Software Design Methodology based on Energy Consumption Model considering Relationship between Software and Hardware

Koji Kurihara† Hiromasa Yamauchi† Toshiya Otomo† Takahisa Suzuki† Yuta Teranishi‡ Koichiro Yamashita†

†Processor Solution Development Lab.
Fujitsu Laboratories Ltd.
1-1, Kamikodanaka 4-chome,
Nakahara-ku, Kawasaki 211-8588 Japan
Tel : +81-44-754-2781
Fax : +81-44-754-2744
e-mail : {kouji3211, yamauchi.h, otomo.toshiya, suzuki.takahisa, yamashita.ko-05}@jp.fujitsu.com

‡Engineering Dept.I
Fujitsu Kyushu Network Technologies Limited
Fujitsu Kyushu R&D Center Bldg, 2-1
Momochihama 2-chome, Sawara-ku,
Fukuoka 814-8588, Japan
Tel : +81-92-852-8159
Fax : +81-92-852-3241
e-mail : teranishi.yuta@jp.fujitsu.com

Abstract - In an SoC for industrial systems, there is a case that we have to optimize energy consumption and performance with existing software and hardware. However, it is difficult to achieve this without evaluation methodology considering relationship between software and hardware. Therefore, we propose an evaluation methodology based on energy consumption model considering relationship between software and hardware. We verified the accuracy of our methodology by comparing it to an experimental result.

I Introduction

SoC becomes more and more complicated because a single chip is composed of many component modules demanding high performance, downsizing and low price. Therefore, each component module such as application programs, processors and memory is optimized before they are integrated.

However, the performance of the whole SoC system is not always optimized even if all modules are optimized separately. In **Fig 1**, "CPU clock frequency" is the peak clock of typical embedded processor. And "performance of whole SoC system" is the result of benchmark score. We use the benchmark program that evaluates total performance of application use-case. As shown in **Fig 1**, the CPU clock frequency is increasing every year, but the performance of whole SoC system is hardly changing after 2009. This is because, in an attempt to optimize performance or energy consumption, improving processor utilization is focused in software design, while, in the hardware design, the focus is on reducing stall time between instructions. Thus, the criteria of optimizing each module are different.

But in an SoC for industrial systems, we have to optimize both energy consumption and performance for the specified software and hardware.

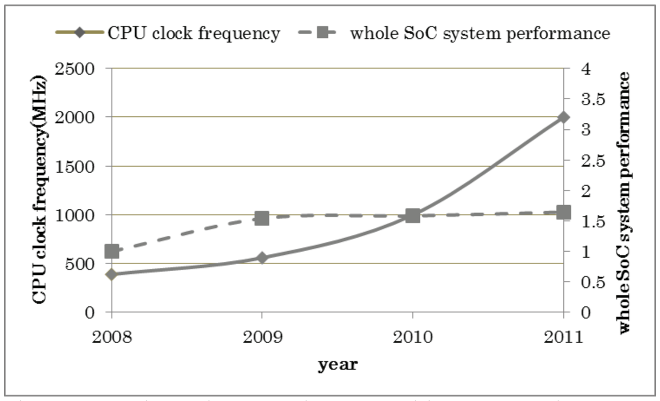


Fig 1. Comparison of CPU performance with System performance

II. Scheduling Method considering Characteristics of Software and Hardware

There are some overheads caused by combining an existing software and hardware because the criteria of software optimization and hardware optimization are different. The memory access contention is one of the overheads caused by running the existing software on existing hardware. A scheduling method considering the memory access contention has been proposed [8]. This scheduling method selects one scheduling policy from two policies according to the influence of memory access contention. One of them is to balance the load of each processor. The other is to avoid memory access contention. Many scheduling methods that balance the load of each processor are proposed [2][3][4]. Therefore, in this paper, we explain the scheduling method to avoid memory access contention.

Memory access contention occurs when multiple processors access a shared memory at the same time. Therefore, if each processor frequently accesses the shared memory, memory access contention increases. The processor frequently accesses the shared memory when it executes a thread whose memory foot print is large. Therefore, in the scheduling method to avoid memory access contention, threads whose memory footprint is large are allocated to the same processor.

We explain the reason why this scheduling method can avoid memory access contention. In this case, thread#0, thread#1, thread#2 and thread#3 are running on a Tightly Coupled Multi-Processor (TCMP) platform which has two processors and one shared memory. Let us suppose that memory footprint of thread#0 and thread#1 is large, and memory footprint of thread#2 and thread#3 is small. The loads of all threads are same. Given that these threads are allocated to each processor using the scheduling method to balance the load of each processor, the result of scheduling is shown in **Fig 2**. As shown in **Fig 2**, there is a case that thread#0 and thread#2 are allocated to processor#0, and thread#1 and thread#3 are allocated to processor#1. When some threads are allocated to the same processor, these threads are executed by multitasking, that is, the processor switches these threads by time slice. Then, when executing four threads as shown in **Fig 2**, there is a case that thread#0 and thread#1 are executed at the same time by processor#0 and processor#1. In this case, there are frequent memory accesses by processor#0 and processor#1. So the probability that memory access contention occurs is very high. Next, we explain the result of allocation by scheduling method to avoid memory access contention.

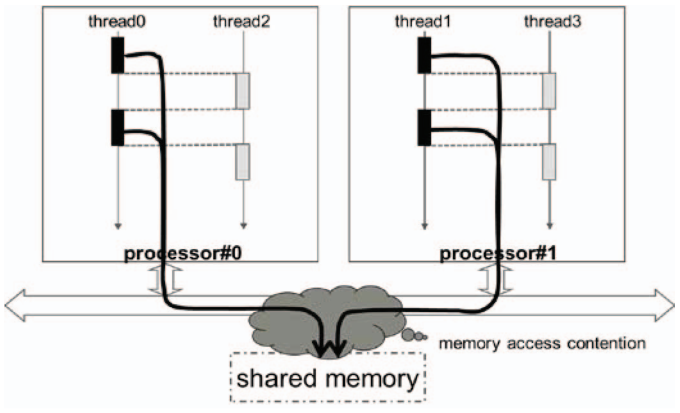


Fig 2. Result of allocating threads by scheduling method of balancing load of each processor

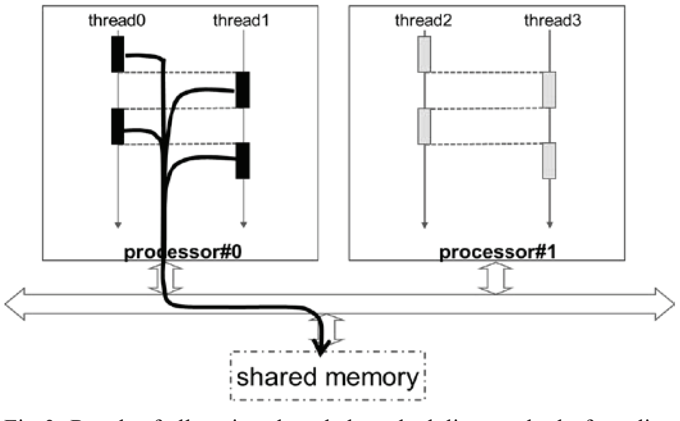


Fig 3. Result of allocating threads by scheduling method of evading memory access contention

As shown in **Fig 3**, if four threads are allocated using the scheduling method to avoid memory access contention, thread#0 and thread#1 are allocated to the same processor without failure. In this case, thread#0 and thread#1 are executed by multitasking. In other words, thread#0 and thread#1 are not executed at the same time. So memory access contention is hardly generated because even if there are frequent memory accesses by processor#0 but not as frequent as by processor#1. When we combine an existing software and hardware, we have to consider some overheads like memory access contention.

III. Motivation

The scheduling method to balance the load of each processor was presented by J. Leung and S. K. Dhall [2] [3]. In these scheduling methods, one thread queue is prepared for all processors or some processors, and threads being not executed are queued. When each processor executes these threads, each processor takes it out of the queue. The order in which the threads are taken out follows the given execution priority of each thread. A different scheduling method to balance the load of each processor was presented by Andersson, B. [4]. In this scheduling method, first, the scheduler classifies all the threads by the amount of their loads. Next, the scheduler gives groups of these threads the priority of allocation. Lastly, the scheduler allocates groups of threads to each processor in the order of their priority, according to the utilization of each processor. Furthermore, the preemptive scheduling method for real-time systems was presented by Madhusudan Rao [5]. In this scheduling method, the scheduler sets the timer on each thread. Each timer is counted up while each thread is running. If the scheduler detects that the value of the timer reached the threshold, the scheduler stops the execution of this thread. Then, the scheduler makes the processor execute another thread. The value of the timer of the stopped thread is initialized. If the memory access contention occurs, the frequency of main memory access increases. Therefore the energy consumption increases if the memory access contention occurs. However, in the above-mentioned papers, overheads like the memory access contention are not considered.

From the viewpoint of the analysis of the energy consumption, some methods were presented. The method to analyze the energy consumption was presented by Hiroshi Mizuho [6]. In this method, first, energy consumption models of each hardware component are created. Next, the energy consumption of entire system is calculated by adding the value of each energy consumption model. A different method to analyze the energy consumption was presented by Da Qi Ren [7]. In this method, according to application program, the energy consumption of entire system is calculated by identifying hardware components that are in operation while application program is executed. However, in the above-mentioned papers, the energy consumption generated by the memory access contention is not considered. Therefore we cannot utilize methods mentioned in above papers for evaluation of scheduling method to avoid the memory access contention.

IV. Formulation of Energy Consumption Model considering Characteristics of Software and Hardware

In this paper, as preliminary step to evaluate the energy consumption of scheduling methods, we create energy consumption model considering relationship between software and hardware.

A. Energy consumption model from the viewpoint of software

Fig 4 shows energy consumption model from the viewpoint of software. Energy consumption model of executing thread#0 and thread#1 is created by considering the power consumption of processor and the execution time of each thread. Therefore, the energy consumption of thread#0 and thread#1 can be represented as follows.

$$J_{software} = P_{processor} \times (t_0 + t_1), \tag{1}$$

where $P_{processor}$ represents power consumption per unit time of processor, t_0 the execution time of thread#0, and t_1 the execution time of thread#1.

B. Energy consumption model from the viewpoint of hardware

In case of creating energy consumption model from the viewpoint of hardware, the model is created by considering the structure of hardware such as processor, cache memory, buss and main memory. In the system shown in **Fig 5**, the energy consumption model from the viewpoint of hardware is formulated by the sum of energy consumption of processors, L2 cache memory, main bus and main memory. A processor consumes energy while it executes threads. So, the energy consumption of processors is calculated by multiplying the power consumption of processor by a total of running time of processors. L2 cache memory, main bus and main memory consume energy when they are accessed by the processor.

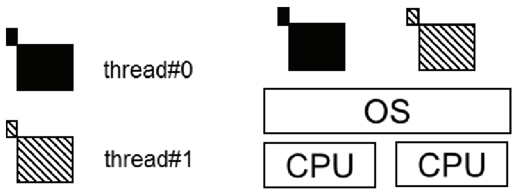


Fig 4. Energy consumption model from the view point of software

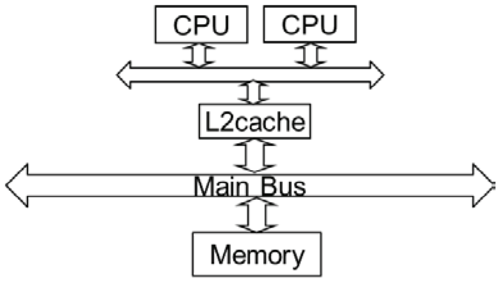


Fig 5. Energy consumption model from the view point of hardware

So, the energy consumption of L2 cache memory is calculated by multiplying the power consumption per L2 cache memory access by a total of number of L2 cache memory accesses. The energy consumption of main bus is calculated by multiplying power consumption per main bus access by a total of number of main memory accesses. This is because the processor accesses main bus without failure when the processor accesses the main memory. The energy consumption of main memory is calculated by multiplying power consumption per main memory access by a total of number of main memory accesses. A total of running time of processors is equal to a total of processing time of thread#0 and thread#1. Therefore, energy consumption of executing thread#0 and thread#1 from the viewpoint of hardware can be formulated as follows.

$$\begin{aligned} J_{hardware} = {} & P_{processor} \times t_0 + P_{L2} \times C_{L2}(ft_0) \\ & + P_{bus} \times C_{mem}(ft_0) + P_{mem} \times C_{mem}(ft_0) \\ & + P_{processor} \times t_1 + P_{L2} \times C_{L2}(ft_1) \\ & + P_{bus} \times C_{mem}(ft_1) + P_{mem} \times C_{mem}(ft_1), \end{aligned} \tag{2}$$

where $P_{processor}$ represents power consumption per unit time of processor,
P_{L2} power consumption per L2 cache memory access,
P_{bus} power consumption per main bus access,
P_{mem} power consumption per main memory access,
ft_0 memory footprint of thread#0, and
ft_1 memory footprint of thread#1.
$C_{L2}(ft)$ represents the frequency of L2 cache access when memory footprint of thread is ft. This is because frequency of L2 cache access varies with memory footprint of thread. $C_{mem}(ft)$ represents the frequency of memory access when memory footprint of thread is ft. This is because frequency of memory access varies with memory footprint of thread.

C. Energy consumption model considering relationship between software and hardware

In the target system shown in **Fig 6**, L2 cache memory is shared with two processors. Therefore, one thread may kick out data of the other thread which is executed at the same time from L2 cache memory. The processor executing the thread whose data may be moved out from L2 cache memory must access main memory. Thus, the frequency of main memory access increases compared to the case that only one thread is executed. The amount of an increase is influenced by the frequency of L2 cache memory access from other thread. The more the thread accesses L2 cache memory, the higher the probability of moving out the data of other thread. Therefore, energy consumption of executing thread#0 and thread#1 created by considering relationship between software and hardware can be represented as follows.

$$\begin{aligned}J_{all} &= P_{processor} \times t_0 + P_{L2} \times C_{L2}(ft_0) \\ &\quad + RA\{P_{bus} \times C_{mem}(ft_0) + P_{mem} \times C_{mem}(ft_0)\} \\ &\quad + P_{processor} \times t_1 + P_{L2} \times C_{L2}(ft_1) \\ &\quad + RB\{P_{bus} \times C_{mem}(ft_1) + P_{mem} \times C_{mem}(ft_1)\}, \qquad (3)\end{aligned}$$

where RA represents increasing proportion of frequency that the thread#0 accesses main memory when the thread#1 is executed at the same time, RB increasing ratio of frequency that the thread#1 accesses main memory when the thread#0 is executed at the same time.

V. Evaluation of Energy Consumption Model considering Relationship between Software and Hardware

We verified the accuracy of our methodology by comparing it to an experimental result. The evaluated value is the energy consumption generated by memory access contention.

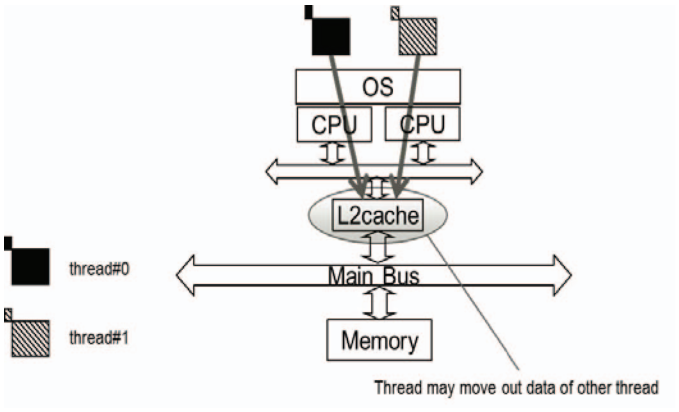
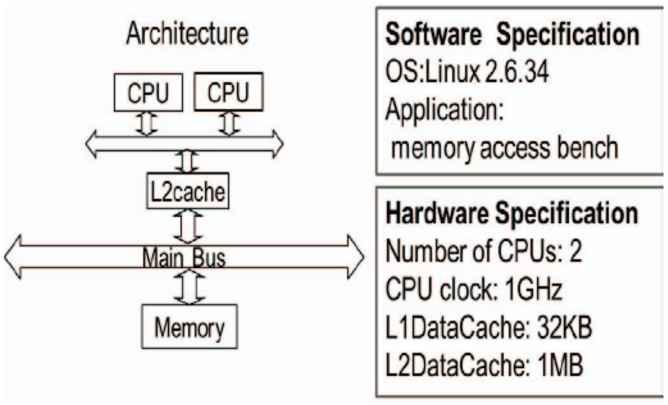


Fig 6. Model considering relationship between software and hardware



Software Specification
OS:Linux 2.6.34
Application:
memory access bench

Hardware Specification
Number of CPUs: 2
CPU clock: 1GHz
L1DataCache: 32KB
L2DataCache: 1MB

Fig 7. Target actual system

A. *Target actual system*

Target actual system is shown in **Fig 7**. This system has two processors, one L2 data cache memory and one main memory. Each processor has L1 data cache memory. Size of each L1 data cache memory is 32KB. L2 data cache memory and main memory are shared with two processors. Size of L2 data cache memory is 1MB. We created a benchmark program such that both of the two threads (thread#0 and thread#1) frequently access the memory. Memory footprint size of one thread is fixed to 2MB. Memory footprint size of the other thread is changed from 32KB to 2MB. We measured the increase of energy consumption generated by memory access contention.

B. *Calculation by using our energy consumption model*

In our energy consumption model, the increase of energy consumption generated by memory access contention can be calculated by subtracting the equation (2) from the equation (3). This is because the equation (2) represents energy consumption excluding energy generated by memory access contention, and the equation (3) represents energy consumption including it. Therefore, the increment of energy consumption generated by memory access contention can be shown as follows.

$$\begin{aligned}&D(ft_0, ft_1) \\ &= J_{all} - J_{hardware} \\ &= (P_{bus} + P_{mem})\{RA \times C_{mem}(ft_0) \\ &\quad + RB \times C_{mem}(ft_1) - (C_{mem}(ft_0) + C_{mem}(ft_1)\} \\ &= (P_{bus} + P_{mem})\{(RA - 1) \times C_{mem}(ft_0) \\ &\quad + (RB - 1) \times C_{mem}(ft_1)\}. \qquad (4)\end{aligned}$$

RA and RB are derived from characteristics of software and hardware. They changes according to memory footprint of threads. Therefore, if thread#0 whose memory footprint is ft_0 and thread #1 whose memory footprint is ft_1 are executed at the same time, RA and RB can be shown as follows.

$$RA = X(ft_0, ft_1)$$
$$RB = Y(ft_1, ft_0). \quad (5)$$

In this measurement, memory footprint size of thread#1 is fixed to 2MB. Thus, ft_1 is set to 2MB.

We acquire RA by measuring target actual system. It is also possible to acquire RA by cycle accurate simulation with accurate main bus model.
RA is a factor of main memory access frequency when thread#0 and thread#1 are executed at the same time. The performance becomes worse if frequency of main memory access increases, that is, performance decreasing rate is proportional to frequency of main memory access. Therefore, RA can be considered as a decreasing rate of performance. The performance decreasing rate can be measured by comparing performance of executing only thread#0 to the performance of executing thread#0 and thread#1 in parallel. First, we execute only thread#0, and measure execution time. We measure each thread#0 memory footprint size(32KB～2MB). This result represents the performance of executing only thread#0. Next, we execute thread#0 and thread#1 simultaneously, and measure execution time. We measure each thread#0 memory footprint size(32KB～2MB). This result represents the performance of executing thread#0 and thread#1 in parallel. RA can be calculated by two results of measurement. RA is performance of executing thread#0 and thread#1 in parallel when the performance of executing only thread#0 is assumed to be 1. RA is represented in **Fig 8** . The X-axis represents memory footprint of thread#0. The Y-axis represents ratio of the performance of executing thread#0 and thread#1 to that of executing only thread#0.

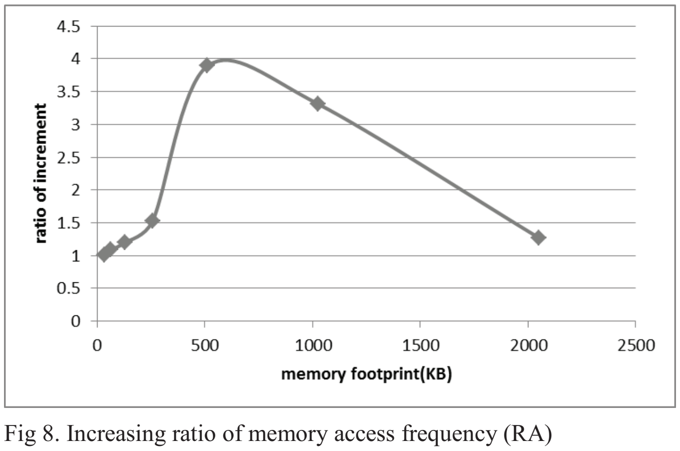


Fig 8. Increasing ratio of memory access frequency (RA)

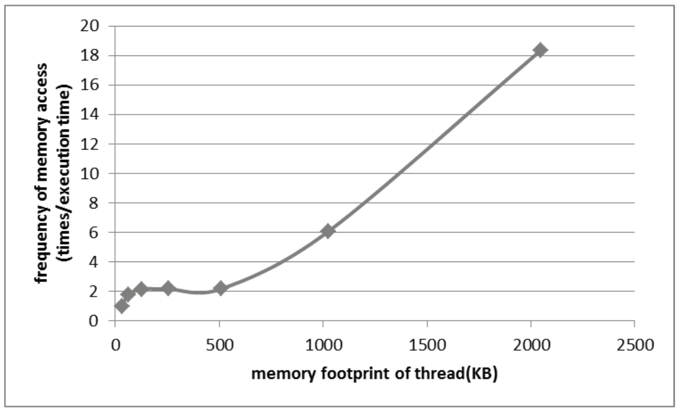


Fig 9. Frequency of memory access in each memory footprint (C_{mem}(ft))

We can acquire RB by measuring target actual system. It is also possible to acquire RB by measurement of simulation. But, in this target system, RB can be considered to be 1. RB becomes largest value when ft_0 is 2MB. According to **Fig 8**, RB is almost 1 when ft_0 is 1. Therefore, RB hardly changes from 1 because the minimum value of RB is 1. Thus, D(ft_0, 2MB) in the equation (4) is represented as follows.

$$D(ft_0, 2MB) = (P_{bus} + P_{mem}) \times \{(X(ft_0, 2MB) - 1) \times C_{mem}(ft_0)\}, (6)$$

C_{mem}(ft) represents the frequency of main memory access. If the frequency of main memory access increases, the execution performance decreases. Therefore, behavior of C_{mem}(ft) is similar to that of performance of executing only one thread. **Fig 9** shows C_{mem}(ft) on target actual system when C_{mem}(32KB) is assumed to be 1. The X-axis represents ft. Given that (P_{bus}+P_{mem}) is equal to 1, D(ft_0, 2MB) is represented in **Fig 10** as a solid line. The X-axis represents memory footprint size of thread#0. The Y-axis represents energy consumption generated by memory access contention.

C. Measurement using target actual system

We measure energy consumption of target actual system by executing benchmark program on it. Then, we change ft_0 from 32KB to 2MB. The result of this experiment is represented in **Fig 10** as a dotted line.

D. Experimental Results

Fig 10 shows the energy comparison results. As can be seen from the **Fig 10**, the value calculated by our energy consumption model is larger than the value measured by executing benchmark in every case.

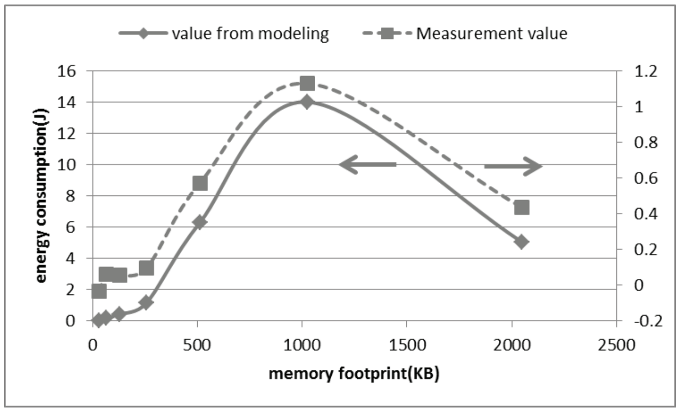


Fig 10. Energy consumption generated by memory access contention

This is because power consumption of main bus and main memory access is not represented from characteristics of hardware and benchmark program. The behavior of solid line in **Fig 10** is similar to that of dotted line in **Fig 10**. Our energy consumption model can simulate the behavior of target system energy consumption. Therefore, our energy consumption model is useful for analyzing behavior of energy consumption generated by memory access contention on the actual system.

VI. Conclusion and Future Work

In this paper, we proposed energy consumption model considering relationship between software and hardware. We verified the accuracy of our energy consumption model by comparing the result to the value measured by executing benchmark on target actual system. As a result, our energy consumption model has proved accurate, and qualified useful for analyzing the behavior of energy consumption generated by memory access contention. For example, we can design a scheduler of OS considering memory access contention by applying our energy consumption model.

The future work is to verify the accuracy of our energy consumption model by using commercial application program because we use only in-house benchmark program in this paper, and to evaluate energy consumption of scheduling methods.

Acknowledgements

Special thanks to my colleagues and supervisor for valuable discussion and helpful support.

References

[1] Koichiro YAMASHITA, “A software centric system design for OS scheduling scheme in the upstream phase,” MPSoC’11, http://www.mpsoc-forum.org/slides/16.6-Yamashita.pdf, 2011.

[2] J. Leung and J.Whitehead, “On the Complexity of Fixed-priority Scheduling of Periodic Real-Time Tasks,” Performance Evaluation, Elsevier Science, vol. 22, pp.237-250, 1982.

[3] S. K. Dhall and C. L. Liu, “On a real-time scheduling program,” Operations Research, vol.26, pp.127-140, 1978.

[4] Andersson, B. and Tovar, E., “Multiprocessor Scheduling with Few Preemptions,” Proc. Of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, pp.322-334, 2006.

[5] Madhusudan Rao, B. , Keerthi Teja, M. , Nitin, N., “Comparison of Process Scheduling Methodologies for Embedded Systems,” Emerging Trends in Engineering and Technology(ICETET), pp.387-391, 2009.

[6] Hiroshi Mizuno, Hiroyuki Kobayashi, Takao Onoya and Isao Shirakawa, “POWER ESTIMATION AT ARCHITECTURE LEVEL FOR EMBEDDED SYSTEMS,” IEEE International Symposium on Circuits and Systems, pp.II-476-II-479 vol.2, 2002.

[7] Da Qi Ren and Reiji Suda, “Modeling and Estimation for the Power Consumption of Matrix Computation on Multi-core Platform,” International Joint Conference on Computational Science and Optimization, CSO 2009, pp.42-46, 2009.

[8] Koji KURIHARA, Hiromasa YAMAUCHI, Toshiya OTOMO, Takahisa SUZUKI, Naoki ODATE and Koichiro YAMASHITA, “SMP-scheduling scheme for the mobile platform of next-generation,” Technical Committee on Integrated Circuits and Devices, pp.63-68, 2011