U-Link Scheduling: Bounding Execution Time of Real-Time Tasks with Multi-Case Execution Time on SMT Processors

Shinpei Kato, Hidenori Kobayashi and Nobuyuki Yamasaki
School of Science for Open and Environmental Systems
Keio University
Kohoku, Yokohama City, Japan
{shinpei,kobayashi,yamasaki}@ny.ics.keio.ac.jp

Abstract

The goal of this paper is to achieve hard real-time processing with admitting as many tasks as possible on Simultaneous Multithreaded (SMT) processors. For this goal, we propose U-Link scheduling scheme that determines the co-scheduled set that is the fixed combinations of co-scheduled tasks to bound the task execution time. Also we present practical algorithms, RR-DUP for building co-scheduled sets and UL-EDF for task scheduling. The performance evaluation shows that UL-EDF with RR-DUP outperforms the conventional scheduling algorithms, EDF-FF and EDF-US, in the point of execution time stability, task rejection ratio and deadline miss ratio.

1 Introduction

Recent real-time systems such as humanoid robots require both real-time and high-performance processing. Due to the background above, the demand of parallel and distributed processing in real-time systems is increasing.

Simultaneous Multithreading (*SMT*) [6] combines a superscalar architecture with fine-grained multithreading. SMT with eight threads (8-way SMT) improves throughput double to three times over a conventional superscalar or multithreading [1]. Moreover, since it shares hardware resources among threads, it does not require as many hardware resources as *Symmetric Multiprocessor* (*SMP*) and *Multicore*. Therefore SMT processors are expected in real-time systems with limited resources. However, it is not straightforward to introduce SMT into such systems because of its characteristic *C-variability*; the task execution time fluctuates due to the variation in the combination of *co-scheduled tasks* (tasks running simultaneously).

In a system where hard real-time tasks reside, an admission control is required to guarantee whether those tasks can meet their deadline when they arrive at the system. In general, worst-case execution time (WCET) is usually taken as a computation time to give an admission control. However, because of the C-variability, it is difficult to predict a proper WCET on SMT processors so it possibly engages in underestimation and overestimation of WCET. It is apparent that underestimation of WCET causes missing the deadline. Overestimation of WCET avoids missing the deadline but results in low system utilization. Despite that SMT processors are becoming active now in the research and market, very little work has been done in this area.

In the past work, there was an attempt which applies the scheduling algorithms in multiprocessors to SMT processors [3]. However, as shown in the later section, the scheduling algorithms in multiprocessors as *EDF-FF* [5] and *EDF-US* [2] which are respectively the representatives of the partitioning scheme and the global scheduling scheme are not suitable on SMT processors since it cannot hold down the C-variability. This paper concerns real-time scheduling with SMT, which takes the C-variability into account. The goal of this paper is bounding the task execution time with admitting as many tasks as possible on SMT processors. We propose a new scheduling scheme *U-Link scheduling* in this paper which overcomes the issues in the conventional scheduling schemes.

The rest of this paper is organized as follows. The next section explains the system model in this paper. Section 3 describes U-Link scheduling including a practical algorithm, UL-EDF. Section 4 evaluates the performance of UL-EDF. In section 5, we conclude our work.

2 System Model

Suppose the followings in this paper. All the tasks are periodic and independent. Any task can join, leave and be preempted at any time. There are M threads executing simultaneously on an SMT processor. Task set τ including n

tasks $\{\tau_1, \tau_2, ..., \tau_n\}$ is given in the system at the beginning. Each task τ_i has a period T_i, computation time C_i, relative deadline D_i where $C_i \leq D_i = T_i$. The rate between T_i and C_i is called task utilization U_i, i.e. $U_i = C_i/T_i$.

3 U-Link Scheduling

In general, the task execution time fluctuates when the combinations of the co-scheduled tasks vary for every instance. Based on this foundation, we propose a new scheduling scheme, *Utilization-Link* (*U-Link*) *scheduling*, that bounds the task execution time by fixing the combinations of co-scheduled tasks base on the task utilization. Each fixed combinations are called *co-scheduled set*. Figure 1 shows the scheme of U-Link scheduling.

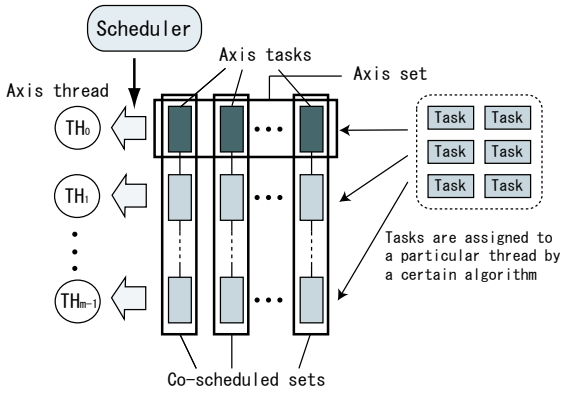

Figure 1. U-Link scheduling

The tasks are divided into u co-scheduled sets $\{\Gamma_1, \Gamma_2, ..., \Gamma_u\}$ by a certain method as presented in the later section. Also they are assigned to a particular thread. Each thread TH_j has its own task set θ_j. Each co-scheduled set Γ_i has at most M tasks where one of them is defined as axis task α_i. That is, there are u axis tasks $\{\alpha_1, \alpha_2, ..., \alpha_u\}$ in the system, so-called *axis set* α. In other words, one of $\theta_j : 1 \leq j \leq M$ becomes axis set α. The scheduler dispatches an axis task from α then tasks which run the rest of the threads, i.e. non-axis tasks, are chosen automatically since they are linked to the axis task in the co-scheduled set.

3.1 Co-Scheduled Set

First of all, we explain a method to assign tasks to dedicated threads and to build co-scheduled sets. The method we propose is *RR-DUP* (*Round Robin assignment in Decreasing Utilization Period*) which first sorts tasks in order of decreasing utilization where ties are broken in favor of shorter periods. Then it assigns the sorted tasks in sequence to threads in round-robin.

This RR-DUP has a drawback under the case that the axis task has a large utilization and the non-axis tasks have a small utilization. Since the utilization of non-axis task is regarded as that of the axis task in U-Link scheduling, the execution margins generate if there are differences and the system utilization is decreased as a result. For example, suppose axis task τ_1 and non-axis task τ_2 are in the same co-scheduled set where $C_1/T_1 = 3/5$ and $C_2/T_2 = 2/5$. Since no other task but τ_2 can execute during τ_1 executes, the execution margins appear every period in the execution of τ_2 (Figure 2).

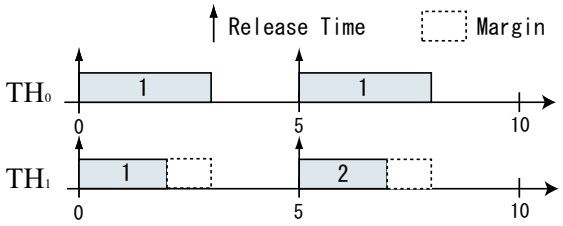

Figure 2. Generation of execution margins

In order to boost the system utilization, we allow each element of co-scheduled sets to consist of plural tasks. An element consisting of plural tasks is called *compound*. There are at most M compounds in co-scheduled set Γ_l where the kth compound is denoted by $\sigma_k(\Gamma_l)$. Each compound $\sigma_k(\Gamma_l)$ must satisfy Equation (1).

$$\sum_{\tau_i \in \forall \sigma_k(\Gamma_l)} U_i \leq (U_j : \tau_j = \alpha_l) \quad (1)$$

In the case of Figure 2, a task with a utilization less than or equal to $1/5$ can compose a compound with τ_2 since $2/5 + 1/5 \leq 3/5$. The algorithm of RR-DUP is shown in Figure 3. RR-DUP adds task τ_i to co-scheduled set Γ_k as axis task α_k and adds the next-indexed task, τ_{i+1}, to Γ_k next. Unlike the one mentioned above, the improved RR-DUP also adds τ_{i+1} to compound $\sigma_l(\Gamma_k)$ then tests if the bottom-pointed task (τ_n at first), whose utilization is the smallest, can join $\sigma_l(\Gamma_k)$ with satisfying Equation (1). If task τ_n passes the test, it is added to $\sigma_l(\Gamma_k)$. Then the bottom-pointer moves so as to refer to the previous-indexed task, i.e. τ_{n-1}. The same test is executed against the new bottom-pointed task.

3.2 Scheduling Algorithm

As a first practical algorithm, we design UL-EDF that the scheduler in Figure 1 takes Earliest Deadline First (EDF) [4] to schedule the axis tasks. Figure 4 shows the algorithm of UL-EDF. In the line 3, it selects axis task α_i from axis set α according to EDF. Co-scheduled set Γ_i is identified from axis task α_i, that is, the co-scheduled tasks are chosen

Algorithm RR-DUP
Input: Task set τ
Output: Co-scheduled set Γ

Assume that the tasks are sorted so that $U_1 \geq U_2 \geq ... \geq U_n$ where ties are broken in favor of shorter periods.

1: *Begin*
2: set 1 to i and k; set n to j;
3: *while* $i \leq j$
4: set τ_i to α_k; add τ_i to $\sigma_1(Gamma_k)$;
5: $i = i + 1$; *if* $i > j$ *then* exit; *end if*
6: *for* $l = 2, 3, ..., M$
7: add τ_i to $\sigma_l(\Gamma_k)$;
8: $i = i + 1$; *if* $i > j$ *then* exit; *end if*
9: *while* $U_j \leq U_k - \sum_{\tau_m \in \sigma_l(\Gamma_k)} U_m$
10: add τ_j to $\sigma_l(\Gamma_k)$;
11: $j = j - 1$; *if* $i > j$ *then* exit; *end if*
12: *end while*
13: *end for*
14: $k = k + 1$
15: *end while*
16:*End*

Figure 3. The algorithm RR-DUP

automatically. Due to the introduction of the compound, we notice that there should be another scheduling policy to choose a task in the compound. The basic idea in UL-EDF is that EDF is also used for a task selection in the compound. During the line 5 to 10, it selects each task τ_j that is going to run on each non-axis thread TH_k. Then they start their execution at the same time.

Table 1. Task set τ

τ_i	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6
C_i	1	3	3	2	2	2
T_i	5	10	10	10	5	5

Figure 5, 6 and 7 show task scheduling under EDF-FF, EDF-US and UL-EDF with task set τ in Table 1. While the combinations of the co-scheduled tasks vary under EDF-FF and EDF-US, there is no variation under UL-EDF. Hence, UL-EDF can hold down the C-variability.

3.3 Multi-Case Execution Time

In the case that there are utilization differences in the co-scheduled set, the task execution time fluctuates even under UL-EDF. Here we propose the notion of *multi-case execution time* (MCET) which provides multiple execution times for each task dependently on the condition. In the case of Figure 2, as for the execution time, the fourth instance of τ_1 should be concerned separately from the other instances

Algorithm UL-EDF
Input: The set of axis tasks α
Output: $\epsilon = \{\epsilon_i | 1 \leq i \leq M\}$: a running task set

Define a as the number of the axis thread.
1: *Begin*
2: *if* no ready task in α *then* exit; *end if*
3: select α_i according to EDF;
4: set α_i to ϵ_a;
5: *for each* $\{\sigma_k(\Gamma_i) \mid k > 1\}$
6: *if* ready tasks in $\sigma_k(\Gamma_i)$ *then*
7: select τ_j from $\sigma_k(\Gamma_i)$ according to EDF;
8: set τ_j to ϵ_k;
9: *end if*
10: *end for each*
11: *End*

Figure 4. The algorithm UL-EDF

since the condition of its execution is different. The question is how we define MCET.

We provide the multiple versions of WCET for each task τ_i, i.e. $WCET_{i1}, WCET_{i2}, ..., WCET_{iK}$ where K is a constant value. To decide MCET, at first, it samples how often each WCET appears for a certain period at runtime. To be precise, the execution time of the jth instance of τ_i, i.e. ET_{ij}, is regarded as one of the WCETs that is the closest to and greater than the actual execution time (AET_{ij}).

$$ET_{ij} = min\{WCET_{ik} \mid WCET_{ik} \geq AET_{ij}\} \quad (2)$$

Secondly, it calculates MCET based on the sampled data. In general, the scheduling pattern is iterated in a *hyperperiod*. It is a natural selection that a hyperperiod H is taken for a sample period. We calculate MCET according to Equation (3).

$$MCET_i = \frac{\sum_{j=1}^{H/T_i} ET_{ij}}{H/T_i} \quad (3)$$

Note that the notion of MCET is more effective under UL-EDF because a hyperperiod might be overlong under EDF-FF and EDF-US since it is the least common multiple of all the task periods existing in the system. UL-EDF, on the other hand, affords a shorter hyperperiod since it needs no more than the least common multiple of only the tasks in each co-scheduled set. A dynamic scheduling with MCET works as follows.

1. use the largest $WCET_{ik}$ as a computation time C_i for an admission control at the beginning.
2. acquire $MCET_i$ by sampling $WCET_{ik}$ for the first hyperperiod in each co-scheduled set.
3. update the computation time C_i with the acquired $MCET_i$ at step 2.

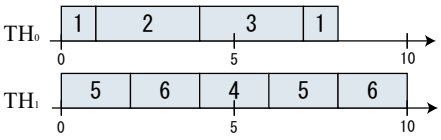

Figure 5. EDF-FF scheduling

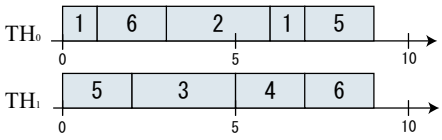

Figure 6. EDF-US scheduling

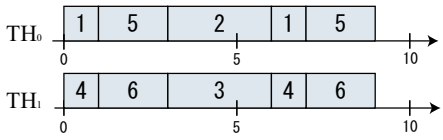

Figure 7. UL-EDF scheduling

4 Performance Evaluation

In this section, we evaluate UL-EDF and compare it with the existing algorithms EDF-FF and EDF-US. We used *Responsive Multithreaded Processor* (*RMT Processor*) in the evaluation, with letting four threads active (4-way SMT), which is an SMT-based processor for real-time processing that has been developed in our previous work [7].

For periodic tasks, we prepared three types: *PD-INT*, *PD-FP* and *MEM*. PD-INT and PD-FP are the PD control tasks which are integer-intensive and FP-intensive respectively. MEM is the task which accesses a memory intensively. The period of PD tasks is set to either $200\mu s$ or $1ms$ and the period of MEM tasks is set to either $2ms$ or $3ms$. Twenty of these tasks selected randomly are submitted to the system at the beginning. Then four of these tasks selected randomly are periodically submitted up to forty. The scheduling point is set every $10\mu s$. We measured WCET of each task type in advance. According to the measurement, WCET of PD-INT was $24.68\mu s$, PD-FP was $26.19\mu s$ and MEM was $291.33\mu s$. Based on the measurement, we provide two WCETs for PD-INT/PD-FP and three WCETs for MEM to calculate MCET. Specifically, we set $WCET_{pd1} = 20\mu s$, $WCET_{pd2} = 30\mu s$, $WCET_{mem1} = 250\mu s$, $WCET_{mem2} = 275\mu s$ and $WCET_{mem3} = 300\mu s$. A proper WCET is taken as a computation time from these values optionally in UL-EDF.

For the admission controls of the three algorithms, we take the following methods.

EDF-FF Tests if First-Fit can find any thread TH_j to pack a task into. Then tests if Equation (4) is met.

$$\sum_{\tau_i \in \theta_j} U_i \leq 1 \tag{4}$$

EDF-US Tests if Equation (5) is met.

$$\sum_{\tau_i \in \tau} U_i \leq \frac{m^2}{2m-1} \tag{5}$$

UL-EDF Tests if RR-DUP can find any thread to assign a task to. Then tests if Equation (1) and (6) are met.

$$\sum_{\tau_i \in \alpha} U_i \leq 1 \tag{6}$$

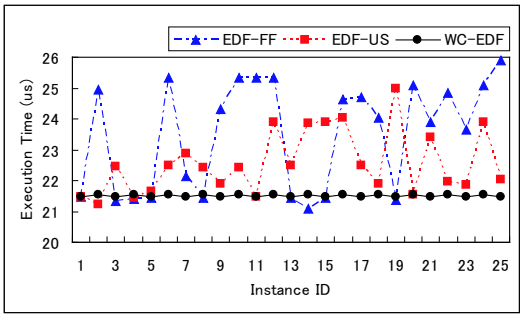

Figure 8. Fluctuation of execution time

Figure 8 depicts the execution time of a certain task for each instance. Under UL-EDF, best-case execution time (BCET) is $21.47\mu s$ and WCET is $21.56\mu s$, so the difference between them is only $0.09\mu s$. Meanwhile, under EDF-FF and EDF-US, the differences are $25.89\mu s - 21.84\mu s = 4.05\mu s$ and $24.98\mu s - 21.22\mu s = 3.76\mu s$ respectively. It is obvious that UL-EDF gives the most stable execution time. The execution time with EDF-FF fluctuates most fiercely in the three. That is because EDF-FF partitions the tasks arbitrarily by first-fit. Hence it is likely to happen that the members of the co-scheduled tasks change every time. Unlike EDF-FF, EDF-US and UL-EDF try to choose the similar tasks globally.

Secondly, Figure 9 shows the task rejection ratio where WCET of both PD-INT and PD-FP are set $30\mu s$ and WCET of MEM is set $300\mu s$. UL-EDF reduces the task rejection ratio by 9% compared to EDF-FF and by 32% compared to EDF-US at the end when forty of the tasks are submitted. The difference between UL-EDF and EDF-FF comes from the method of partitioning and the value of computation time taken in the scheduling model. RR-DUP used in UL-EDF can pack more tasks to threads effectively than first-fit used in EDF-FF. Also UL-EDF takes MCET as a computation time, that is a more proper value on an SMT processor, whereas EDF-FF takes just WCET. EDF-US, on the other hand, gives the worst performance about the task rejection ratio because its the theoretical schedulability is quite low. No deadline miss has been observed with all the algorithms in this situation.

Next, we set WCET of both PD-INT and PD-FP $20\mu s$

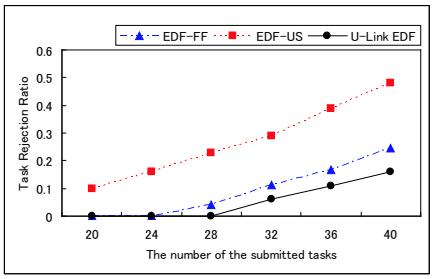

Figure 9. Rejection ratio (PD: 30μs, MEM: 300μs)

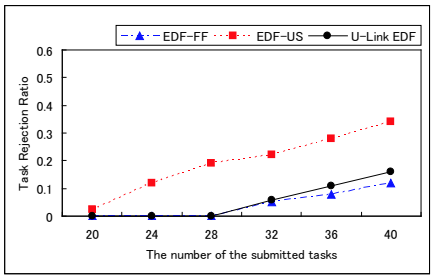

Figure 10. Rejection ratio (PD: 20μs, MEM: 250μs)

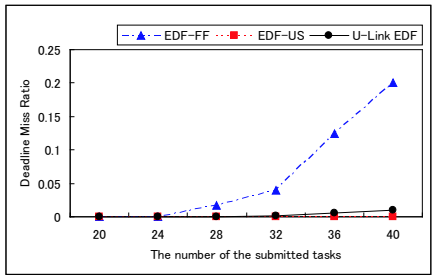

Figure 11. Deadline miss ratio (PD: 20μs, MEM: 250μs)

and MEM 250μs to enable to admit more tasks (Figure 10). While both EDF-FF and EDF-US reduce their task rejection ratio by 13% and 14% respectively, UL-EDF gives nearly the same result as the case in Figure 9. That is because UL-EDF optimizes the computation time C with MCET dynamically on the way. Thus the rejection ratio is almost steady in UL-EDF regardless of the value of WCET set in advance. In this case, EDF-FF admits tasks by 4% over UL-EDF at the end and gives the best task admission at any point. However, as shown in Figure 11, underestimation of WCET causes more deadline misses with EDF-FF.

Figure 11 depicts the deadline miss ratio with regard to the number of the submitted tasks when WCET of PD-INT, PD-FP and MEM are set 20μs, 20μs and 250μs respectively. Although EDF-FF accomplishes the best rejection ratio, the deadline miss ratio goes up to about 20% finally in compensation for it. As mentioned before, EDF-US gives such a strict admission control that it still rejects enough tasks to keep the deadline miss ratio zero. UL-EDF only allows for a little deadline misses. Although at most 1% of the tasks missed their deadline due to the underestimation of WCET during the sample period, no deadline miss has been found after the sample period.

5 Conclusion and Future Work

In this paper, we proposed U-Link scheduling including RR-DUP, UL-EDF and MCET. The evaluation showed that UL-EDF is more effective than EDF-FF and EDF-US, in the point of execution time stability, task rejection ratio and deadline miss ratio. However there are a lot of issues to be considered like a schedulability analysis, proof of a correctness, variation in tasks, and so on. Also we need to evaluate the algorithm on various and more practical environments. Although there are the issues, we believe U-Link scheduling brings a new possibility into real-time scheduling.

As the future work, we provide a theoretical analysis which proves the effectiveness of U-Link scheduling. Then we present algorithms which enables more sophisticated hard real-time processing on SMT processors.

Acknowledgement

This study was conducted through the fund of CREST, JST.

References

[1] S. J. Eggers, J. S. Emer, H. M. Levy, J. L. Lo, R. L. Stamm, and D. M. Tullsen. Simultaneous multithreading: A platform for next-generation processors. *IEEE Micro*, 17:12–19, 1997.

[2] J. Goossens, S. Funk, and S Baruah. Priority-driven scheduling of periodic task systems on multiprocessor. *Real-Time Systems*, 25:187–205, 2003.

[3] R. Jain, C. J. Hughes, and S. V. Adve. Soft real-time scheduling on simultaneous multithreaded processors. In *Proceedings of Real-Time Systems Symposium*, pages 134–145, 2002.

[4] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of ACM*, 20:46–61, 1973.

[5] J.M. Lopez, M. Garcia, J.L. Diaz, and D.F. Garcia. Utilization bounds for edf scheduling on real-time multiprocessor systems. *Real-Time Systems*, 28:39–68, 2004.

[6] D. M. Tullsen, S. J. Eggers, and H. M. Levy. Simultaneous multithreading: Maximizing on-chip parallelism. In *Proceedings of Annual International Symposium on Computer Architecture*, pages 392–403, 1995.

[7] N. Yamasaki. Responsive multithreaded processor for distributed real-time systems. *Journal of Robotics and Mechatronics*, 17:130–141, 2005.