Especially, P-state changes operating frequency and voltage while instructions can be processed. It is also called a DVFS (Dynamic Voltage and Frequency Scaling) control. It is clear that CPU consumes less power[1] at low frequency than high frequency. However, total consumed energy of a workload can be reduced with high operating frequency, and the following graph shows this situation.

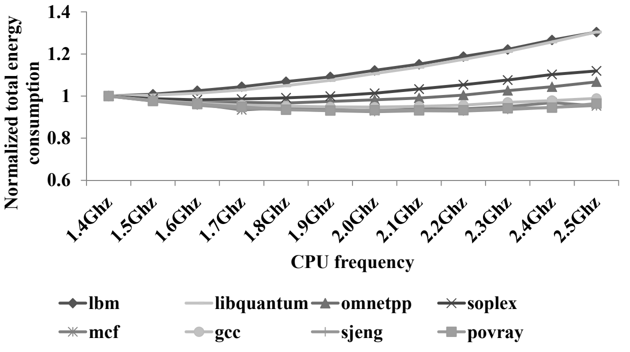


Fig. 1. Normalized total energy consumption of eight SPECcpu 2006 benchmark programs with varying CPU frequency. The CPU frequency for the lowest energy consumption is different from each other.

Figure 1 shows normalized total energy consumption of eight SPECcpu 2006 benchmarks at various CPU frequencies. This experiment is performed on a server with two Intel Sandybridge-based 8-core Xeon E5-2670 (2.6 GHz) processors and 128 GB memory. This CPU has 14 DVFS levels. The server runs Ubuntu 11.10 of Linux kernel 3.0.0 and KVM virtualization framework. There are 16 active VMs, and each VM runs one benchmark program in its entirety. Each VM is configured to have one virtual CPU with 2 GB memory, and is pinned to a dedicated CPU core. We measure system-wide total energy consumption while 16 VMs run the same benchmark concurrently. The energy consumption is calculated by multiplying average execution time and average consumed power (Watts). As shown in Figure 1, each benchmark shows a different energy consumption changing pattern with varying CPU frequency, and has a different CPU frequency for the lowest energy consumption. For example, `lbm` consumes more energy at higher CPU frequency and `povray` vice versa. The reason is contention in shared resources, such as last-level cache (LLC) and memory bus, among VMs. In the case of `lbm`, because total memory traffic is already close

[1]Power refers to the amount of energy (Joules) per 1 second.

the maximum bandwith of the memory bus even with the lowest CPU frequency, its runtime has little decrease as CPU frequency increases. In contrast, the runtime of CPU-intensive `povray` decreases in proportional to the increase of CPU frequency. Moreover, in the case of `lbm` and `libquantum`, the maximum difference of energy consumption is more than 30%. Nevertheless, the current "on-demand" CPU governor of Linux keeps CPU frequency high at all times, because the CPU utilization is always 100% for all benchmarks. Now, a new DVFS controlling method is needed to save energy.

In this work, we propose an energy-centric DVFS controlling method called "*eDVFS*". Through various experiments with SPECcpu 2006 benchmark, we found that high energy efficiency can be achieved with high utilization of shared resources, such as memory bus. High utilization means short runtime, and it can lead to lower total energy consumption when the shared resources are underloaded. On the other hand, if shared resources are overloaded, CPU frequency should be lowered to mitigate congestion. Our *eDVFS* method is based on these two observations, and experimental results show that *eDVFS* is more energy-efficient and faster than current "on-demand" CPU governor.

The rest of this paper is organized as follows: Section 2 explores energy efficiency of a multi-core platform with SPECcpu 2006 benchmark, and Section 3 introduces our *eDVFS* method. Section 4 presents experimental results and Section 5 concludes this paper.

II. Related Work

Energy-efficient processing on multi-core platforms is strongly related with mitigating contention in shared resources. Because the amount of consumed energy is the product of the average power and the execution time, reducing the execution time results in the low energy consumption. Mitigating shared resource contention in multi-core processors and virtualization environments have been studied by many researchers. Govindan et al. [6] and Mars et al. [7] proposed methodology to estimate performance degradation in virtualization environments. They profile performance degradation ratio with varying pressure in LLC by using synthetic cache load generator, and decide VM allocation based on the information. Although these techniques can achieve high efficiency, they cannot be adopted directly to the public cloud service due to the prior profiling phase before execution. Public cloud service providers do not know about customers' applications beforehand, and customers also do not want to report their work to the cloud providers. In constrast, our *eDVFS* method can dynamically adjust DVFS level in according to the contention status in shared resources.

As shown in the previous section, energy efficiency of memory-intensive workloads can be improved by lowering CPU frequency, and cpu-intensive workloads vice versa. Weissel et al. [9] and Choi et al. [10] proposed CPU frequency controlling technique based on workload decomposing technique into on-chip or off-chip workload with help of the performance monitoring unit (PMU). These work are different from our work in some points. First, the platforms used in [10] and [9] are built around the Intel's PXA255 and XScale processors for low-power, embedded system. Our platform is built upon Intel's Sandybridge-based 8-core Xeon processors for high performance server system. Moreover, we focus on the contention in shared resources rather than workload characterization. Dhiman et al. [11] presented power efficient sheduling method for multi-core platforms by using dynamic workload characterization, but they did not utilize the DVFS function. We et al. [12] presented dynamic compiler DVFS techniques, which optimize the application binary code and insert DVFS control instructions at program execution time. This technique can provide more fine-grained and code-aware DVFS control, but it requires user's applications to be re-compiled. Our *eDVFS* require no re-compilation and has very simple DVFS controlling policies.

III. Energy Efficiency of Multi-core Platform

A. Tested Multi-core Platform

We use the two-socket server in the experiment of Figure 1 throughout this paper. The server has two 8-core processors, eight 16 GB DDR3 DRAM modules and two SSDs. It is also equipped with three cooling fans, and a 1020 W power supply. Rotational speed of the cooling fans is fixed regardless of temperature. The tested processor provides energy usage information of each CPU and DRAM module. We use Intel Performance Counter Monitor [13] to get these per-component energy consumption, and Yokogawa WT-210 powermeter to measure system-wide total energy consumption. OS and virtualization environment are the same as the previous experiment.

The tested processor has its own memory controller, and the peak memory bandwidth from one thread and 32 threads are measured at about 7.2 GB/s and 68.4 GB/s respectively by LMbench. [14] We perform experiments to measure the memory bus interference between processors. Workloads are four memory-intensive benchmarks, which are `lbm`, `libquantum`, `omnetpp` and `mcf`. First, we configured 8 VMs to be located in one processor, and the other processor to be in an idle state. Each VM is pinned to a dedicated core, and all VMs run the same benchmark at the same time. We measure runtime of benchmarks for every available CPU frequency from 1.4 GHz to 2.5 GHz. Next, we allocate additional 8 VMs in the idle processor, and repeat the same experiments with 16 VMs. Figure 2 shows the results. In Figure 2, "8+0" means the configuration that only 8 VMs are in one processor, and "8+8" means that 16 VMs are in two processors. As shown in the graph, there is negligible difference between "8+0" and "8+8" in all cases. This means that there is little memory bus interference between the two processors. Because the tested processor allows only socket-wide DVFS control, the two processors can be viewed as two independent DVFS control regions.

B. Energy Consumption of Memory-intensive workloads

In this section, we analyze the energy consumption of memory-intensive workloads. In Figure 2, the runtimes of `lbm`

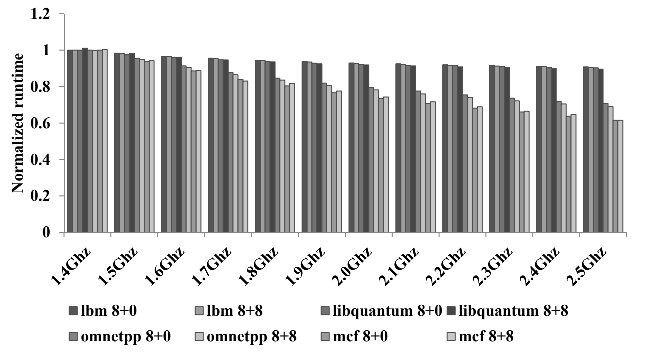


Fig. 2. Normalized average runtime of SPECcpu 2006 benchmark when 8 VMs are executed only in one processor (8+0) and 16 VMs in both processors (8+8).

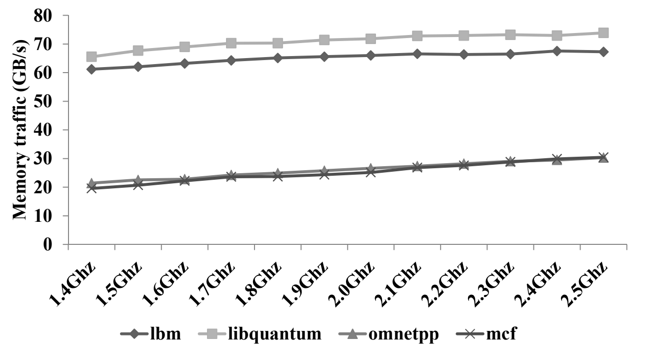


Fig. 3. Average memory bandwidth for memory-intensive benchmarks with varying CPU frequency.

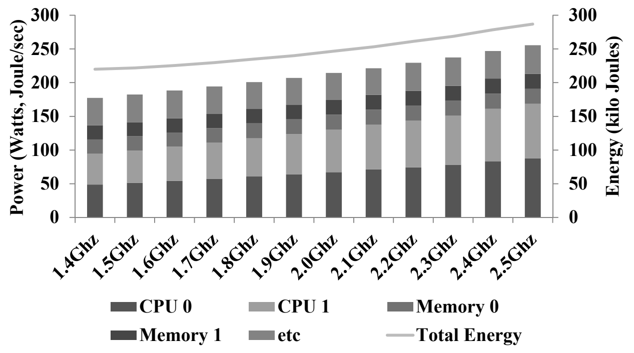


Fig. 4. Average power and energy consumption of memory-intensive `lbm` with varying CPU frequency.

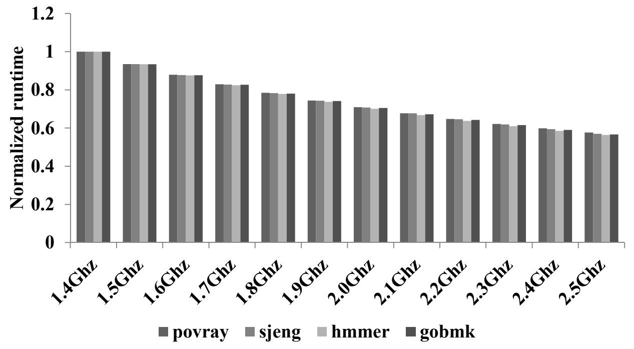


Fig. 5. Normalized average runtime of CPU-intensive benchmarks, `povray`, `sjeng`, `hmmer` and `gobmk`.

and `libquantum` are cut down by just 10% while CPU frequency is increased from 1.4 GHz to 2.5 GHz. However, the runtimes of `omnetpp` and `mcf` are dropped by 31% and 38%. The reason is the limited memory bandwidth, which is depicted in Figure 3. In the case of `lbm` and `libquantum`, average memory bandwidth is already close to the maximum memory bus capacity even at the lowest CPU frequency. Consequently, the memory bus becomes a performance bottleneck for the two benchmarks.

Figure 4 shows the average power consumption of `lbm` while it is executed in 16 VMs concurrently as like the "8+8" configuration in the previous experiment. "CPU 0" and "CPU 1" are the power consumed in each processor, and "Memory 0" and "Memory 1" are the power consumed in DRAM owned by each processor. "etc" includes the conversion loss of power supply, cooling fans, system board and so on. Unlike steady and high memory bandwidth of `lbm`, the total system power and CPU power grow in proportion to CPU frequency. The power consumed in DRAM has little change as like the memory bandwidth in all cases. We compare the memory power and bandwidth for all benchmarks in the later section.

We calculate the energy consumption of `lbm` by multiplying the average runtime with power. The result is depicted as a gray line in Figure 4. Because the runtime of `lbm` has a small change by increased CPU frequency, the total energy consumption is increased as the total power grows. Thus, the most energy-efficient CPU frequency for `lbm` is the lowest one. `libquantum`, which is as memory-intensive as `lbm`, also consumes less energy as the CPU frequency gets lowered. However, as shown in Figure 1, the most energy-efficient DVFS level is not the lowest one for `omnetpp` and `mcf`. We further explore the energy efficiency of the case where the memory bus is not fully utilized in the following section.

C. Energy Consumption of CPU-intensive Workloads

We consider a workload CPU-intensive if its average memory traffic is under 10% of memory bus capacity at all DVFS levels while 16 concurrent instances are running. CPU-intensive workloads includes `povray`, `sjeng`, `hmmer` and `gobmk` in SPECcpu 2006. Figure 5 shows runtime of these benchmarks with varying CPU frequency. In contrast to memory-intensive benchmarks in Figure 2, the runtime of CPU-intensive workloads is decreased as CPU frequency is increased.

Figure 6 shows the average power and energy consumption of `povray` with varying CPU freqeuncy. In this case, CPU consumes comparable power to that of `lbm`, but DRAM consumes much lower power than that of `lbm`. Note that the energy consumption pattern of `povray` is quite different to that of `lbm` with varying CPU frequency. The changing direction of total consumed energy is opposite to the total

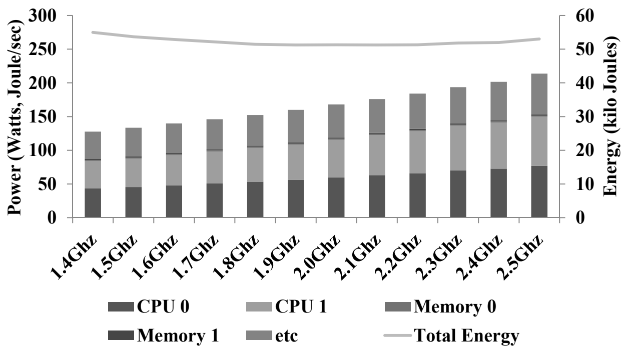


Fig. 6. Average power and energy consumption of CPU-intensive `povray` with varying CPU frequency.

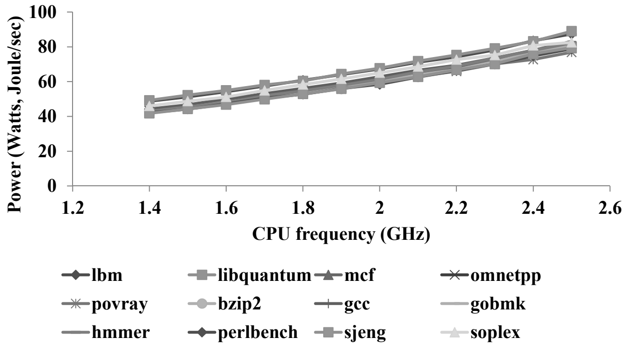


Fig. 7. CPU power consumption with varying CPU frequency.

power. The reason is that the ratio of decreased runtime is greater than the ratio of increased power. Therefore, if we can know power and task processing rate with varying DVFS level, total amount of energy consumption can be estimated.

D. Power Consumption vs. DVFS Level

Figure 7 shows the CPU power consumption of benchmarks with varying CPU frequency. As shown in the graph, CPU power is strongly related with CPU frequency, and the slope of all lines is almost the same regardless of workloads. This means that the change of CPU power can be estimated by just CPU frequency. We can also observe that memory-intensive workloads consume more power than CPU-intensive ones. However, we are not sure if it is related with cache or integrated memory controller.

Figure 8 shows the DRAM power consumption with varying memory traffic. The DRAM consumes more power as memory traffic increases, but the change of power is not large after memory traffic exceeds about 30 GB/s. Actually, DVFS does not have any direct control over DRAM power. But, memory traffic is related with processing speed, and processing speed is controlled by DVFS. Consequently, DVFS can control CPU power as well as DRAM power.

Memory traffic is generated by misses in the last-level cache (LLC). Because LLC is shared across all CPU cores, the number of cache misses can be increased by contention among consolidated workloads. There has been much work [6], [7], [15] to mitigate interference in LLC, and this work can be combined to improve energy efficiency of multi-core servers.

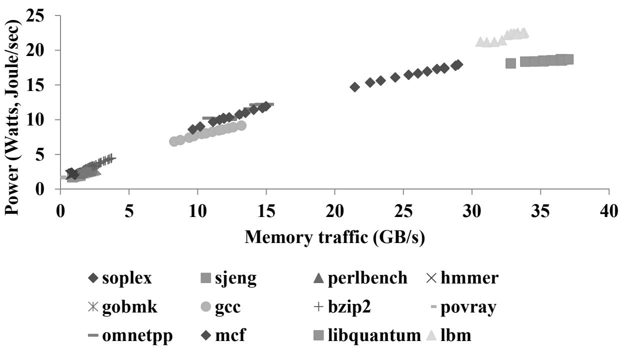


Fig. 8. DRAM power consumption with varying memory traffic.

IV. Workload-aware DVFS Control

A. Energy Efficiency Measurement

As mentioned before, the total amount of consumed energy (E) is calculated by multiplying power (P) and runtime (T) of a workload (W).

$$E = P \times T \tag{1}$$

The workload can be considered as a set of instructions, and we define a performance metric, "*Instructions Per Second* (IPS)". The widely used performance measure, "*Instructions Per Cycle* (IPC)", is not a good choice when CPU frequency is changing. Now, energy (E) can be rewritten as follows:

$$IPS = \frac{W}{T} \tag{2}$$

$$E = W \times \frac{P}{IPS} \tag{3}$$

Because workload (W) is an invariant term, the second term $\frac{P}{IPS}$ should be minimized to improve energy efficiency.

Energy consumption pattern of memory-intensive and CPU-intensive workloads can be explained using this equation. In the case of memory-intensive workloads, the change of T is relatively small compared to P. It means that P grows faster than IPS while CPU frequency increases. Thus, E is increased as CPU frequency gets higher. In contrast, in the case of CPU-intensive workloads, IPS grows faster than P while CPU frequency increases. In summary, DVFS level should be controlled to minimize $\frac{P}{IPS}$, and our *eDVFS* method utilizes this metric.

B. eDVFS

We propose a new DVFS controlling method *eDVFS* which only aims to minimize total energy consumption. *eDVFS* has two basic policies.

- **Policy 1:** If memory traffic exceeds a given threshold, decrease CPU frequency.
- **Policy 2:** If memory traffic is under a given threshold, move towards the direction in which $\frac{P}{IPS}$ is reduced.

eDVFS reads the number of instructions, memory traffic and consumed power from performance counters in CPU periodically. Applying policy 1 is straightforward, but not for policy 2. Estimating IPS for the changed CPU frequency is a very complex problem. To deal with this problem, *eDVFS* changes CPU frequency every control period. As shown in Figure 4 and 6, the line of total energy is a non-linear concave curve. The reason is that power is proportional to the square of the supply voltage as follows: [16]

$$P \propto fV^2 \tag{4}$$

where P is the consumed power, f is the operating frequency and V is the supply voltage. The voltage required for stable operation is determined by the frequency at which the CPU is clocked, and can be reduced if the frequency is also reduced. By using this fact, we can decide the moving direction by comparing the previous and current value of $\frac{P}{IPS}$. If the previous $\frac{P}{IPS}$ is smaller than the current one, CPU frequency is returned to the previous state. If not, CPU frequency moves one step further in the same direction.

We also need to decide the memory bandwidth threshold for the policies of *eDVFS*. There is little change in the memory access latency while the amount of memory traffic is increased up to a certain threshold. But if the amount of memory traffic exceeds the threshold, the main memory access latency sharply rises (Figure 5 in [17]). Therefore, we need to keep the total amount of memory traffic under smaller than the maximum capacity. This memory bandwidth threshold can be decided by an empirical way, we use 85% for our tested machine.

V. Experimental Results

We evaluate our *eDVFS* method with two experimental scenarios. The first scenario is for high memory traffic, and the second one is for low memory traffic. We measure energy consumption of CPU and DRAM, and compare it to the case when CPU frequency is fixed. Table I shows the list of benchmarks for the two experiments. Experimental environments are the same as the previous one, and the two groups of benchmarks are executed concurrently in each processor. All benchmark programs starts at the same time, and execution time of each one is listed in Table II. Note that the execution times of `lbm` and `libquantum` in case 2 are shorter than those of case 1 due to less contention in shared resources.

	Benchmarks
#1	2 × `lbm`, 2 × `libquantum`, `soplex`, `omnetpp`, `mcf`, `bzip2`
#2	`lbm`, `libquantum`, `gobmk`, `hmmer`, `perlbench`, `sjeng`, 2 × `povray`

TABLE I
List of benchmarks for the experiments. "#1" is for the high memory traffic, and "#2" is for the low memory traffic.

Figure 9 shows the varied memory traffic and CPU frequency with our *eDVFS* controller for the two cases. When total memory traffic is high, CPU frequency is lowered dynamically, and vice versa. In these experiments, *eDVFS* changes the DVFS level every 5 seconds, and the memory threshold for the policy 1 is configured as 85% of the peak bandwidth. As shown in the graph, the maximum memory traffic is kept under 32 GB/s (85 % of 38 GB/s). Figure 10 shows the sum of power consumed in CPU and memory. For the case 1 (Figure 10(a)), the changing behaviors of power and memory traffic are very similar. Because CPU frequency is kept at low, the variance of power is dominated by the power consumed in memory. For the case 2 (Figure 10(b)), the power is higher than case 1 and is not related with memory traffic. The reason is that CPU frequency is kept at high at most time. Total consumed energy and runtime are summarized in Table III. The static CPU frequency configurations refers to the current "on-demand" governor of Linux. *eDVFS* consumes less energy than static CPU frequency configurations in all cases. Moreover, the runtime with *eDVFS* is shorter than static 1.4 GHz and 1.9 GHz in the case 1 and 2 respectively. It means that our workload-aware dynamic DVFS controlling method achieved high energy-efficiency and good performance.

Case 1		Case 2	
lbm	2606	lbm	1861
lbm	2595	povray	901
libquantum	2438	libquantum	1801
libquantum	2436	povray	901
soplex	732	gobmk	802
omnetpp	1217	hmmer	774
mcf	1493	perlbench	666
bzip2	1504	sjeng	986

TABLE II
Execution time of all benchmarks for case 1 and 2.

	Static CPU frequency			*eDVFS*
	2.6GHz	1.9GHz	1.4GHz	
#1	193,156 J 2,138 sec	160,385 J 2,389 sec	156,889 J 2,856 sec	148,584 J 2,593 sec
#2	113,571 J 1,570 sec	103,799 J 1,945 sec	106,182 J 2,497 sec	102,184 J 1,830 sec

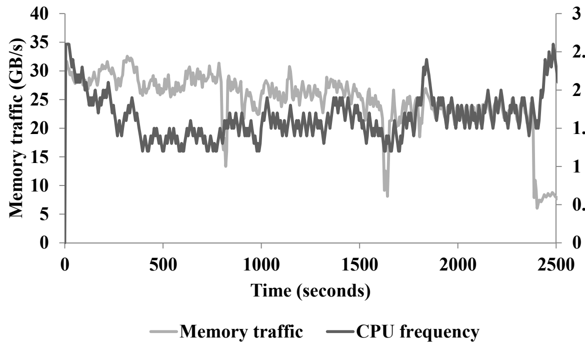
TABLE III
Total energy consumption and runtime with *eDVFS* and static CPU frequencies.

VI. Conclusion

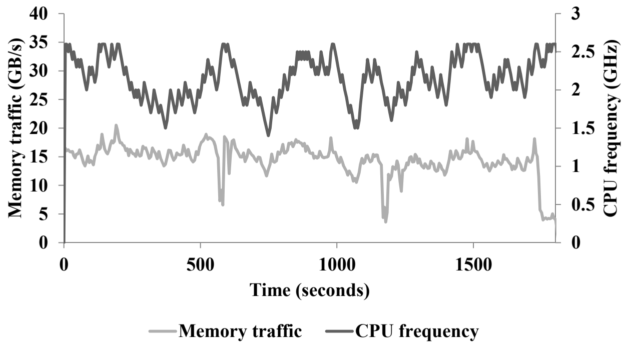
In this paper, we proposed an energy-centric dynamic DVFS controlling method to save energy of a multi-core server. We observed that energy efficiency can be improved by adjusting CPU frequency according to the characteristics of workloads. We showed that our *eDVFS* works well with memory-intensive and CPU-intensive workloads, and is more energy-efficient and faster than current "on-demand" DVFS controls.

Acknowledgment

This research was supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology (No. 2010-0020731),

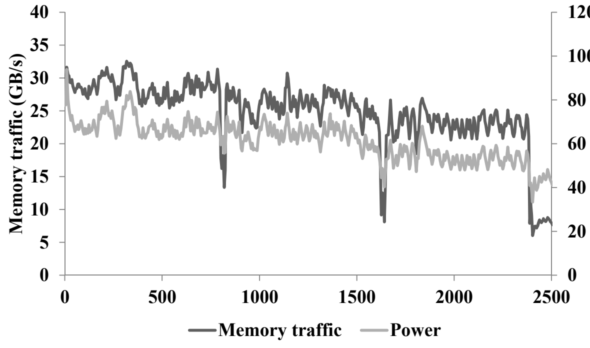


(a) Case 1: High memory traffic.

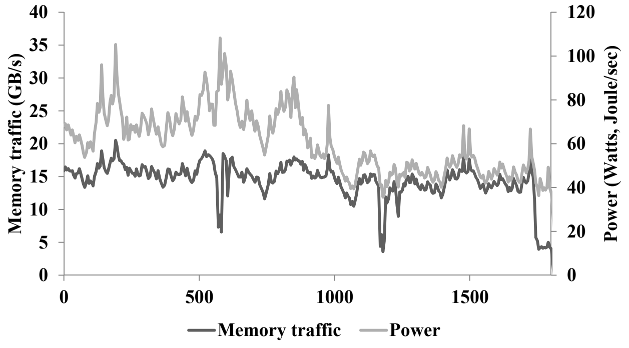


(b) Case 2: Low memory traffic.

Fig. 9. Memory traffic and CPU frequency that is controlled by our *eDVFS* method.



(a) Case 1: High memory traffic.



(b) Case 2: Low memory traffic.

Fig. 10. Memory traffic and consumed power that is controlled by our *eDVFS* method.

and Seoul R&BD Program (No. JP110034). The ICT at Seoul National University provided research facilities for this study.

REFERENCES

[1] Greenpeace, "How dirty is your data?" http://goo.gl/QCiKE.

[2] D. Meisner, B. T. Gold, and T. F. Wenisch, "Powernap: eliminating server idle power," in *Proceedings of the 14th international conference on Architectural support for programming languages and operating systems*, New York, NY, USA, 2009. [Online]. Available: http://doi.acm.org/10.1145/1508244.1508269

[3] H. Lim, A. Kansal, and J. Liu, "Power budgeting for virtualized data centers," in *Proceedings of the 2011 USENIX conference on USENIX annual technical conference*, Berkeley, CA, USA, 2011. [Online]. Available: http://dl.acm.org/citation.cfm?id=2002181.2002186

[4] 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*, New York, NY, USA, 2011. [Online]. Available: http://doi.acm.org/10.1145/2000064.2000117

[5] C. Li, A. Qouneh, and T. Li, "iSwitch: Coordinating and Optimizing Renewable Energy Powered Server Clusters," in *Proceedings of the 39th ACM/IEEE International Symposium on Computer Architecture*, Portland, OR, USA, 2012.

[6] S. Govindan, J. Liu, A. Kansal, and A. Sivasubramaniam, "Cuanta: quantifying effects of shared on-chip resource interference for consolidated virtual machines," in *Proceedings of the 2nd ACM SOCC*, 2011.

[7] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. L. Soffa, "Bubble-up: Increasing sensible co-locations for improved utilization in modern warehouse scale computers," in *Proceedings of the 44th IEEE/ACM MICRO*, 2011.

[8] F. Y.-K. Oh, H. S. Kim, H. Eom, and H. Y. Yeom, "Enabling consolidation and scaling down to provide power management for cloud computing," in *Proceedings of HotCloud 2011*, Portland, USA, 2011.

[9] A. Weissel and F. Bellosa, "Process cruise control: event-driven clock scaling for dynamic power management," in *Proceedings of the 2002 international conference on Compilers, architecture, and synthesis for embedded systems*, 2002.

[10] K. Choi, R. Soma, and M. Pedram, "Dynamic voltage and frequency scaling based on workload decomposition," in *Proceedings of the 2004 international symposium on Low power electronics and design*, 2004.

[11] G. Dhiman, V. Kontorinis, D. Tullsen, T. Rosing, E. Saxe, and J. Chew, "Dynamic workload characterization for power efficient scheduling on cmp systems," in *Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design*, 2010.

[12] Q. Wu, M. Martonosi, D. W. Clark, V. J. Reddi, D. Connors, Y. Wu, J. Lee, and D. Brooks, "A dynamic compilation framework for controlling microprocessor energy and performance," in *Proceedings of the 38th annual IEEE/ACM International Symposium on Microarchitecture*, 2005.

[13] Intel, "Intel Performance Counter Monitor," http://goo.gl/wsGVr.

[14] Larry McVoy and Carl Staelin, "LMbench - Tools for Performance Analysis," http://www.bitmover.com/lmbench/.

[15] S. Zhuravlev, S. Blagodurov, and A. Fedorova, "Addressing shared resource contention in multicore processors via scheduling," in *Proceedings of the fifteenth ASPLOS*, 2010.

[16] E. Le Sueur and G. Heiser, "Dynamic voltage and frequency scaling: the laws of diminishing returns," in *Proceedings of the 2010 international conference on Power aware computing and systems*, 2010.

[17] H. David, C. Fallin, E. Gorbatov, U. R. Hanebutte, and O. Mutlu, "Memory power management via dynamic voltage/frequency scaling," in *Proceedings of the 8th ACM international conference on Autonomic computing*, ser. ICAC '11. New York, NY, USA: ACM, 2011, pp. 31–40.