Bulletin of Networking, Computing, Systems, and Software – www.bncss.org, ISSN 2186-5140
Volume 13, Number 1, pages 12–15, January 2024

A Design Scheme for Highly Efficient Mixed-Criticality Systems Using IPC Control

1st Kosuke Yashima
Graduate School of Science and Technology
Keio University
Kouhoku-ku, Yokohama-shi, kanagawa, Japan, 223-8852
yashima@ny.ics.keio.ac.jp

2nd Nobuyuki Yamasaki
Graduate School of Science and Technology
Keio University
Kouhoku-ku, Yokohama-shi, kanagawa, Japan, 223-8852
yamasaki@ny.ics.keio.ac.jp

***Abstract*—In a safety-critical real-time system, the system must keep running even when the task execution time is longer than expected. The Mixed Criticality (MC) system can satisfy the time constraint of the hi task by discarding the low-priority task (lo task) and allocating computing resources to the high-priority task (hi task) in an emergency when the amount of data to be processed increases significantly. However, discarding the lo task may degrade the QoS of the system. In this paper, Fluid scheduling is implemented in the MC system using an IPC control mechanism that controls the execution speed of each thread in the SMT (Simultaneous Multithreading) processor, and the number of lo tasks discarded in an emergency is reduced. The proposed method in this paper successfully maintains the high QoS of the MC system.**

***Index Terms*—real-time scheduling, mixed criticality, fluid scheduling, SMT processor**

I. Introduction

In real-time systems, the value of processing depends not only on the completion of task execution but also on meeting time constraints. Real-time scheduling determines the order of task execution by determining the priority of tasks from the time constraints of the tasks to satisfy the time constraints. Fluid scheduling is one of the optimal real-time scheduling methods applicable to multiprocessors [1]. Fluid scheduling is a scheduling algorithm that processes tasks at a constant execution rate from their release until their deadline. However, Fluid Scheduling requires controlling the execution speed of threads, which is not feasible on general processors due to the lack of a mechanism to control the execution speed. Fluid scheduling can be realized by using a mechanism to control the execution speed of each thread in SMT (Simultaneous multi-threading) processors called the IPC control mechanism [2].

In recent years, mixed-criticality (MC) systems have attracted attention in real-time systems, which can cope with systems where the execution time of critical tasks varies. Here, a critical task is a task that would cause a severe loss to the system if the time constraint is not kept. Such a system is exemplified by air traffic control, where the execution time of the air traffic control task varies depending on the number of aircraft. In an emergency where the execution time of a task with high importance is extended, and the time constraint cannot be kept, MC systems can satisfy the time constraint and execute the task by cutting off the less important task and securing the execution time of the more critical task. Lee and his teams proposed MC-Fluid as a multiprocessor MC system using optimal real-time scheduling [3]. Since MC-Fluid uses Fluid scheduling, it cannot be applied to natural systems on general processors where the execution speed of tasks cannot be controlled. In addition, the QoS of the entire system is lowered due to the truncation of less critical tasks.

This paper applies MC-Fluid to a real system by executing tasks based on MC-Fluid using the IPC control mechanism. In addition, by using surplus computing resources, MC-Fluid can execute as many less important tasks as possible, which are discarded in the MC system in an emergency.

This paper is organized as follows. Chapter 2 describes the background of this paper, MC system, fluid scheduling, and IPC control mechanism. Chapter 3 discusses related research. Chapter 4 describes the proposed method. In Section 5, the effectiveness of the proposed method is evaluated and discussed. Finally, Section 6 summarizes the paper.

II. Background

A. Fluid scheduling

Fluid scheduling is an optimal scheduling method for multiprocessors [**?**] because it ensures that the processors always process tasks at a constant execution rate from the time of task release to the deadline, which results in 100% processor utilization. Let C_i denote the remaining execution time of a task τ_i whose deadline coincides with the cycle, C_i the cycle T_i, and r_i the arrival time, Fluid scheduling runs tasks at a constant speed from C_i to $r_i + T_i$. The execution speed θ_i based on Fluid Scheduling can be expressed as follows.

$$\theta_i = \frac{C_i}{T_i} \tag{1}$$

Figure 1 shows an example of Fluid Scheduling. The vertical and horizontal axes represent the remaining execution time and time, respectively. The vertical and horizontal axes are scaled the same. Scheduling on a general processor is shown by a solid line, while a red dotted line shows Fluid scheduling. It can be seen that Fluid scheduling runs at a constant execution speed from the remaining execution time to the deadline. However, since Fluid Scheduling requires a

Identify applicable funding agency here. If none, delete this.

constant thread execution speed, a general processor cannot follow the straight line of Fluid Scheduling.

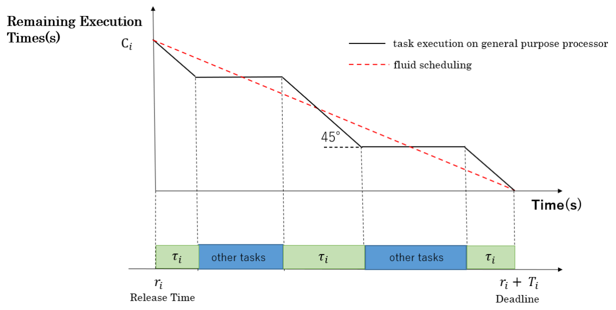

Fig. 1. execution exaple of fluid scheduling

Therefore, pseudo-fluid scheduling, such as P-fair, has been proposed to achieve linear execution of fluid scheduling on general processors [4]. P-fair reproduces the same execution as Fluid Scheduling by repeatedly executing and stopping tasks. In this case, the slope is 45 degrees when the task is executed and 0 degree when it is stopped. However, P-fair scheduling has a disadvantage in that there are many context switches, and the overhead is significant.

B. Mixed Criticality System

In a Mixed Criticality (MC) system, tasks and system modes have a criticality parameter. Tasks that cause significant damage to the system in the event of a deadline error are given a higher priority. For simplicity, we assume two types of criticality, LO and HI. In this case, a task is classified as either a low-priority LO task or a high-priority HI task. There are two system modes: LO mode for normal mode and HI mode for emergency mode. The MC system defines several worst-case execution times (WCET) for hi tasks, assuming task execution times vary depending on the situation. In the typical case, LO_WCET(C_i^L) is used. When the execution time of the hi task is extended in an emergency, the WCET is extended to HI_WCET(C_i^H), which is longer than LO_WCET, to keep the time constraint of the hi task.

Next, the behavior of the MC system is described. First, the system operates in LO mode, which is the normal mode. In LO mode, both lo and hi tasks are executed. When the execution time of the hi task exceeds C_i^L, the system transitions to HI mode. In HI mode, the lo task is discarded, and only the hi task is executed using the remaining computing resources to keep the time constraint of the hi task. In this case, the WCET of the hi task is extended to HI_WCET. The system administrator sets the condition for returning from HI mode to LO mode, which is often determined on a Utilization basis. WCET of the hi task returns to LO_WCET. In a standard MC system, all lo tasks are discarded when the system transitions to HI mode, which reduces the QoS of the entire system.

C. IPC Control Scheme

IPC (Instruction Per Clock cycle) is the number of instructions executed per clock cycle. The IPC control scheme controls the execution speed of each thread of the SMT processor. This scheme improves the predictability of task execution time and scheduling accuracy. The operation procedure of the IPC control is as follows: First, the software inputs the target IPC and the control period. Then, the IPC of the control cycle is observed, the difference between the observed IPC and the target IPC is calculated, and the number of instruction fetches is controlled so that the target IPC is approached in the next cycle. This kind of feedback control brings the IPC closer to the target IPC.

III. Related Works

A. MC-Fluid

MC-Fluid is an MC system that uses fluid scheduling, an optimal multiprocessor scheduling [3]. MC-Fluid defines two execution rates as well as a system mode. When the remaining execution time is less than or equal to C_i^L, the execution follows the execution rate of the LO mode (θ_i^L), and no mode transition occurs. However, when the actual execution time exceeds C_i^L, a mode transition occurs, and the mode becomes HI mode. In this case, all lo tasks are discarded, and the hi task uses the freed computational resources, which runs at a faster execution speed (θ_i^H).

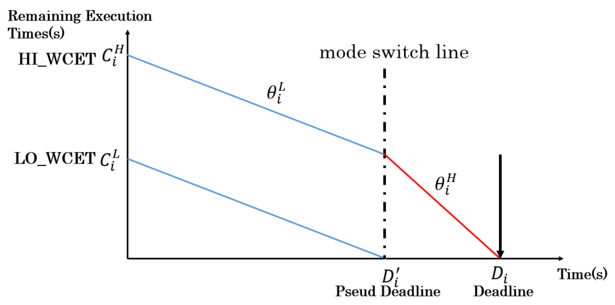

Fig. 2. execution exaple of MC-Fluid

Figure 2 shows an example of MC-Fluid execution. The blue line represents the execution speed θ_i^L in LO mode, and the red line represents the execution speed θ_i^H in HI mode. In addition, a pseudo-deadline$D_i^{'}$ is set for the transition condition from LO mode to HI mode. Since the product of execution speed and execution time represents the amount of execution, let $E_i(t)$ denote the amount of execution after t seconds of execution, which is represented by the following equation.

$$E_i(t) = \theta_i \times t \tag{2}$$

Also, since the execution speed θ_i^L is $\theta_i^L < 1$

$$E_i(C_i^L) = \theta_i^L \times C_i^L < C_i^L \tag{3}$$

At time C_i^L, the execution volume is less than C_i^L. By defining the pseudo-deadline D_i', the execution volume becomes C_i^L at the D_i'.

$$D_i' = \frac{C_i^L}{\theta_i^L} \tag{4}$$

If the sum of lo tasks utilization in LO mode is U_L^L, the sum of hi tasks utilization in LO mode is U_H^L. The sum of the utilization of hi tasks in HI mode is U_H^H, the execution speed is defined as follows, and it is possible to keep the time constraint after the mode transition when the execution speed is given like this [5].

$$\rho = \max\{ \frac{U_L^L + U_H^L}{m}, \frac{U_H^H}{m}, \max_{\tau_i \in \tau_H} \{u_i^H\} \} \tag{5}$$

$$\theta_i^H = \frac{u_i^H}{\rho} \tag{6}$$

$$\theta_i^L = \{ \frac{u_i^L \theta_i^H}{\theta_i^H - (u_i^H - u_i^L)} \} \quad (\tau_i \in \tau_{HI}) \tag{7}$$

$$\theta_i^L = u_i^L \quad (\tau_i \in \tau_{LO}) \tag{8}$$

B. Fluid Scheduling by Using IPC Control Scheme

Fluid scheduling requires controlling the execution speed of tasks for each thread, but general processors cannot realize Fluid scheduling because they cannot control the thread execution speed. Fluid scheduling using the IPC control mechanism (IPC-Fluid) [6] controls the thread execution speed of SMT processors to enable task execution according to Fluid scheduling. In IPC-Fluid, the remaining execution clock cycles are set to WCEC at the beginning of each cycle and executed at a constant speed. While WCET is used for general scheduling, WCIN (worst-case Case Instruction Number) is used for Fluid scheduling with the IPC control mechanism. The target IPC (IPC_target) is expressed as follows using WCIN.

$$IPC_{target} = \frac{WCIN}{WCEC} \tag{9}$$

IV. IPC-MC-Fluid

In order to realize the MC-Fluid mentioned above in a natural system, we propose IPC-MC-Fluid, which enables IPC-Fluid to be applied to MC-Fluid. In addition, the conventional MC-Fluid could not execute lo tasks in HI mode, but we have made it possible to execute lo tasks in HI mode by using extra computing resources.

A. Design of IPC-MC-Fluid

The execution speeds in the IPC-MC-Fluid LO and HI modes are the same as those proposed for MC-Fluid, respectively. The parameter required for IPC-Fluid control is Target_instruction. Target_instruction is calculated by IPC control period and execution speed as follows.

$$Target_{instruction} = \theta \times IPC_{controlperiod} \tag{10}$$

As in MC-Fluid, the transition condition from LO mode to HI mode is when the hi task has not finished executing when the pseudo-deadline is reached. In this case, the lo task that can be executed in HI mode is selected, and the lo task that is not selected is discarded. Lo task selection is determined by the algorithm described below.

Algorithm 1 Selecting lo task to execute in HI mode

```
1:  $i \leftarrow 0$
2:  $IPC_{remaining} \leftarrow IPC_{max} - U_H^H$
3:  Sort lo tasks in order of decreasing IPC
4:  while $IPC_{remainig} > 0$ do
5:      if $\tau_i \in \tau_i^{LO}$ then
6:          $IPC_{remaining} \leftarrow IPC_{remaining} - u_i^{LO}$
7:          $i \leftarrow i + 1$
8:      end if
9:  end while
10: for ; $i < TASK\_MAX; i \leftarrow i + 1$ do
11:     $\tau_i$ into dropped queue
12: end for
```

The transition condition from HI mode to LO mode is when the sum of Utilization in the current period ($U_{current}$) falls below $U_L^L + U_H^L$, the sum of Utilization of all tasks in LO mode. This judgment is made every hyper period of the task set.

B. Selection of lo task to execute in HI mode

We propose an algorithm to execute as many lo tasks as possible in HI mode. If the total Utilization of hi tasks in HI mode is UHH, the excess IPC in HI mode (IPC_remaining) can be calculated as follows:

$$IPC_{remaining} = IPC_{max} - U_H^H \tag{11}$$

IPC_max is the maximum total IPC the system can accept. If the IPC of a lo task is less than or equal to IPC_remaining, the lo task can be executed in HI mode. Lo tasks that meet this condition are selected in order of decreasing IPC, and each time, the IPC of the selected lo task is subtracted from IPC_remaining. This operation is repeated until IPC_remaining is less than or equal to 0.

V. Evaluation

This section compares the schedulability performance and lo task execution rate between the proposed method IPC-MC-Fluid and other MC system multiprocessor scheduling. G-EDF was selected as the multiprocessor scheduling for comparison. First, we describe the evaluation environment.

A. Evaluation Environment

In this study, RMTP (Responsive Multi-threded Processor), an SMT processor for embedded real-time systems, is the implementation target. RMTP is an 8-way SMT processor with eight hardware logic cores. The configuration of RMTP is shown in Table 1. The evaluation was performed by mapping the SRMTP described above to the Virtex UltraScale+ HBM VCU128 FPGA evaluation kit. The Benchmark used was Mibench's qsort,bitcount.

Bulletin of Networking, Computing, Systems, and Software – www.bncss.org, ISSN 2186-5140
Volume 13, Number 1, pages 12–15, January 2024

TABLE I
CONFIGURATION OF RMTP

papameter	value
ALU	4
FPU	2
FPU Div	1
Load/Store Unit	1
64bit SIMD	1
Branch Unit	2
Vector Int	1
Vector FP	1
Active threads	8 threads
Fetch Width	8 inst from 1 thread
Issue/Commit Width	4 inst
Reservation Station (Int, FP, Mem/Br)	16 entry each
Reorder Buffer	16 entry each
Instruction Cache	8way set associative Cache argorithm:LRU / priority
Data Cache	8way set associative Cache argorithm:LRU / priority

B. Schedulability Performance

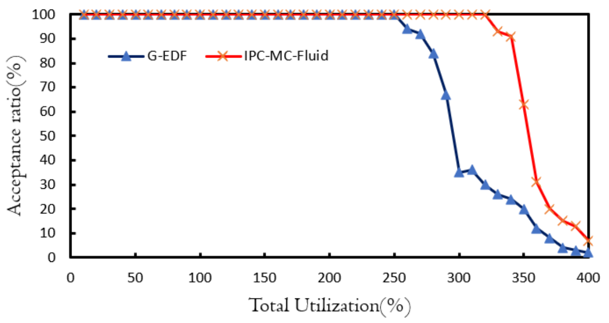

Fig. 3. Schedulability evaluation

In this section, we evaluate the schedulability, which indicates to what extent the total utilization of the task set can be executed within the time constraints. The figure 3 shows the schedulability evaluation of G-EDF and IPC-MC-Fluid. The vertical axis represents the Acceptance ratio, and the horizontal axis represents the total Utilization of the task set. The acceptance ratio is calculated using the following formula.

$$acceptance\ raio = \frac{\sharp Num\ of\ task\ sets\ with\ no\ deadline\ misses}{\sharp Num\ of\ generated\ task\ sets} \tag{12}$$

G-EDF was able to schedule up to 250% utilization without missing deadlines. The proposed method, IPC-MC-Fluis, could schedule up to 320% utilization without missing deadlines.

C. lo task execution rates

The figure shows the evaluation result of lo task execution rates. lo, task execution rates are calculated by the following equation.

$$acceptance\ raio = \frac{\sharp Num\ of\ successfully\ executed\ lo\ tasks}{\sharp Num\ of\ generated\ lo\ tasks} \tag{13}$$

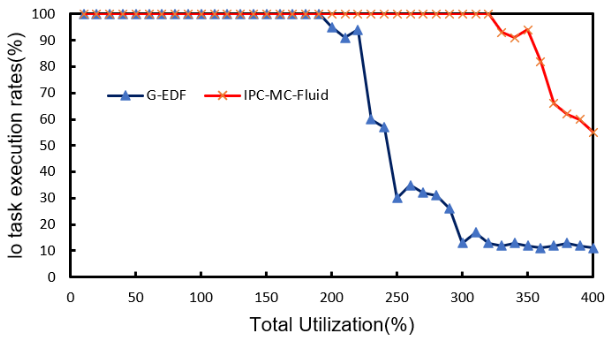

Fig. 4. lo task execution ratio

Since G-EDF discards all lo tasks during the mode switch, the more frequently the mode switch occurs, the fewer lo tasks are executed. The figure also shows that the execution rate drops from 200% utilization to about ten at 400% utilization.

On the other hand, IPC-MC-Fluid executes the lo task even in HI mode, so the lo task execution rates remain high compared to G-EDF.

VI. CONCLUSION AND FUTURE WORK

Because the execution speed cannot be arbitrarily controlled on general processors, fluid scheduling, an optimal multiprocessor scheduling method, cannot be realized, and MC-Fluid using fluid scheduling could not be realized on a natural system. In this paper, we control the execution speed of SMT processor threads by the IPC control mechanism to realize fluid scheduling and realize MC-Fluid, a highly efficient MC system, on a natural system. Compared with conventional multiprocessor MC systems, the proposed method shows improved schedulability. In addition, the execution rate of lo tasks was kept high, and the QoS of the whole system was maintained.

As a future issue, it is necessary to make MC systems viable even with three or more importance levels since few systems are completed with only two (LO and HI).

REFERENCES

[1] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel, "Proportionate progress: A notion of fairness in resource allocation," vol. 15, June 1996, pp. 600–625.
[2] K. Matsumoto, H. Umeo, and N. Yamasaki, "A thread control scheme for real-time microprocessors," in *2011 IEEE 17th International Confernce on Embedded and Real-Time Computing Systems and Applications*, vol. 2, 2011, pp. 16–21.
[3] J.Lee, S.Ramanathan, K.Phan, A.Easwaran, and I.Shin, "MC-Fluid: Multi-core fluid-based mixed-criticality scheduling," in *IEEE Transaction on Computers*, vol. 67, no. 4, April 2018, pp. 469–483.
[4] P. Holman and J. Anderson, "Adapting pfair scheduling for symmetric multiprocessors," in *2020 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS)*, June 2005, pp. 543–564.
[5] S. Baruah, A. Easwaran, and Z. Guo, "MC-Fluid: simplified and optimally quantified," in *IEEE Real-Time Systems Symposium*, December 2015, pp. 327–337.
[6] Y. Tsukahara and Y. Nobuyuki, "Implementation of fluid scheduling using ipc control mechanism," in *2019 IEEE 25th International Conferenceon Parallel and Distributed Systems (ICPADS)*, 2019, pp. 402–405.