# Control Method of Multiple Services for CMP Based on Continuation Model

Hideaki Moriyama, Toshihiro Yamauchi, and Hideo Taniguchi
Graduate School of Natural Science and Technology, Okayama University
3-1-1 Tsushima-naka, Kitaku, Okayama, 700-8530, Japan

***Abstract*—In a chip multiprocessor based on the continuation concept, the hardware scheduler controls threads and achieves high performance on thread scheduling. However, the priority of threads is not considered during execution because the hardware thread scheduler schedules threads in a FIFO manner. Therefore, when multiple services execute simultaneously, the execution of each service cannot consider the priority of service. In such a case, software support is needed to control the execution of each service. This paper presents a software scheduler for multiple services that supports the hardware scheduler. In addition, this paper also reports the evaluation of the software scheduler, which targets multiple services.**

## I. Introduction

With the increase in the use of multi-core processors, the effective execution of parallel computation in a computer with multiple cores has become important. However, in parallel computation that involves a commodity processor and an operating system, the overheads of the operating system, such as the system call or context switch, are a major disadvantage [1]. Previously, a study reported a method for parallel computation by using the data flow machine [2]. By using this machine, a thread that is in ready state starts executing sequentially. Therefore, this method helps in achieving fast parallel computation by reducing the overheads of thread management. However, it has two disadvantages. First, it is not possible to control the priority of individual threads precisely. Second, owing to the type of instruction set architecture, it is difficult to implement this method.

To solve the abovementioned problems, the concept of fine-grained multithreading was introduced [3]. This method, which is based on the data-flow-computation model, achieves effective parallel execution of the threads. The *FUsion of Communication and Execution (Fuce)* processor is developed as a variant of the chip multiprocessor (CMP) [4] [5], which has multiple *Thread Execution Units (TEUs)* and adapts fine-grained multithreading. Services built on *Fuce* are apt to have many threads. An individual thread is allowed to run throughout without any interruption. Fast thread-synchronization mechanism called *continuation* is realized using a hardware scheduler. Therefore, the *Fuce* processor achieves high performance with regard to thread scheduling.

Figure 1 shows the concept of *continuation*. The figure shows dependence among the four threads *A*, *B*, *C* and *D*. Threads *B* and *C* need the result of calculation by thread *A*. Thread *D* needs the result of calculation by threads *B* and *C*. For executing these four threads, it is necessary to transmit the results of calculation and signals from thread *A* to threads *B* and *C*, and then from threads *B* and *C* to thread *D*. On a *Fuce* processor, this signaling mechanism is called *continuation*. In Figure 1, thread *A* continues to threads *B* and *C*, and threads *B* and *C* continue to thread *D*. The hardware thread scheduler on the *Fuce* processor schedules the threads in a FIFO manner. Therefore, the bare hardware for thread scheduling is not sufficient to control the priority of the individual threads precisely. Therefore, on parallel and concurrent execution of multiple services, the execution of each service cannot be prioritized.

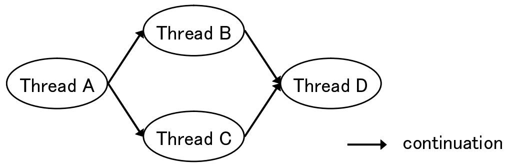


Fig. 1. Concept of continuation.

We proposed a scheduling method that is used by the software scheduler to control the *continuation* between threads [6], and we evaluated the scheduling method for single service [7]. On evaluation, we found the characteristics of the control parameters and established the control method. However, to control the execution of multiple services, it is important to extend the existing scheduling method.

In this paper, we propose a software scheduler, which helps in controlling the priority of execution of individual services on parallel and concurrent execution of multiple services. First, we propose a scheduling method for multiple services, which is the extension of [7]. Second, we describe a control method, which can control the execution of multiple services on the basis of priority. Finally, we describe the result of the evaluation of the scheduling method on a *Fuce* Software Simulator.

## II. Continuation-based Scheduling

### A. Basic Method

Following are the prerequisites for a scheduling method.

1) Processing is performed only by the *Processing Units (PUs)*.
2) The number of threads that can execute in parallel at same time $N$ is known.
3) The total number of *TEUs*, $E$ is known.

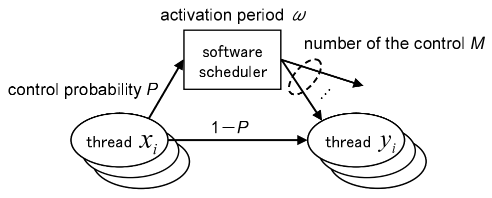


Fig. 2. Basic method of the software scheduler.

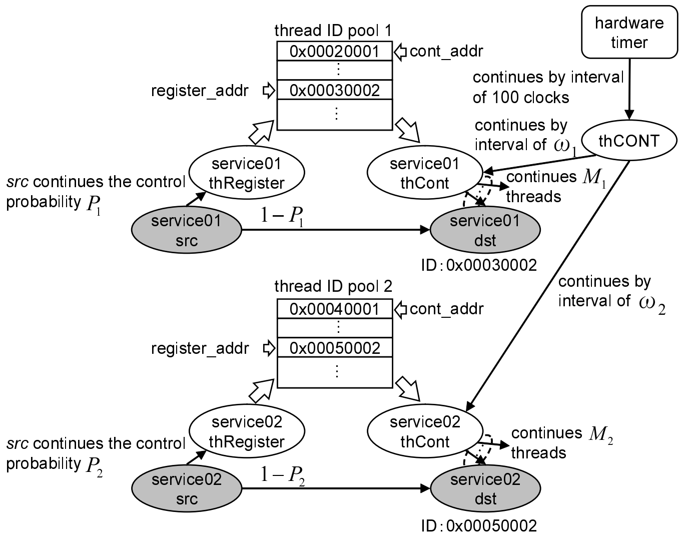


Fig. 3. Processing flow for scheduling two services.

The basic method of scheduling is to control *continuation*. Figure 2 shows the basic method of scheduling, and details are explained below.

1) In *continuation* from $thread\ x_i$ to $thread\ y_i$, $thread\ x_i$ changes destination of *continuation* to the thread of the software scheduler with the control probability $P$.
2) The hardware timer continues the threads from the software scheduler in an interval of $\omega$ clocks, where $\omega$ is called the activation period, and the software scheduler begins execution.
3) The software scheduler continues $M$ threads; $M$ is the number of the control.

## B. Extension for Multiple Services

According to the basic method shown in Figure 2, we have described the details of implementation of the scheduling method for single service in [7]. To control the multiple services on the basis of priority, in this section, we propose the scheduling method, which is an extension of the scheduler for single service.

Figure 3 shows the processing flow for scheduling two services, and following are the details of this scheduling process.

1) In each service, the preceding thread (*src*) changes the destination thread (*dst*) of the *continuation* to the registration thread (*thRegister*) of the software scheduler with the control probability $P$.
2) *thRegister* registers the thread ID of *dst* with the thread ID pool. By updating the thread ID pool, *thRegister* attempts exclusion control if necessary.
3) The hardware timer continues to the continuation thread (*thCont*), and *thCont* starts processing. The interval of *continuation* by hardware timer is denoted by $\omega$, which is the activation period. *thCont* gets $M$ thread IDs from the thread ID pool.
4) *thCont* continues $M$ threads.

## III. Control Method of Multiple Services

### A. Priority Execution and Even Execution

In the parallel and concurrent execution of multiple services, a priority is set for each service. The priority of service $i$ is indicated by $p_i(1, 2, \cdots, i, \cdots)$, and the number of services with priority $p_i$ is indicated by $m_i$. If the value of $i$ is small, then the priority of the service is high. For example, services with priority $p_i$ have higher priority than the services with priority $p_{i+1}$.

By controlling the number of *TEUs* available for services with high priority, such services can use many *TEUs*. Thus, it is possible to run each service on the basis of priority.

By controlling the number of *TEUs* available for services with the same priority, it is possible to run such services evenly.

### B. Control Method

The control method requires the settings of the control parameters to control the execution of multiple services on the basis of priority. The settings of the control parameters are described as follows.

1) The value of $\omega$ is set to $T$, and $T$ is the value of the average execution time of thread. This implies that the setting $\omega = T$ helps in achieving low overhead of the software scheduler.
2) $P$ is set to 0 for the number of services, $m_1$, with $p_1$. Setting $P = 0$ implies that the software scheduler cannot control the number of *TEUs* available for these services.
3) If the service satisfies the following definition, then it implies that the service is not necessary for controlling execution, and $P$ is set to 0.

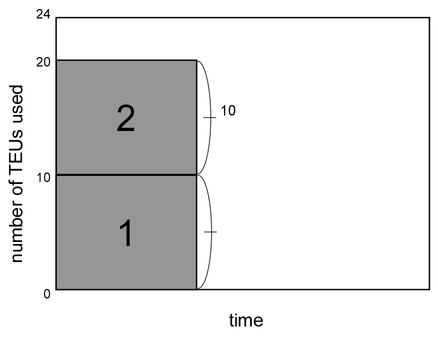
$$E \geq \sum_{i=1}^{j} m_i \times N > E - m_{j+1} \times N \tag{1}$$

4) In a service with the next priority, the software scheduler equally controls the *TEUs* available for each service. If the following is defined,

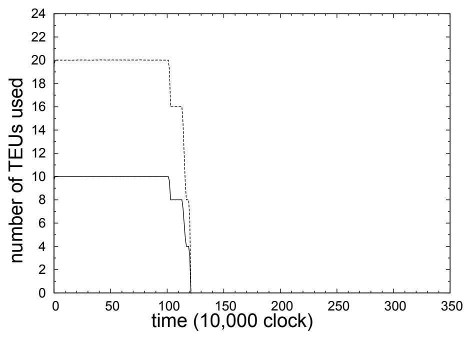
$$E_j = \sum_{i=1}^{j} m_i \times N \tag{2}$$

then the software scheduler evenly controls the number of *TEUs* available in each $m_{j+1}$ service. The following definition gives the number of *TEUs* available for these services.

$$\frac{E - E_j}{m_{j+1}} \tag{3}$$



(A) Ideal.



(B) Measurements.

Fig. 4. The number of *TEUs* used by 2 controlled services ($AP1 = AP2$).

This is the target value for control of the number of *TEUs* used. In order to control the number of *TEUs* used by the service to be close to the target value, we use the result of the evaluation of control for single service. Then, the average number of *TEUs* used by the service is given by the following expression.

$$N_{av} = \frac{1}{ET}\sum_{t=1}^{ET} N_t \tag{4}$$

Here, $ET$ is the execution time of the service, and $N_t$ is the number of *TEUs* used by the service in $t$ clocks ($1 \leq t \leq ET$). By setting $P$ to 1 and $M$ to an appropriate value, the software scheduler controls the number of *TEUs* used by the service to be close to the target value.

5) In services with priority lower than $p_{k+1}$, the control parameters are set to $P = 1$ and $M = 0$. Then, the state of the thread of these services is in waiting for *continuation* and cannot begin execution.

When the service with the highest priority finishes execution, the software scheduler resets the control parameters again.

## IV. Evaluation

### A. Target Service

As a target service for evaluation, we used the 10-Queens Problem Solver, which is a typical example for the evaluation of thread parallelism. We had described the evaluation of control of a 10-Queens Problem Solver in [7], and we have used this result of evaluation in scheduling multiple services.

### B. Point of View

In the evaluation, we describe the measurement and also the case when the software scheduler controls multiple 10-Queens Problem Solvers on the basis of priority.

Table I shows the combination of the priorities of the services. Following are the different numbers of services considered: 2, 3, and 5. In the table, multiple services are denoted by $AP1, AP2, \cdots$. The column *Priority* lists the magnitudes of priority for each service. The last column of the table, *Setting of the control parameters*, lists the settings of the control parameters on the basis of the priority of each service.

We evaluate our proposed method on the basis of the following viewpoints.

1) The total degree of parallelism is lower than $E$.
2) The total degree of parallelism is higher than $E$.
    a) Each service is executed evenly.
    b) A service is given high priority and each service is executed on the basis of priority.
    c) Two services are given high priority and each service is executed on the basis of priority.
    d) More than three services are given high priority and each service is executed on the basis of priority.

### C. Consideration

We used an ordinary PC and installed our in-house *Fuce* Software Simulator, which has 24 *TEUs*. In this section, we show the number of *TEUs* used by multiple controlled services, as shown in Figure 4. In addition, the points shown in the figure indicates the average number of *TEUs* used between 10000 clocks. To evaluate the controllability, we considered a criterion, namely $V(S)$. $V(S)$ is the variance number of *TEUs* used, and it indicates the stability of our scheduler. The following expressions define $V(S)$.

$$s_t = \frac{N_t}{N_{av}} \tag{5}$$

$$V(S) = \frac{1}{ET}\sum_{t=1}^{ET}(s_t - \bar{s})^2 \tag{6}$$

If our scheduler can limit $V(S)$ to a low value, then it implies that the variance of execution throughput is small, and it is easy to control execution when many thread groups run simultaneously.

*1) When the total degree of parallelism is lower than $E$:* Figure 4 shows the evaluation of control for 2 services ($AP1 = AP2$), which is indicated by number 1 in Table I. This evaluation shows the case where the total degree of parallelism is lower than the total number of *TEUs* $E$. The total degree of parallelism is $2 \times 10 = 20$, and this value is lower than $E = 24$. Therefore, the number of *TEUs* used by each service is 10. The average number of used *TEUs* is $N_{av} = 9.52$, and it is close to the degree of parallelism $N = 10$. In addition, $V(S) = 0.021$.

TABLE I
COMBINATION OF PRIORITIES.

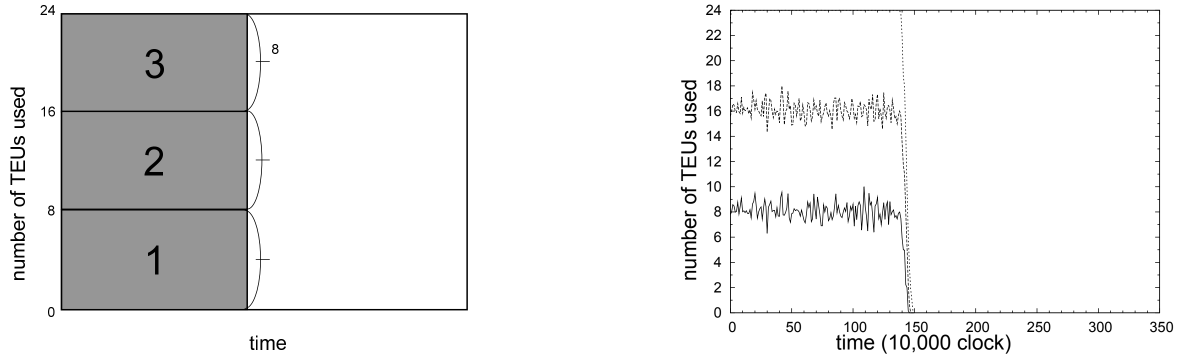
| Number | # of services | Priority | Setting of the control parameters |
|---|---|---|---|
| 1 | 2 | $AP1 = AP2$ | (default) $AP1, AP2$:$P = 0$ |
| 2 | 3 | $AP1 = AP2 = AP3$ | (default) $AP1$ to $AP3$:$P = 0$ |
| 3 | | $AP1 > AP2 = AP3$,<br>$AP1 > AP2 > AP3$ | (default) $AP1$:$P = 0$, $AP2, AP3$:$M = 10$(target value is 7, and control $N_{av} = 5.80$)<br>(finished $AP1$) $AP2, AP3$:$P = 0$ |
| 4 | | $AP1 = AP2 > AP3$ | (default) $AP1, AP2$:$P = 0$, $AP3$:$M = 6$(target value is 4, and control $N_{av} = 3.58$)<br>(finished $AP1, AP2$) $AP3$:$P = 0$ |
| 5 | 5 | $AP1 = AP2 = AP3 = AP4 = AP5$ | (default) $AP1$ to $AP5$:$P = 0$ |
| 6 | | $AP1 > AP2 = AP3 = AP4 = AP5$ | (default) $AP1$:$P = 0$, $AP2$ to $AP5$:$M = 5$(target value is 3.5, and control $N_{av} = 2.98$)<br>(finished $AP1$) $AP2$ to $AP5$:$P = 0$ |
| 7 | | $AP1 = AP2 > AP3 = AP4 = AP5$,<br>$AP1 > AP2 > AP3 = AP4 = AP5$ | (default) $AP1, AP2$:$P = 0$, $AP3$ to $AP5$:$M = 2$(target value is 2, and control $N_{av} = 1.23$)<br>(finished $AP1, AP2$) $AP3$ to $AP5$:$P = 0$ |
| 8 | | $AP1 = AP2 = AP3 > AP4 = AP5$,<br>$AP1 = AP2 = AP3 = AP4 > AP5$,<br>$AP1 = AP2 = AP3 > AP4 > AP5$ | (default) $AP1$ to $AP3$:$P = 0$, $AP4, AP5$:$M = 0$<br>(finished $AP1$ to $AP3$) $AP4, AP5$:$P = 0$ |
| 9 | | $AP1 > AP2 = AP3 > AP4 = AP5$,<br>$AP1 > AP2 = AP3 > AP4 > AP5$ | (default) $AP1$:$P = 0$, $AP2, AP3$:$M = 10$(target value is 7, and control $N_{av} = 5.80$), $AP4, AP5$:$M = 0$<br>(finished $AP1$) $AP2, AP3$:$P = 0$, $AP4, AP5$:$M = 3$(target value is 2, and control $N_{av} = 1.82$)<br>(finished $AP2, AP3$) $AP4, AP5$:$P = 0$ |
| 10 | | $AP1 > AP2 = AP3 = AP4 > AP5$ | (default) $AP1$:$P = 0$, $AP2$ to $AP4$:$M = 7$(target value is 4.67, and control $N_{av} = 4.12$), $AP5$:$M = 0$<br>(finished $AP1$) $AP2$ to $AP4$:$P = 0$, $AP5$:$M = 0$<br>(finished $AP2$ to $AP4$) $AP5$:$P = 0$ |
| 11 | | $AP1 = AP2 > AP3 > AP4 = AP5$,<br>$AP1 > AP2 > AP3 > AP4 = AP5$ | (default) $AP1, AP2$:$P = 0$, $AP3$:$M = 6$(target value is 4, and control $N_{av} = 3.58$), $AP4, AP5$:$M = 0$<br>(finished $AP1$,$AP2$) $AP3$:$P = 0$, $AP4, AP5$:$M = 10$(target value is 7, and control $N_{av} = 5.80$)<br>(finished $AP3$) $AP4, AP5$:$P = 0$ |
| 12 | | $AP1 = AP2 > AP3 = AP4 > AP5$,<br>$AP1 > AP2 > AP3 = AP4 > AP5$ | (default) $AP1, AP2$:$P = 0$, $AP3, AP4$:$M = 3$(target value is 2, and control $N_{av} = 1.82$), $AP5$:$M = 0$<br>(finished $AP1$,$AP2$) $AP3, AP4$:$P = 0$, $AP5$:$M = 6$(target value is 4, and control $N_{av} = 3.58$)<br>(finished $AP3$,$AP4$) $AP5$:$P = 0$ |
| 13 | | $AP1 > AP2 > AP3 > AP4 > AP5$,<br>$AP1 = AP2 > AP3 > AP4 > AP5$ | (default) $AP1, AP2$:$P = 0$, $AP3$:$M = 6$(target value is 4, and control $N_{av} = 3.58$), $AP4, AP5$:$M = 0$<br>(finished $AP1$,$AP2$) $AP3, AP4$:$P = 0$, $AP5$:$M = 6$(target value is 4, and control $N_{av} = 3.58$)<br>(finished $AP3$) $AP4, AP5$:$P = 0$<br>(finished $AP4$) $AP5$:$P = 0$ |

*2) When the total degree of parallelism is higher than $E$:* Here, we consider the case in which the total degree of parallelism is higher than $E$. Figure 5 shows the execution of each of three services ($AP1 = AP2 = AP3$) evenly, which is indicated by number 2 in Table I. In addition, Figure 6 shows the execution of each of the 5 services ($AP1 = AP2 = AP3 = AP4 = AP5$) evenly, which is indicated by number 5 in Table I. We can observe the following in Figures 5 and 6.

1) By setting $P$ to 0 for $m_i$ number of $p_i$ services, the software scheduler can execute each service evenly. In Figure 5, each of the 3 services uses 8 *TEUs*, and in Figure 6, each of the 5 services uses 4.8 *TEUs*. This reason is that each service uses the same 10-Queens Problem Solver.
2) When the total degree of parallelism is higher than $E$, $V(S)$ for each service is high. In particular, $V(S)$ increases with the number of the services. The value of $V(S)$ for $AP1$ shown in Figure 5 is 0.025, which is higher than that shown in Figure 4, which is 0.021. The value of $V(S)$ for $AP1$ shown in Figure 6 is 0.112, which is higher than that shown in Figure 5, which is 0.025.

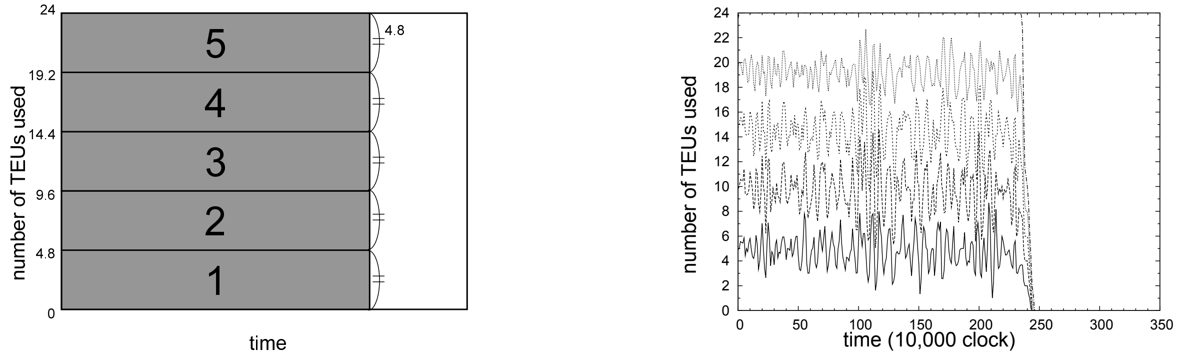
Next, we consider a case where a service is given high priority and then each service is executed on the basis of priority. Figure 7 shows the execution of each of the 3 services ($AP1 > AP2 = AP3$) on the basis of priority, which is indicated by number 3 in Table I. Figure 8 shows the execution of each 5 services ($AP1 > AP2 = AP3 = AP4 = AP5$) on the basis of priority, which is indicated by number 6 in Table I. We can observe the following in Figures 7 and 8.

3) By setting $P = 1$ and $M$ to an appropriate value, the software scheduler can control the number of used *TEUs* to be close to the target value. In Figure 7, the target



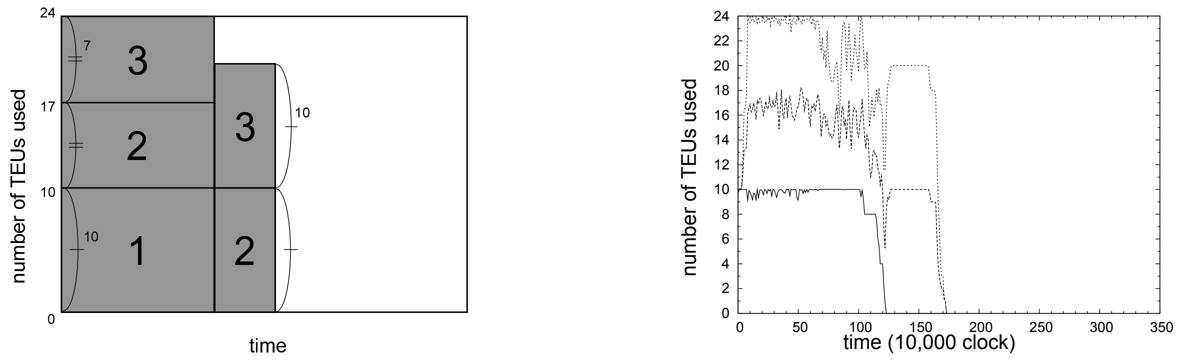
(A) Ideal. (B) Measurements.

Fig. 5. The number of *TEUs* used by 3 controlled services ($AP1 = AP2 = AP3$).



(A) Ideal. (B) Measurements.

Fig. 6. The number of *TEUs* used by 5 controlled services ($AP1 = AP2 = AP3 = AP4 = AP5$).



(A) Ideal. (B) Measurements.

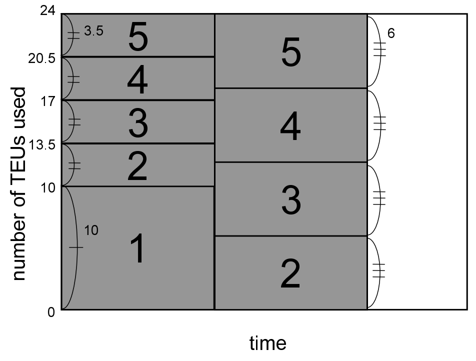
Fig. 7. The number of *TEUs* used by 3 controlled services ($AP1 > AP2 = AP3$).

value of $AP2$ and $AP3$ is 7 while executing $AP1$. By controlling the scheduling method, these services execute with $N_{av} = 5.80$, which is close to the target value of 7. Then, the number of the control $M$ is set to 10 for $AP2$ and $AP3$. In Figure 8, the target value of $AP2$ to $AP5$ is 3.5 while executing $AP1$. By controlling the scheduling method, these services execute with $N_{av} = 2.98$, which is close to the target value 3.5. Then $M$ is set to 5 for $AP2$ to $AP5$.

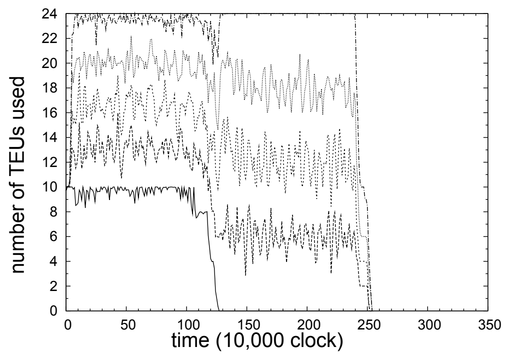
4) Values of $V(S)$ for $AP1$ is 0.112 and 0.036 in Figures 7 and 8, respectively. This implies that the number of *TEUs* used by a service of high priority is not limited by other services.

We consider the case in which 2 services are given high priority and each service is executed on the basis of priority. Figure 9 shows the execution of each of 3 services ($AP1 = AP2 > AP3$) on the basis of priority, which is indicated by number 4 in Table I. Figure 10 shows the execution of each of the 5 services ($AP1 = AP2 > AP3 = AP4 = AP5$) on the basis of priority, which is indicated by number 7 in Table I. We can observe the following in Figures 9 and 10.

5) For the case in which 2 services are given high priority

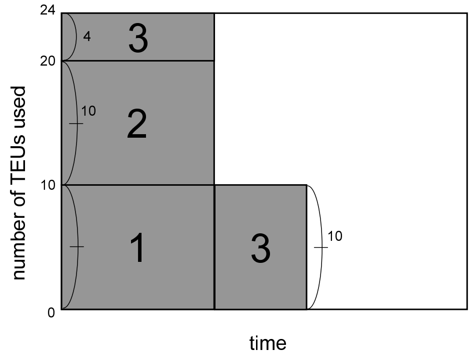


(A) Ideal.

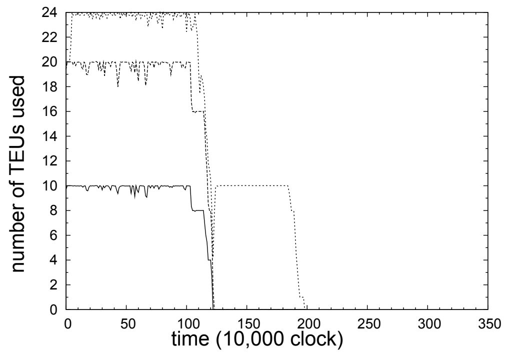


(B) Measurements.

Fig. 8. The number of *TEUs* used by 5 controlled services ($AP1 > AP2 = AP3 = AP4 = AP5$).

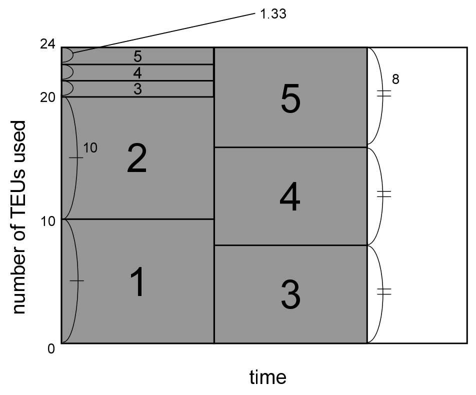


(A) Ideal.

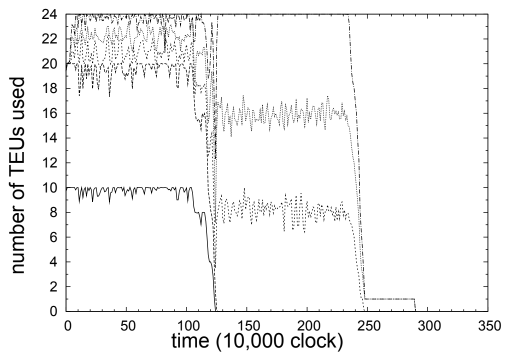


(B) Measurements.

Fig. 9. The number of *TEUs* used by 3 controlled services ($AP1 = AP2 > AP3$).



(A) Ideal.



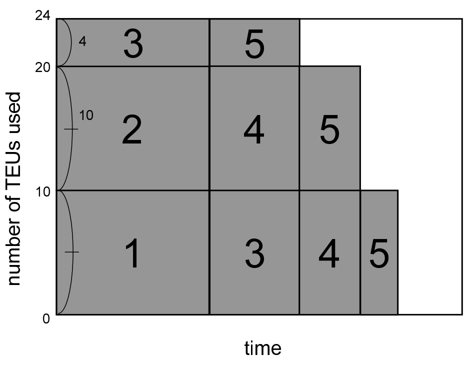
(B) Measurements.

Fig. 10. The number of *TEUs* used by 5 controlled services ($AP1 = AP2 > AP3 = AP4 = AP5$).

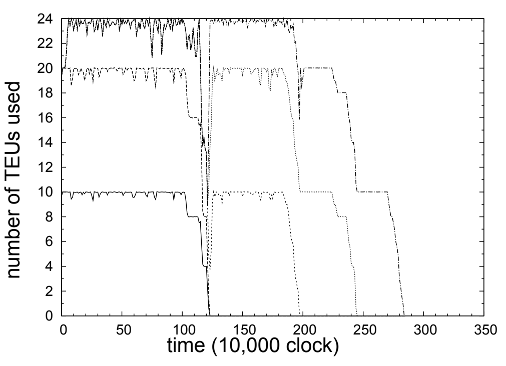
and each service is executed on basis of priority, the number of *TEUs* used by the service of high priority is not limited by other services. In Figure 9, $AP1$ and $AP2$ are the services with high priority. $N_{av}$ of $AP1$ is $9.44$ and that of $AP2$ is $9.37$; these results imply that the service with high priority can execute with a degree of parallelism of approximately 10. In addition, $V(S)$ of $AP1$ is $0.025$ and $AP2$ is $0.030$, and both of these values are small.

Finally, we consider the case in which more than 3 services are given high priority and each service is executed on the basis of priority. Figure 11 shows the execution of each of the 5 services ($AP1 > AP2 > AP3 > AP4 > AP5$) on the basis of priority, which is indicated by number 13 in Table I. We can observe the following in Figure 11.

6) In this figure, the scheduling method can control the execution of 5 services on basis of priority.



(A) Ideal.



(B) Measurements.

Fig. 11. The number of *TEUs* used by 5 controlled services $(AP1 > AP2 > AP3 > AP4 > AP5)$.

## V. Related Work

Scheduling Support Hardware (SSH) [8] is realized as dedicated hardware that performs task scheduling. In addition, SSH was adapted to Linux and evaluated [9]. In this evaluation, the risk was described that adapting the fine-grained parallel-computation model to a general-purpose operating system such as Linux causes an increase in the overhead of a context switch. As the *Fuce* processor has preload units that enable to perform a lookahead, it can reduce the overhead of the context switch.

In the execution of multiple services, gang scheduling [10][11] is an effective scheduling method in case the number of threads executing at the same time is larger than the number of *TEUs*. However, in the execution of multiple services, when the number of threads in a single service is less than the number of *TEUs*, gang scheduling cannot use *TEUs* effectively because occupying all the *TEUs* by single service in time slice causes wastage of the use of *TEUs*. We considered this viewpoint and designed our scheduling method to prevent this wastage.

## VI. Conclusion

We proposed and evaluated a method for CMP-oriented thread scheduling based on the *continuation* model. The goal of the scheduling method is to limit the number of *TEUs* used by each service and control the execution of each service on the basis of priority. According to the priority of each service, the software scheduler sets the control probability $P$, the activation period $\omega$, and the number of the control $M$.

In evaluation, we showed the results of the measurement and also controlled the execution of multiple 10-Queens Problem Solvers on the basis of priority of each service. We evaluated the scheduler from the viewpoint of the following cases: when the total degree of parallelism is lower than the total number of *TEUs*; when each service is executed evenly; and when one, two, or more than two services are given high priority and each service is executed on the basis of priority. In this evaluation, we showed that our scheduling method can control the execution of services on the basis of priority, and the service with high priority does not limit the number of *TEUs* used by the execution of a service with a low priority.

## References

[1] Shigeru Kusakabe, Yoshinari Nomura, Hideo Taniguchi, and Makoto Amamiya, “Wrapped System-Call: Cooperating Interactions between User and Kernel Mode in an Operating System for Fine-Grain Multi-Threading, ” Proc. of the 21st IASTED International Multi-Conference on Applied Informatics, pp.656-661, 2003.

[2] Ben Lee, A.R.Hurson, “Dataflow Architectures and Multithreading, ” IEEE COMPUTER, vol.27, no.8, pp.27-39, 1994.

[3] Shigeru Kusakabe, Satoshi Yamada, Mitsuhiro Aono, Masaaki Izumi, Satoshi Amamiya, Yoshinari Nomura, Hideo Taniguchi, and Makoto Amamiya, “OS Mechanism for Continuation-based Fine-grained Threads on Dedicated and Commodity Processors, ” Proc. of Workshop on Multi-Threaded Architectures and Applications, published in CD, 2007.

[4] Makoto Amamiya, Hideo Taniguchi, and Takanori Matsuzaki, “An architecture of fusing communication and execution for global distributed processing,” Parallel Processing Letters, vol.11, no.1, pp.7-24, 2001.

[5] Takanori Matsuzaki, Satoshi Amamiya, Masaaki Izumi, and Makoto Amamiya, “A Multi-thread Processor Architecture Based on the Continuation Model,” Proc. of 8th Innovative Architecture for Future Generation High-Performance Processors and Systems (IWIA05), pp.83-90, 2005.

[6] Hideaki Moriyama, Yoshinari Nomura, and Hideo Taniguchi, “A Method for CMP-oriented Thread Scheduling Based on Continuation Model,” Proc. of the 2009 2nd International Conference on Computer Science and its Applications (CSA2009), vol.2, pp.697-702, 2009.

[7] Hideaki Moriyama, Yoshinari Nomura, and Hideo Taniguchi, “Evaluation of the S/W Scheduler for CMP Based on Continuation Model,” IPSJ SIG Technical Report, vol.110, no.278, pp.11-16, 2010, (in Japanese).

[8] Takahiro Sasaki, Tetsuo Hironaka, Naoki Nishimura, and Seiji Fujino, “Microprocessor LSI with Scheduling Support Hardware for Operating System on Multiprocssor System, ” The 6th Asia Pacific Conference on cHip Design Languages(APCHDL’99), pp.67-72, 1999.

[9] Kazuki Ohara, Takahiro Sasaki, Kazuhiko Ohno, Toshio Kondo, “Acceleration for fine grained parallel processing on Linux with hardware scheduler,” IPSJ SIG Technical Report, 2006-ARC-170, vol.2006, no.127, pp.25-30, 2006, (in Japanese).

[10] D.G. Feitelson and L. Rudolph, “Gang Scheduling Performance Benefits for Fine Grain Synchronization,” Journal of Parallel and Distributed Computing, vol.16, no.4, pp.306-318, 1992.

[11] A. Gupta, A. Tucker, and Shigeru Urushibara, “The Impact of Operating System Scheduling Policies and Synchronization Methods on the Performance of Parallel Applications,” ACM SIGMETRICS, pp.120-132, 1991.