Power Consumption Reduction by Dynamic Core-Counts Control with Power Gating

Kazuyuki TAMOTO[1], Kunihito YAMAMORI[2], Masaru AIKAWA[3]

[1]Graduated school of Engineering, University of Miyazaki
[2]Faculty of Engineering, University of Miyazaki
[3]Technical Center, Faculty of Engineering, University of Miyazaki
(Tel: +81 985 58 7589, Fax: +81 985 58 7589)

[2]yamamori@cs.miyazaki-u.ac.jp

Abstract: Power Gating (PG) technology is known that it reduces static power consumption by leakage current. Linux operating system on multi-core processor evenly assigns processes to each core. Linux scheduler assigns processes to the core which has the fewest number of processes even if the core is idle. It makes an idle core in PG state change to active state, and avoids reducing power consumption. In this paper, we propose a method to reduce electric power consumption by effective use of PG. Our method modify linux kernel to change number of active cores in system dynamically, and apply PG to idle cores immediately. The number of active cores is changed according to the system load. Experimental results show that our proposed method reduces about 8kWh electric power consumption from the original kernel with keeping performance.

Keywords: Power Management, Power Gating, Linux Kernel

1 INTRODUCTION

Recent computers are required low electric power consumption in terms of thermal problem, running cost, and environmental problems. Fine process rule improves performance of CPU, and reduces total electric power consumption. However, increasing of leakage current becomes as a large problem. Leakage current is a flow through insulator of transistor when transistor is in off state, and it causes unnecessary power consumption. So many researchers try to reduce it.

To reduce electric power consumption of CPU, Clock Gating[1] (CG) is used in long time. As shown in Figure 1, CG technology stops to supply a clock signal for a CPU core and reduces power consumption. CG is easy to implement and efficient to reduce power consumption. However, leakage current remains in the clock circuit. Recently, Power Gating[2] (PG) is attracted in attention. PG technology stops to supply electric power for cores as shown in Figure 2. Thus, PG prevents to appear leakage current and reduces power consumption by leakage current. PG is now replacing with CG, and implementing in many kinds of CPUs.

CG and PG can work when cores are in idle state only. Current linux system on multi-core processor evenly assigns processes to each core. A new process is assigned to the core which has the fewest number of processes even if the core is in idle state. It means PG cannot work effectively.

In this paper, we propose a method to use PG effectively by modifying linux kernel on multi-core processor. Proposed method dynamically changes the number of active cores in order to load of linux system, and immediately apply PG for idle cores. While the system load is low, our method does not assign a new process to the idle cores to avoid interrupting PG state. As a result, power consumption of CPU is reduced. Our method also changes the number of active cores according to current load of system to keep system performance.

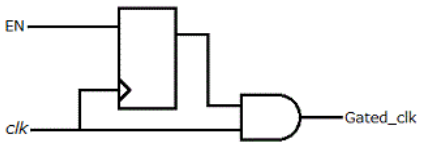

Fig. 1. A model of Clock Gating

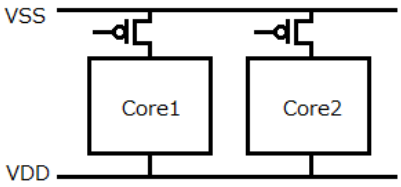

Fig. 2. A model of Power Gating

2 RELATED WORKS

Some works have been reported for effective use of PG to reduce electric power consumption. When CPU switches its state from active/PG to PG/active, additional electric power is consumed. It means that frequently change of CPU state affect power consumption. Kimura et al.[3] point out that additional electric power depends on CPU temperature, and they proposed a method that changes CPU state according to CPU temperature when processes frequently stay in WAIT state.

When CPU switches its state from active to PG, CPU must save the internal conditions such as program counter

to low level memory, and it takes additional time. Otomo et al.[4] propose a method that CPU can quickly switch from active to PG by use of an additional memory. By using of the additional memory, CPU does not need to save its internal conditions to low level memory.

These studies assume to use original CPU or FPGA, there are few researches for general CPUs.

3 IMPROVEMENT OF LINUX KERNEL

We modify a linux kernel to achieve a system that effectively uses PG on general multi-core processor. The system periodically watches load of CPU. When the load of system is low, kernel turns a CPU core off, and applies PG immediately. Meanwhile the load of system becomes high, our method wake core in PG state up. Thus, the system reduces power consumption with keeping system performance.

3.1 Environments

We implement proposed method in linux kernel 2.6.32 [5], and evaluate our method on Intel core i7 Nehalem architecture processor [6] which equips PG technology. Hyper-threading and DVFS is not used.

3.2 Dynamic Core-Counts Control

Linux kernel 2.6.32 controls on/off of core by bitmask and scheduling domain. If a bit in bitmask is 0, corresponding core is in off state. Scheduling domain manages cores with hierarchical group. If a process should move other core, it is preferentially moved to the core belonging to the same group. Figure 3 illustrates a concept of scheduling domain of hyper-threading dual core processor. If a process in core0 should move, preferential destination is core2. When system switches on/off of a core, those groups are reconstructed. In this paper, all cores belong to the same group.

Our proposed method controls the state of cores by bitmask, and reconstructs group of scheduling domain. We make original daemon process that watches the load of system and changes the number of active cores at fixed interval. In linux kernel, the lowest number of active core is one. Thus, core0 is always online, and our daemon process always works on core0.

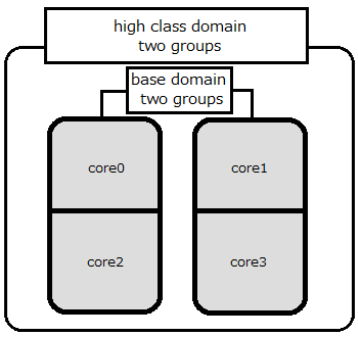

Fig. 3. Scheduling domain of dual core processor with Hyper-threading technology

3.3 Load of CPU

Our method changes the number of active cores according to the load of system. Thus, we implement a new function to get the load of system in linux kernel.

Equation (1) defines the load of the system. Linux kernel manages CPU time as the sum of system time t_s, user time t_u, idle time t_i and others t_o. The load of k-th core is defined as the ratio of active time to total time.

$$Load_k = \frac{t_s + t_u + t_o}{t_s + t_u + t_i + t_o} \times 100(\%) \quad (1)$$

3.4 PG management through ACPI

Linux kernel manages CG and PG by C-states of ACPI [6]. C-states show power states when CPU core is in idle state. C-states consist of four level states, C0 means that the core is in active state, and C3 means in PG state. The core in deep level consumes low electric power. Meanwhile, the core in deep level needs large latency when the core changes its state from PG to active. Figure 4 illustrates relations between electric power consumption and latency in each C-state level. While a core is in idle state, system changes the state from C0 to C1, C2, and C3. If a process is assigned to a core, the core returns to C0.

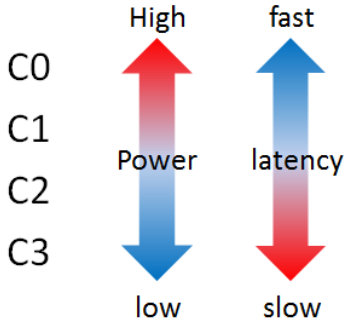

Fig. 4. Power stat and latency of each state

To reduce electric power consumption on multi-core CPU, it is important to keep the idle core in C3 state. However, linux scheduler evenly assigns a process to the core with the fewest number of processes, so C3 state is frequently interrupted. To avoid this situation, we logically removed the idle core from system as described in Section 3.2 since linux operating system cannot detect the logically removed core, no process is assigned to the removed core, and the core can keep C3 state. On the other hand, the remaining cores have to work harder because the number of cores is decreased, and it may make electric power consumption of remaining cores increase. If the load of system is high, this increasing of electric power consumption can ignore because the cores work hard and they already consume all the required electric power. If the load of system is low, increasing of electric power consumption of remaining cores may become a problem since logical removing of a core prevents remaining cores staying C1, C2 or C3.

Here we investigate the relations between the system

load and electric power consumption according to the number of active cores. Figure 5 and Figure 6 show that transition of load and power consumption of each number of active cores when initial average load is about 40%, 30%, 20% and 10% in four active cores. In both figures, bars denote the electric power consumption corresponding to the left vertical axis, and lines denote the average load of the system corresponding to the right vertical axis.

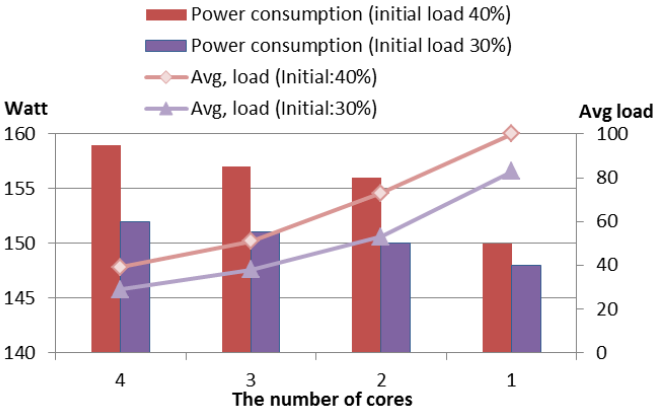

Fig. 5. Average load and power consumption when initial load is 40% and 30%.

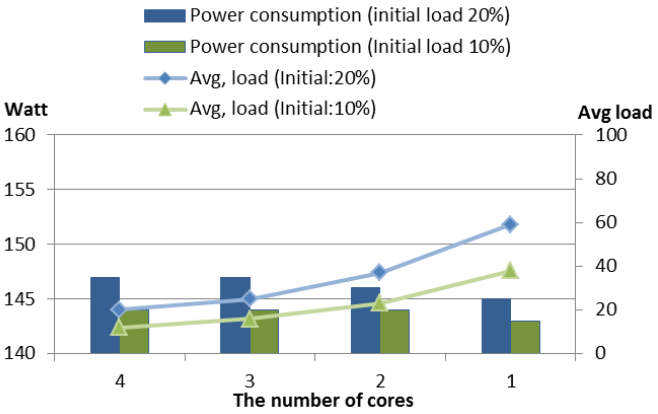

Fig. 6. Average load and power consumption when initial load is 20% and 10%.

Figure 5 shows that power consumption is clearly reduced when turn CPU cores off although the load of remaining cores is increasing. This means that logical removing of cores gives us reducing electric power consumption even if it increases the load of the remaining cores when the initial load of system is high. On the other hand, reducing of electric power consumption is not clear when the initial load of system is low as shown in Figure 6. But electric power consumption continues to be down even if the initial average load is 10%.

From experimental results, power consumption is actually reduced when cores are turned off.

3.5 Threshold for core state switching

Figure 5 and Figure 6 show that system can run with fewer cores even if load is moderately high. However, if high load leads poor performance, system should increase the number of active cores.

Threshold to turn the core on/off is not clear. Since early switching from off to on leads high electric power consumption, this switching is desirable near to 100% of system load. Figure 5 and Figure 6 shows that the average load of system is increased when a core is logically removed because the processes assigned to the removed core are distributed to other remaining cores. Therefore the threshold to turn the core on/off should be decided by the load of remaining cores. Suppose N denotes the number of remaining cores, the average load is defined by Equation (2).

$$Avg_{load} = \frac{\sum Load_k}{N-1} \ (N \geq 2) \qquad (2)$$

When the Avg_{load} becomes near to100%, it means the system performance may down. In the following experiments, we use Avg_{load} = 90% to switch the core state on/off.

4 EXPERIMENTS AND DISCUSSIONS

4.1 Experimental conditions

We evaluate electric power consumption and execution time by comparing our proposed method with original linux kernel 2.6.32. Our modified kernel watches the load of system at every two seconds, and decides to make a core remove or return. Test programs change Avg_{load} at random by forking small programs in many times, so system changes the number of active cores. We measure elapse time until the test programs have finished. Moreover, we evaluate performance of a floating-point matrix calculation under the modified kernel.

4.2 Results and Discussion

Figure 7 shows electric power consumption at every second of proposed method and original kernel. The horizontal axis is seconds, and vertical axis is actual power consumption of whole system. Figure 7 says that our method globally reduces power consumption. At the beginning, the number of active cores is one in our proposed method. Since our proposed method uses only a few cores for several seconds from the beginning, electric power consumption stays low. As shown in Figure 7, electric power consumption of our kernel is lower than that of the original kernel after 109 seconds since our kernel finishes test programs faster than the original kernel. Figure 7 also shows that our method is effective in the case which the electric power consumption by original kernel is more than 165W. Our kernel employs a hard threshold to decide on/off of a core, the curve of our kernel sometimes waves around 40 seconds and 65 seconds.

Table 1 shows mathematical total electric power consumption by our proposed kernel and the original kernel calculated from Figure 7. As shown in Table 1, our method can reduce about 8kWh from original kernel.

Table 2 shows executing time of test programs. It shows our method is faster than original system. It is because

migration mechanism of linux kernel. Linux kernel keeps load balance through this migration mechanism to move the process from busy core to idle core. This migration itself becomes overhead for execution of processes. Our test program forks many small processes, and the kernel assigns these processes into online cores. Since our kernel makes some cores stay offline, migration over cores is rare case and it reduce overhead by migration. Therefore execution time on our kernel becomes shorter than that of the original kernel.

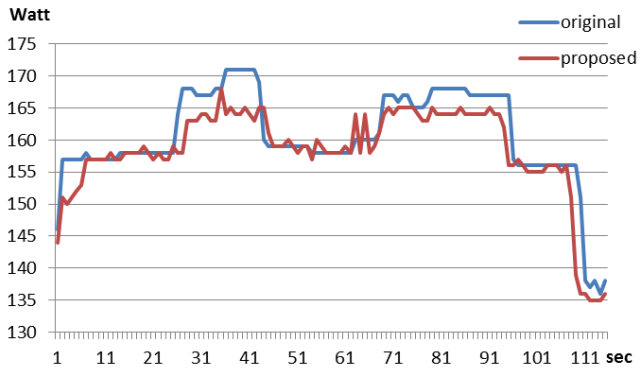

Fig. 7. Comparison with electric power consumption

Table 1. Average of power consumption

Original(kWh)	Proposed(kWh)
578.22	570.30

Table 2. Execution time of test programs

Original(sec)	Proposed(sec)
109.20	106.11

We also evaluate execution time of floating-point calculation by inverse matrix calculation whether our modified kernel affects the performance of scientific calculation. Table 3 shows the execution time of floating-point inverse matrix calculation. "Number of processes" denotes the number of floating-point calculation process simultaneously executed. Table 3 says that our kernel is about 3% slower than the original kernel. It is because the daemon process to decide on/off of the core described in Section 3.2. This daemon process always stays in core0. In the case of one floating-point calculation is executed, only the core0 is the active core. The core0 has to execute daemon process at every two seconds, it disturbs floating-point calculation. When we throw two floating-point calculations simultaneously, the one finishes about 6 seconds faster than the other. It means that the process assigned into core0 is affected by the daemon process. The number of active cores is the same between the original kernel and our modified kernel, so the electric power consumption is almost the same.

From the experiments, our method is suitable for the systems that execute many small processes or threads with high load. In that case, our method copes with both low electric power consumption and faster execution.

Table 3. Processing time of floating-point calculation

Number of Processes	Original(sec)	Proposed(sec)
1	133.26	139.68
2	132.99	134.81
	133.99	141.09

5 CONCLUSIONS

In this paper, we propose a method to reduce electric power consumption on linux servers. Our method dynamically manages the number of active cores, and immediately applies PG for idle cores. In addition, our method logically removes the idle cores from the system, and makes no process assign into idle cores to keep the core in PG state. When the system load is over threshold, our method return the cores in PG state to active. Experimental results show that our method copes with both low electric power consumption and good performance for the system executing many small processes with high load. It remains future works to employ "soft" threshold to stable electric power consumption and improve algorithm for core state switching.

REFERENCES

[1] Wu Q, Pedram M, Wu X (1997), Clock-Gating and its application to low power design of sequential circuits. Custom Integrated Circuits Conference, 1997, Proceedings of the IEEE 1997, pp.479-482

[2] Pakbaznia E, Fallah F, Pedram M (2008), Charge Recycling in Power-Gated CMOS Circuits. Computer-Aided Design of Integrated Circuits and Systems, Vol.27, Issue.10, pp.1798-1811

[3] Kimura K, Kondo M, Amano H (2011), Fine Grain Power Gating Control Adapting to a Change in Core Temperature with Operating System, Vol.2011-ARC-195 No.30, pp.1-8

[4] Otomo T, Kurihara K, Teranishi Y (2012), High-reliabillity, Low-power and High-performance Implementation by Memory Redundancy and Power Gating, Vol.2012-ARC-199 No.14,pp.1-8

[5] http://www.kernel.org/

[6] Intel(R) Turbo Boost Technology in Intel(R) Core(TM) Microarchitecture (Nehalem) Based Processors (2008) Intel White Paper