Gang EDF Scheduling of Parallel Task Systems *

Shinpei Kato and Yutaka Ishikawa
Graduate School of Information Science and Technology
The University of Tokyo, Japan

Abstract

The preemptive real-time scheduling of sporadic parallel task systems is studied. We present an algorithm, called Gang EDF, which applies the Earliest Deadline First (EDF) policy to the traditional Gang scheduling scheme. We also provide schedulability analysis of Gang EDF. Specifically, the total amount of interference that is necessary to cause a deadline miss is first identified. The contribution of each task to the interference is then bounded. Finally, verifying that the total amount of contribution does not exceed the necessary interference for every task, the schedulability test is derived. Although the techniques proposed herein are based on the prior results for the sequential task model, we introduce new ideas for the parallel task model.

1 Introduction

Ever since the emergence of multiprocessor systems, the idea of parallel processing – simultaneous use of multiple processors for an individual task – has appeared in various environments, particularly in the domain of high-performance computing. Given that much attention has been focused on the research and development of many-core architectures [38, 35], which integrate many low-power cores on a chip, parallel processing is now relevant to even embedded real-time systems [33, 36, 29].

It has been shown that the performance of parallel processing highly depends on the scheduling of parallel tasks. We expect individual tasks to achieve excellent performance by parallelization. The primary issue is how to allocate CPU resources among competing tasks, in such a way that satisfies the demand of tasks and produces good overall performance. These objectives have raised a number of scheduling issues with respect to various parallel task systems.

In parallel applications, there is evidence that threads of the same task should or even must be executed in parallel. Let us consider two examples related to shared-memory multiprocessor systems, which are introduced in [16]. Imagine a case in which many threads access a critical section guarded by a lock. The lock must be acquired by threads before using the critical section. A thread which captured the lock may lose its processor due to preemptive scheduling policies. Then, other threads must wait for the release of the lock until the next time quantum in which the thread holding the lock is running. Thus, a thread which is holding the lock should not be preempted when other threads of the same task are running. Consider the second example: two threads of the same task, a sender and a receiver, communicate with each other but are run in different times. A message from the sender must wait until the receiver obtains the CPU resource. A response from the receiver must also wait until the sender is resumed. The above two examples imply that threads of the same task should be run in parallel.

In order to achieve efficient parallel processing, *Gang scheduling* and *Coscheduling* [34, 22, 19] have been invented. They grant the processors to the threads of the same task at the same time quantum. Gang scheduling is stricter than Coscheduling in that it requires all threads of the same task to run concurrently, while Coscheduling allows for fragments, which are sets of threads that do not run concurrently with the rest of the gang.

This research is aimed at exploring the real-time scheduling of parallel task systems. In particular, we study such an algorithm that applies the Earliest Deadline First (EDF) [31] policy to Gang scheduling. To the best of our knowledge, no research has ever focused on this subject, while different scheduling algorithms and assumptions are found in [32, 28, 30, 12]. In addition, this is the first challenge that obtains the schedulability conditions of EDF for the parallel task model, though some heuristics and simulation-based tests have been considered in [32, 28].

The rest of this paper is organized as follows. In the next section, the task model is defined. The Gang EDF scheduling is presented in Section 3, and its schedulability analysis is given in Section 4. The related work is presented in Section 5. This paper is concluded in Section 6.

*This work is supported by the fund of Research Fellowships of the Japan Society for the Promotion of Science for Young Scientists. This work is also supported in part by the fund of Core Research for Evolutional Science and Technology, Japan Science and Technology Agency.

2 Model

We consider parallel task systems on shared-memory muti-core/many-core platforms with m processing units. In general, a parallel task is said to be: (i) *rigid* if the number of processors simultaneously used by the task is fixed *a priori*, (ii) *moldable* if the number of processors simultaneously used by the task is not fixed but determined before the execution, and (iii) *malleable* if the number of processors simultaneously used by the task can change at runtime. We focus on the moldable task model in this paper.

According to [16, 17], most parallel applications in the real world are moldable, and at the submission time users or schedulers select the number of processors upon which a parallel task will run. We particularly assume that a parallel task generates the same number of threads as used processors before the execution, and each of which is handled as a generic individual task, like most MPI [40] and OpenMP [8] implementations do.

Let $\tau = \{\tau_1, \tau_2, ..., \tau_n\}$ denote a set of n sporadic moldable parallel tasks. Each task $\tau_i = (v_i, C_i, D_i, T_i)$ is characterized by the number v_i of used processors, a worst-case execution time C_i when executed in parallel on v_i processors, a minimum inter-arrival time T_i (also called period), and a relative deadline D_i. The utilization of τ_i is defined by $U_i = C_i/T_i$. In this paper, we restrict our attention to a constrained-deadline case: i.e., $D_i \leq T_i$ holds for any τ_i.

Each task τ_i generates an infinite sequence of jobs, with arrival times of successive jobs separated by at least T_i time units. Each job of τ_i, executed in parallel by v_i threads (on v_i processors), has a worst-case execution time equal to C_i and a deadline at D_i time units after its arrival time. In fact, perfect parallelism may not be possible, and actual execution times may be different among the v_i threads within a job. We therefore consider C_i to be the worst-case time amount that at least one thread within the job is executing for. We also assume that all threads within the job consume C_i time units, including idle and waiting times to synchronize with each other for parallel execution. As a result, the execution of a job of τ_i is represented as a $C_i \times v_i$ rectangle in time $\times$ processor space.

Jobs are fully independent and preemptive: any job being executed can be suspended (preempted) at any time instant, and it is later resumed with no cost or penalty. Resource sharing is restricted to threads within a task.

While prior work [15, 32, 12] paid attention on the speedup ratio of C_i with respect to v_i, to obtain better runtime scheduling results, the speedup model is not strictly defined in this paper. The primary goal of this paper is rather to provide the guarantees for all tasks to be schedulable when C_i and v_i are given. Specifically, we assume that C_i is determined by v_i, and v_i is determined by users or schedulers before the execution.

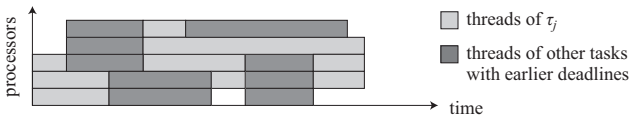

Figure 1. Coscheduling case.

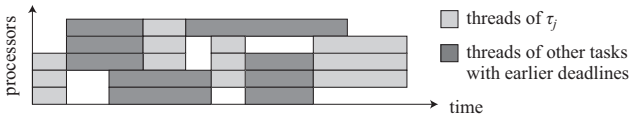

Figure 2. Gang scheduling case.

The *demand bound function* $\mathrm{dbf}(\tau_i, L)$, originally introduced by Baruah *et al.* [6], is extended to compute the upper bound of the *processor demand* of τ_i, parallelized on v_i processors, over any time interval of length L.

$$\mathrm{dbf}(\tau_i, L) = \max\left(0, \left\lfloor \frac{L - D_i}{T_i} \right\rfloor + 1\right) \times C_i \times v_i \tag{1}$$

We also define the *horizontal-demand bound function*, $\mathrm{hbf}(\tau_i, L)$, that computes the upper bound of the *time length demand* of τ_i over any time interval of length L, which results in the traditional form of the demand bound function [6] designed for the sequential task model.

$$\mathrm{hbf}(\tau_i, L) = \mathrm{dbf}(\tau_i, L) \times \frac{1}{v_i} \tag{2}$$

3 Gang EDF Scheduling

We present an algorithm, called *Gang EDF*, which applies the EDF policy to the Gang scheduling scheme. Gang EDF is basically the same as the classical Global EDF: *jobs with earlier deadlines are assigned higher priorities*. However, we need to take into account spacial limitation on the number of available processors in Gang EDF, while we can always choose m ready tasks in Global EDF.

Let Q_k be a set of k tasks that have ready jobs with the earliest (and earlier) deadlines. For the k tasks included in such Q_k that satisfies $\sum_{\tau_i \in Q_k} v_i \leq m$ and $\sum_{\tau_i \in Q_{k+1}} v_i > m$, they can be uneventfully dispatched for execution. However, we need to take care of such a task τ_j that is not included in Q_k but in Q_{k+1}, i.e., the one has a ready job with the $k+1$th earliest deadline, since there are only $m - \sum_{\tau_i \in Q_k} v_i$ ($< v_j$) processors available for τ_j.

For scheduling such τ_j, we have two scheduling policies. Coscheduling just partially executes $m - \sum_{\tau_i \in Q_k} v_i$ threads of τ_j, and its remaining threads are executed when some processors become available for τ_j, as shown in Figure 1. Under this scheduling strategy, tasks with later deadlines are never scheduled ahead of τ_j, as the classical Global EDF.

Let Q_{ready} be a set of ready tasks that are sorted in order of early deadlines.
Let Q_{run} be a set of running tasks being scheduled.
Let *first_task*(Q_{ready}) be a function that returns the first task in Q_{ready}.

1. $Q_{\text{run}} = \emptyset$;
2. **while** $Q_{\text{ready}} \neq \emptyset \wedge \sum_{\tau_i \in Q_{\text{run}}} v_i \neq m$ **do**
3. $\quad \tau_j = \textit{first\_task}(Q_{\text{ready}})$;
4. $\quad$ **if** $v_j + \sum_{\tau_i \in Q_{\text{run}}} v_i \leq m$ **then**
5. $\quad\quad Q_{\text{run}} = Q_{\text{run}} \cup \{\tau_j\}$;
6. $\quad$ **end if**
7. $\quad Q_{\text{ready}} = Q_{\text{ready}} \setminus \{\tau_j\}$;
8. **end while**
9. execute all the tasks in Q_{run};

Figure 3. Gang EDF scheduling policy.

In fact, Coscheduling is appropriate for the malleable task model rather than the moldable one. Gang scheduling, on the other hand, temporarily blocks all the threads of τ_j until more than or equal to v_j processors become available for τ_j, as shown in Figure 2, and tasks with later deadlines may be scheduled ahead of τ_j.

Coscheduling may complete the job of τ_j earlier than Gang scheduling. However, it may increase the total amount of time that the job executes for, because some threads may cause busy wait to synchronize with other suspended ones in Coscheduling. If such a synchronization occurs at the end of job execution, even the completion time may not be very different between the two policies. Hence, we adopt Gang scheduling in this paper.

Gang EDF selects a job with the earliest deadline, and if the job cannot start due to spacial limitation on the number of available processors, then it selects the next one according to a first fit heuristic. The pseudo-code of Gang EDF is described in Figure 3. In order to realize Gang EDF scheduling, we need additional implementations to preempt all the threads within a parallel job, when at least one thread needs to be preempted, for most commodity operating systems, like Linux. However, we claim that the implementation is still realistic, since most those operating systems support software interruption functions that enable one scheduler to invoke others executing on different processors.

4 Schedulability Analysis

In this section, we derive a schedulability test of Gang EDF for sporadic moldable parallel task systems with constrained deadlines. Our approach is based on the techniques proposed by Baruah *et al.* [4]: we obtain necessary conditions for any constrained sporadic task system τ scheduled by Gang EDF to miss a deadline on m processors – henceforth referred to as the [BAR] test. Specifically, the first part of the schedulability test is a direct modification of the [BAR] test, however we introduce new ideas for Gang EDF in the second part.

In the schedulability test, we assume that the number of used processors for parallel execution of each task τ_i has been already given (by users or schedulers). Then, the test verifies the given system is schedulable or not. The test may also help designers to decide the number of used processors for each task.

Following the [BAR] test, we first consider any legal sequence of job requests of task system τ, on which a deadline is missed by Gang EDF. Suppose that a job of task τ_k – henceforth referred to as the *problem job* – is the one to first miss a deadline at time t_d, i.e., t_d is equal to the deadline of this job. Let t_a be the arrival time of this job: $t_a = t_d - D_k$, and t_o be the latest time instant earlier than or equal to t_a, at which at least v_k processors are idled. We then denote by $\Delta_k = t_d - t_o$ the length of interval $[t_o, t_d)$. There is no need to take care of all jobs with deadlines later than t_d, since such jobs have no effect on those with earlier deadlines under preemptive EDF scheduling.

As approaches in [2, 7, 3, 4, 5], our goal is to identify necessary conditions for a deadline miss to occur. In order for the problem job to miss a deadline, it is necessary that the job is blocked for strictly more than $D_k - C_k$ time units over $[t_a, t_d)$. Since τ_k requires v_k processors at the same time for parallel execution, it is blocked while $m - v_k + 1$ or more processors are busy. Given the definition of t_o, if the deadline of the problem job is missed, the total length of intervals over $[t_o, t_d)$, during which at least $m - v_k + 1$ processors are executing jobs other than the problem job, must be greater than $\Delta_k - C_k$. Notice that this minimum interference necessary for the deadline miss can be depicted by a rectangle in time $\times$ processor space, whose width w_k and height h_k are given by Equation (3) and (4) respectively.

$$w_k = \Delta_k - C_k \tag{3}$$

$$h_k = m - v_k + 1 \tag{4}$$

Henceforth, this rectangle is called *interference rectangle* of τ_k. Now, we know that if the total amount of work done by jobs other than the problem job over $[t_o, t_d)$ is greater than the size of the interference rectangle of τ_k, the deadline of the problem job is missed.

Figure 4 shows an example of the schedule produced by Gang EDF, on which the problem job missed a deadline. The light-gray box indicates the execution of the problem job, and the dark-gray boxes indicate the executions of jobs of other jobs, which have earlier deadlines than the problem job. We can observe that the problem job missed a deadline, because the total length of intervals over $[t_o, t_d)$, during which at least $h_k = m - v_k + 1$ processors are busy as depicted by the shadow area, is greater than $\Delta_k - C_k$.

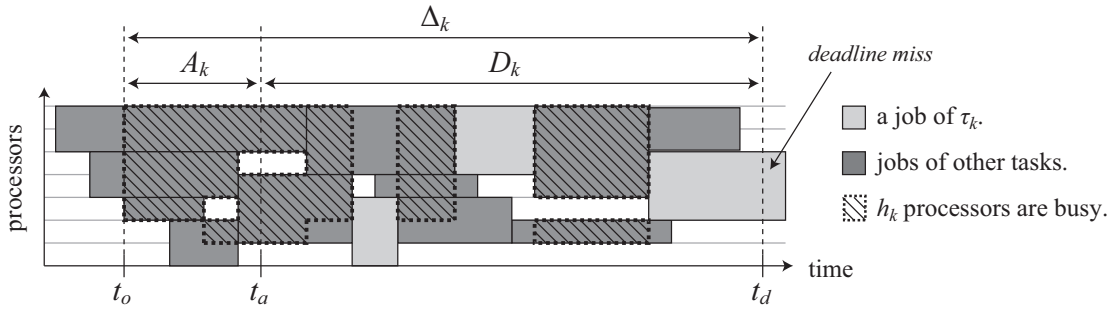

Figure 4. Example of deadline miss in Gang EDF scheduling.

We realize that some processors may be left idle in Gang EDF scheduling, even though there are ready jobs. However, we can still say that Gang EDF is *work-conserving*, since none of the ready jobs can fit the idle processors, which means that none of them is eligible.

Let $I_k(\tau_i)$ denote the contribution of τ_i to the work that interferes the problem job over $[t_o, t_d)$, meaning that it blocks the problem job over $[t_a, t_d)$ and executes over $[t_o, t_a)$. In order for the deadline miss of the problem job to occur, it is necessary that the total amount of work that interferes over $[t_o, t_d)$ satisfies:

$$\sum_{\tau_i \in \tau} I_k(\tau_i) > w_k \times h_k \tag{5}$$

Notice that h_k is fixed while w_k is not, since Δ_k is not determined, so we have not known yet the time at which at least $m - v_k$ processors are idled. To show that a given system is schedulable by Gang EDF, we must show that Condition (5) cannot be satisfied for all tasks τ_k and for all values of Δ_k, but the range of the check points for Δ_k can be reduced to a finite number, as demonstrated later in this section.

According to the [BAR] test, we have *carry-in jobs* in the interference rectangle who arrive before t_o and have not completed execution by t_o. In the following, the upper bound on the contribution of τ_i to the interference rectangle is denoted by $I_1(\tau_i)$ if τ_i does not have a carry-in job, and is denoted by $I_2(\tau_i)$ if it does.

4.1 Simple Bounds

We first consider a simple analysis that is directly modified from [4]. If a task τ_i does not have a carry-in job, its contribution to the interference rectangle of τ_k is generated by jobs that arrive in, and have deadlines within, the interval $[t_o, t_d)$. So for the case $i \neq k$, the horizontal contribution of τ_i is at most $\text{hbf}(\tau_i, \Delta_k)$. It cannot also exceed the width of the interference rectangle of τ_k. Hence, the horizontal contribution of τ_i to the interference rectangle of τ_k is at most

$$\min\{\text{hbf}(\tau_i, \Delta_k), w_k\}.$$

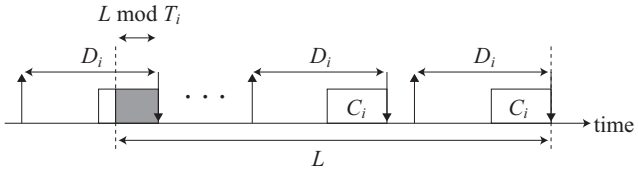

Figure 5. Carry-in job execution.

Meanwhile, its vertical contribution is constant and is at most v_i. It cannot also exceed the height of the interference rectangle of τ_k. Hence, the vertical contribution of τ_i to the interference rectangle of τ_k is at most

$$\min(v_i, h_k).$$

Now, consider the case $i = k$. From $D_i \leq T_i$ we know that the job of τ_k arriving at t_a cannot contribute to the interference rectangle. Its contribution cannot also exceed the length of the interval $[t_o, t_a)$, which is represented as $A_k = \Delta_k - D_k$ in Figure 4.

Putting these pieces together, the upper bound on the contribution of τ_i to the interference rectangle is obtained by Equation (6).

$$I_1(\tau_i) = \begin{cases} \min\{\text{hbf}(\tau_i, \Delta_k), w_k\} \times \min(v_i, h_k) & \text{if } i \neq k \\ \min\{\text{hbf}(\tau_i, \Delta_k) - C_k, A_k\} \times \min(v_i, h_k) & \text{otherwise} \end{cases} \tag{6}$$

Let us now consider the case in which a task τ_i offers a carry-in job. It was shown in [7] that the total amount of time that τ_i can consume in this case can be bounded by considering the scenario in which some job of τ_i has a deadline at t_d, and all jobs of τ_i are executed without preemptions and complete at their deadlines, as shown in Figure 5.

Let $\text{hbf}'(\tau_i, L)$ be the horizontal demand of τ_i over a contiguous interval of length L, if some job of τ_i has its deadline at the very end of the interval and all jobs of τ_i complete at their deadlines without preemptions. It is easily seen from

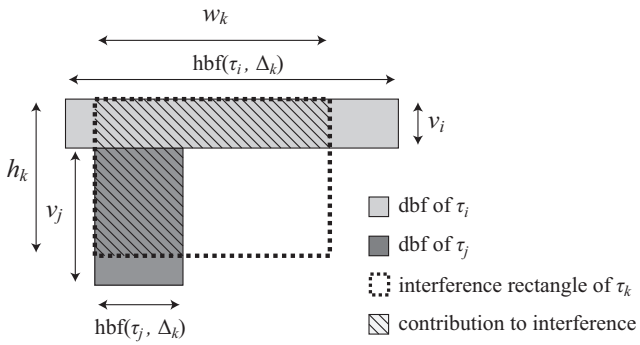

Figure 6. Contributions of two tasks to interference rectangle when $\text{hbf}(\tau_j, \Delta_k) \leq w_k \leq \text{hbf}(\tau_i, \Delta_k)$ **and** $h_k - v_i \leq v_j \leq h_k$ **hold.**

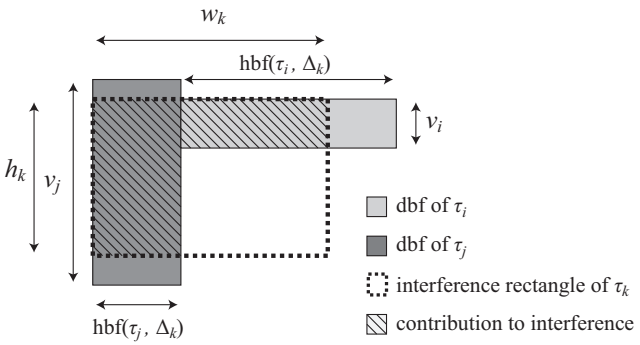

Figure 7. Contributions of two tasks to interference rectangle when $v_i \leq h_k \leq v_j$ **and** $w_k - \text{hbf}(\tau_j, \Delta_k) \leq \text{hbf}(\tau_i, \Delta_k) \leq w_k$ **hold.**

Figure 5 that the number of jobs of τ_i that arrive in, and have deadlines within, the interval is $\lfloor L/T_i \rfloor$. It is also easily seen that the carry-in amount is at most $\min(C_i, L \bmod T_i)$. Thus, $\text{hbf}'(\tau_i, L)$ is given by:

$$\text{hbf}'(\tau_i, L) = \left\lfloor \frac{L}{T_i} \right\rfloor \times C_i + \min(C_i, L \bmod T_i) \tag{7}$$

Using $\text{hbf}'(\tau_i, L)$ given above, the contribution of τ_i with a carry-in job to the interference rectangle can be bounded in the same way as the one without a carry-in job, and is derived as follows.

$$I_2(\tau_i) = \begin{cases} \min\{\text{hbf}'(\tau_i, \Delta_k), w_k\} \times \min(v_i, h_k) & \text{if } i \neq k \\ \min\{\text{hbf}'(\tau_i, \Delta_k) - C_k, A_k\} \times \min(v_i, h_k) & \text{otherwise} \end{cases} \tag{8}$$

4.2 Improved Bounds

The bounds of the contribution of each task to the interference rectangle of τ_k, derived in Section 4.1, are obtained simply based on the [BAR] test that has been designed for Global EDF. In this section, we improve the tightness of the bounds in consideration of the Gang scheduling behavior, to obtain better schedulability results.

We notice that Equation (6) is pessimistic in that it overestimates the amount of the interference from each task τ_i to τ_k. In order to reduce the parts of $I_1(\tau_i)$ that are obviously overestimated, we give the following lemmas.

Lemma 1. *For any two tasks τ_i and τ_j ($i \neq j \neq k$), if $\text{hbf}(\tau_j, \Delta_k) \leq w_k \leq \text{hbf}(\tau_i, \Delta_k)$ and $h_k - v_i \leq v_j \leq h_k$ are satisfied, where w_k and h_k are the width and height of the interference rectangle of τ_k, the sum of $I_1(\tau_i)$ and $I_1(\tau_j)$ is at most:*

$$w_k \times v_i + \text{hbf}(\tau_j, \Delta_k) \times (h_k - v_i) \tag{9}$$

Proof. In the $w_k \times h_k$ interference rectangle, for any task the horizontal contribution is no greater than w_k and the vertical one is no greater than h_k.

We first consider the case in which τ_i contributes to the interference rectangle by $w_k \times v_i$. It is obvious that the contribution of τ_j to the interference rectangle is at most $\text{hbf}(\tau_j, \Delta_k) \times (h_k - v_i)$, as shown in Figure 6. Note that the interference rectangle may not be contiguous, and therefore may not be a single rectangle, but the total size is equal to $w_k \times h_k$ in any case.

We next consider the case in which τ_j contributes to the interference rectangle by $\text{hbf}(\tau_j, \Delta_k) \times v_j$. In this case, τ_j is moved upwards by the length $v_j + v_i - h_k$, and therefore the contribution of τ_j is increased by

$$I_{\text{inc}} = \text{hbf}(\tau_j, \Delta_k) \times (v_j + v_i - h_k),$$

as compared to the situation depicted in Figure 6. Since τ_j moves upwards by the length $v_j + v_i - h_k$, τ_i must also move upwards by the length $v_j + v_i - h_k$ or rightwards by the length $\text{hbf}(\tau_j, \Delta_k)$ in the interference rectangle.

- If τ_i moves upwards, the contribution of τ_i is decreased by $I_{\text{dec}} = w_k \times (v_j + v_i - h_k)$. Since we know $\text{hbf}(\tau_j, \Delta_k) \leq w_k$, we have $I_{\text{dec}} \geq \text{hbf}(\tau_j, \Delta_k) \times (v_j + v_i - h_k) = I_{\text{inc}}$.
- If τ_i moves rightwards, the contribution of τ_i is decreased by $I_{\text{dec}} = \text{hbf}(\tau_j, \Delta_k) \times v_i$. Since we know $v_j \leq h_k$, we have $I_{\text{dec}} \geq \text{hbf}(\tau_j, \Delta_k) \times v_i + v_j - h_k = I_{\text{inc}}$.

There is no need to consider the case in which τ_i moves both rightwards and upwards, since it must move either upwards by $v_j + v_i - h_k$ or rightwards by $\text{hbf}(\tau_i, \Delta_k)$ after all, to let τ_j move upwards. As a result, moving τ_j upwards never increases the interference. Hence, the lemma is true. □

Lemma 2. *For any two tasks τ_i and τ_j ($i \neq j \neq k$), if $v_i \leq h_k \leq v_j$ and $w_k - \text{hbf}(\tau_j, \Delta_k) \leq \text{hbf}(\tau_i, \Delta_k) \leq w_k$ are*

satisfied, where w_k and h_k are the width and height of the interference rectangle of τ_k, the sum of $I_1(\tau_i)$ and $I_1(\tau_j)$ is at most:

$$\{w_k - \mathrm{hbf}(\tau_j, \Delta_k)\} \times v_i + \mathrm{hbf}(\tau_j, \Delta_k) \times h_k \tag{10}$$

Proof. The corresponding situation is shown in Figure 7. Reversing the width and the height of Figure 7, we obtain the same situation in Figure 6, in which τ_i and τ_j are swapped. Thus, the lemma is true by the same reason as Lemma 1. □

Note that Equation (9) and (10) are equal. From Lemma 1 and 2, it is clear that Equation (6) is pessimistic when any two tasks in Lemma 1 and 2 are given, because it assumes that the vertical contribution of τ_j to the interference rectangle is always at most v_j, while Lemma 1 implies that it is at most $h_k - v_i$. By the same token, Equation (6) assumes that the horizontal contribution of τ_j to the interference rectangle is always at most $\mathrm{hbf}(\tau_j, \Delta_k)$, while Lemma 2 implies that it is at most $w_k - \mathrm{hbf}(\tau_j, \Delta_k)$.

Lemma 1 leads to that for a task τ_j whose vertical demand is $h_k - v_i \le v_j \le h_k$, the size of the interference rectangle looks like $w_k \times (h_k - v_i)$. Lemma 2 also leads to that for a task τ_j whose horizontal demand is $w_k - \mathrm{hbf}(\tau_i, \Delta_k) \le \mathrm{hbf}(\tau_j, \Delta_k) \le w_k$, the size of the interference rectangle looks like $\{w_k - \mathrm{hbf}(\tau_\mathrm{i}, \Delta_\mathrm{k})\} \times h_k$. The above discussion gives the following lemmas.

Lemma 3. *The $w_k \times h_k$ interference rectangle of τ_k can be minified to the $w_k \times (h_k - v_i)$ one, if there exists a task τ_i with $hbf(\tau, \Delta_k) \ge w_k$.*

Lemma 4. *The $w_k \times h_k$ interference rectangle of τ_k can be minified to the $\{w_k - \mathrm{hbf}(\tau_\mathrm{i}, \Delta_\mathrm{k})\} \times h_k$ one, if there exists a task τ_i with $v_i \ge h_k$.*

Using Lemma 3 and 4, the interference $I_1(\tau_i)$ of each task τ_i to τ_k is renewed as follows.

The interference rectangle of τ_k is initially given as $w_k = \Delta_k - C_k$ and $h_k = m - v_k + 1$. Let us denote $\alpha_1 = \{\tau_{i \ne k} \in \tau \mid \mathrm{hbf}(\tau_i, \Delta_k) \ge w_k\}$: a set of tasks whose horizontal demands are greater than or equal to the width of the interference rectangle. Like Equation (6), we first let the contribution of every such task $\tau_i \in \alpha_1$ be

$$I_1(\tau_i \in \alpha_1) = w_k \times \min\{v_i, h_k\}.$$

Now, we know from Lemma 3 that the height of the interference rectangle can be reduced to

$$h_k(\alpha_1) = h_k - \sum_{\tau_i \in \alpha_1} v_i.$$

Note that if $\sum_{\tau_i \in \alpha_1} v_i \ge h_k$ holds, the interference rectangle is filled with a set of tasks $\{\tau_i \in \alpha_1\}$, meaning that τ_k is not schedulable.

Let us next denote $\beta_1 = \{\tau_i \notin \alpha_1 \mid v_i \ge h_k(\alpha_1)\}$: a set of tasks whose vertical demands are greater than or equal to the height of the renewed interference rectangle. The contribution of every such task $\tau_{i \ne k} \in \beta_1$ is then at most

$$I_1(\tau_i \in \beta_1) = \mathrm{hbf}(\tau_i, \Delta_k) \times h_k(\alpha_1).$$

Now, the width of the interference rectangle can be reduced to

$$w_k(\beta_1) = w_k - \sum_{\tau_i \in \beta_1} \mathrm{hbf}(\tau_i, \Delta_k).$$

By the same token, we can repeat reducing the size of the interference rectangle by finding a set of tasks that (i) have horizontal demands greater than or equal to the width of the interference rectangle and (ii) have vertical demands greater than or equal to the height of the interference rectangle, one after the other. Let us now give the following notations, where $h_k(\alpha_0) = h_k$ and $w_k(\beta_0) = w_k$.

$$\alpha_s = \left\{ \tau_{i \ne k} \notin \bigcup_{r<s} \alpha_r \cup \bigcup_{r<s} \beta_r \;\middle|\; \mathrm{hbf}(\tau_i, \Delta_k) \ge w_k(\beta_{s-1}) \right\} \tag{11}$$

$$\beta_s = \left\{ \tau_{i \ne k} \notin \bigcup_{r \le s} \alpha_r \cup \bigcup_{r<s} \beta_r \;\middle|\; v_i \ge h_k(\alpha_s) \right\} \tag{12}$$

$$h_k(\alpha_s) = h_k - \sum_{\tau_i \in \sum_{r \le s} \alpha_r} v_i \tag{13}$$

$$w_k(\beta_s) = w_k - \sum_{\tau_i \in \sum_{r \le s} \beta_r} \mathrm{hbf}(\tau_i, \Delta_k) \tag{14}$$

The procedure of reducing the size of the interference rectangle is then formalized as follows.

1. Initialize as $s = 1$.
2. Find α_s. If $\alpha_s = \emptyset$, go to Step 5. Otherwise define $\alpha_{\max} = \alpha_s$.
3. Find β_s. If $\beta_s = \emptyset$, go to Step 5. Otherwise define $\beta_{\max} = \beta_s$.
4. Increment as $s = s + 1$. Go back to Step 2.
5. Exit the procedure.

Let γ be a set of the remaining tasks that are not included in any α_s and any β_s:

$$\gamma = \{\tau_i \notin \alpha_1 \cup \ldots \cup \alpha_{\max} \cup \beta_1 \cup \ldots \cup \beta_{\max}\} \tag{15}$$

The size of the interference rectangle is now reduced to $w_k(\beta_{\max}) \times h_k(\alpha_{\max})$. By definition of γ, we know that $\forall \tau_i \in \gamma$ satisfies $\mathrm{hbf}(\tau_i, \Delta_k) < w_k(\beta_{\max})$ and $v_i < h_k(\alpha_{\max})$. In addition, by definitions of α_s and β_s, we know that $\forall \tau_i \in \alpha_s$

satisfies $\mathrm{hbf}(\tau_i, \Delta_k) \geq w_k(\beta_{s-1})$, and $\forall \tau_i \in \beta_s$ satisfies $v_i \geq h_k(\alpha_s)$. Consequently, the upper bound on the contribution of τ_i with a carry-in job to the interference rectangle of τ_k can be renewed by Equation (16).

$$I_1(\tau_i) = \begin{cases} w_k(\beta_{s-1}) \times v_i & \text{if } \tau_i \in \alpha_s \\ \mathrm{hbf}(\tau_i, \Delta_k) \times h_k(\alpha_s) & \text{if } \tau_i \in \beta_s \\ \mathrm{hbf}(\tau_i, \Delta_k) \times v_i & \text{if } \tau_i \in \gamma \wedge i \neq k \\ \min\{\mathrm{hbf}(\tau_i, \Delta_k) - C_k, A_k\} \times v_i & \text{otherwise} \end{cases} \tag{16}$$

We can remove the overestimated parts of Equation (8) in the same way. As seen in Section 4.1, all we have to do is to replace $\mathrm{hbf}(\tau_i, \Delta_k)$ with $\mathrm{hbf}'(\tau_i, \Delta_k)$ that is given by Equation (7). So let us give the following notions, where $h_k(\alpha'_0) = h_k$ and $w_k(\beta'_0) = w_k$.

$$\alpha'_s = \left\{ \tau_{i \neq k} \notin \bigcup_{r<s} \alpha'_r \cup \bigcup_{r<s} \beta'_r \;\middle|\; \mathrm{hbf}'(\tau_i, \Delta_k) \geq w_k(\beta'_{s-1}) \right\} \tag{17}$$

$$\beta'_s = \left\{ \tau_{i \neq k} \notin \bigcup_{r \leq s} \alpha'_r \cup \bigcup_{r<s} \beta'_r \;\middle|\; v_i \geq h_k(\alpha'_s) \right\} \tag{18}$$

$$h_k(\alpha'_s) = h_k - \sum_{\tau_i \in \sum_{r \leq s} \alpha'_r} v_i \tag{19}$$

$$w_k(\beta'_s) = w_k - \sum_{\tau_i \in \sum_{r \leq s} \beta'_r} \mathrm{hbf}'(\tau_i, \Delta_k) \tag{20}$$

Let us also define γ' as a set of the remaining tasks that are not included into any α'_s and any β'_s:

$$\gamma' = \{\tau_i \notin \alpha'_1 \cup \ldots \cup \alpha'_{\max} \cup \beta'_1 \cup \ldots \cup \beta'_{\max}\} \tag{21}$$

Consequently, the upper bound on the contribution of τ_i with no carry-in job to the interference rectangle of τ_k can be renewed by Equation (22).

$$I_2(\tau_i) = \begin{cases} w_k(\beta'_{s-1}) \times v_i & \text{if } \tau_i \in \alpha'_s \\ \mathrm{hbf}'(\tau_i, \Delta_k) \times h_k(\alpha'_s) & \text{if } \tau_i \in \beta'_s \\ \mathrm{hbf}'(\tau_i, \Delta_k) \times v_i & \text{if } \tau_i \in \gamma' \wedge i \neq k \\ \min\{\mathrm{hbf}'(\tau_i, \Delta_k) - C_k, A_k\} \times v_i & \text{otherwise} \end{cases} \tag{22}$$

4.3 Schedulability Condition

We now put the pieces of $I_1(\tau_i)$ and $I_2(\tau_i)$ together. Note that the following discussion is true for both the simple bounds derived in Section (4.1) and the improved bounds derived in Section (4.2).

Let I_{diff} be the difference between $I_2(\tau_i)$ and $I_1(\tau_i)$:

$$I_{\mathrm{diff}}(\tau_i) = I_2(\tau_i) - I_1(\tau_i) \tag{23}$$

It is clear that $I_{\mathrm{diff}}(\tau_i)$ represents the contribution of τ_i by its carry-in job to the interference rectangle. By definition of t_o, the number of used processors just before time t_o is at most $m - v_k$. Thus, the total amount of work contributed by the carry-in parts is at most $I_{\mathrm{carry-in}}$ defined by Equation (24) below, where σ is any set of tasks, which satisfies $\sum_{\tau_i \in \sigma} v_i \leq m - v_k$.

$$I_{\mathrm{carry-in}} = \begin{cases} \max\left\{\sum_{\tau_i \in \sigma} I_{\mathrm{diff}}(\tau_i)\right\} & \text{if } \exists \sigma \subset \tau \\ \max\{I_{\mathrm{diff}}(\tau_i)\} & \text{otherwise} \end{cases} \tag{24}$$

In order to obtain $I_{\mathrm{carry-in}}$, we first need to determine σ if it exists, however this problem is reduced to the knapsack problem that is known to be NP-hard to solve, given that we can consider $m - v_k$ as the size of a knapsack, each v_i as the size of an item, and each $I_{\mathrm{diff}}(\tau_i)$ as the value of an item.

To simplify the problem, we allow $I_{\mathrm{carry-in}}$ to be slightly overestimated. Let τ' be a copy of τ, in which the tasks are sorted such that $I_{\mathrm{diff}}(\tau_i)/v_i \geq I_{\mathrm{diff}}(\tau_{i+1})/v_{i+1}$ for any i, where ties are broken in favor of greater v_i. We then just choose a set $\tau_{\mathrm{carry-in}}$ of the first tasks in τ' satisfying

$$\sum_{\tau_i \in \tau_{\mathrm{carry-in}}} v_i \geq m - v_k.$$

On obtaining $I_{\mathrm{carry-in}}$ from $\tau_{\mathrm{carry-in}}$, we need not to take into account a part of the vertical demand, which protrudes from the height $m - v_k$ of the interference rectangle. In other words, we can consider that the number of processors used by the last task τ_l in $\tau_{\mathrm{carry-in}}$ is n'_l, which is given as

$$n'_l = m - v_k - \sum_{\tau_i \in \tau_{\mathrm{carry-in}} \setminus \tau_l} v_i.$$

Note that $n'_l = v_l$ holds if $\sum_{\tau_i \in \tau_{\mathrm{carry-in}}} v_i = m - v_k$ holds. The total amount of work that is contributed by the carry-in jobs is now approximated by Equation (25) below, instead of Equation (24).

$$I_{\mathrm{carry-in}} = \sum_{\tau_i \in \tau_{\mathrm{carry-in}}} I_{\mathrm{diff}}(\tau_i) - (v_l - n'_l) \times I_{\mathrm{diff}}(\tau_l) \tag{25}$$

We know that the tasks $\{\tau_i \in \tau_{\mathrm{carry-in}}\}$ contribute to each $I_2(\tau_i)$, meaning that the remaining tasks $\{\tau_i \in \tau \setminus \tau_{\mathrm{carry-in}}\}$ must contribute to each $I_1(\tau_i)$. Hence, Equation (5) may be rewritten as Equation (26) below, where $I_{\mathrm{carry-in}}$ is given by Equation (25).

$$\sum_{\tau_i \in \tau} I_1(\tau_i) + I_{\mathrm{carry-in}} > w_k \times h_k \tag{26}$$

Theorem 1. *It is guaranteed that a task system τ is successfully scheduled by Gang EDF upon m processors, if the following condition is satisfied for all tasks $\tau_k \in \tau$ and all $\Delta_k \geq D_k$.*

$$\sum_{\tau_i \in \tau} I_1(\tau_i) + I_{\mathrm{carry-in}} \leq w_k \times h_k \tag{27}$$

To the best of our knowledge, no schedulability tests other than the [BAR] test are exact even for uniprocessor systems. Our test provided in Theorem 1 also succeeds this superior property of the [BAR] test. In fact, our test is a generalization of the [BAR] test.

Consider the case in which $v_i = 1$ holds for all tasks $\{\tau_i \in \tau\}$. From Equation (6) and (8), $I_1(\tau_i)$ and $I_2(\tau_i)$ are rewritten as follows.

$$I_1(\tau_i) = \begin{cases} \min\{\mathrm{hbf}(\tau_i, \Delta_k), w_k\} & \text{if } i \neq k \\ \min\{\mathrm{hbf}(\tau_i, \Delta_k) - C_k, A_k\} & \text{otherwise} \end{cases}$$

$$I_2(\tau_i) = \begin{cases} \min\{\mathrm{hbf}'(\tau_i, \Delta_k), w_k\} & \text{if } i \neq k \\ \min\{\mathrm{hbf}'(\tau_i, \Delta_k) - C_k, A_k\} & \text{otherwise} \end{cases}$$

Notice that w_k can be rewritten as $A_k + D_k - C_k$. We also know (i) $\mathrm{hbf}(\tau_i, L) = \mathrm{dbf}(\tau_i, L)$, (ii) $h_k = m$, and (iii) the number of tasks containing carry-in jobs is at most $m - 1$, for the special case $v_i = 1$. Condition (27) can be therefore rewritten as:

$$\sum_{\tau_i \in \tau} I_1(\tau_i) + \sum_{\text{the } (m-1) \text{ largest}} I_{\mathrm{diff}}(\tau_i) \leq m(A_k + D_k - C_k).$$

The above expression is exactly the same as the schedulability condition of the [BAR] test (presented as **Theorem 2** in [4]). Our claim is thus true.

Corollary 1. *The schedulability test derived in Theorem 1 is a generalization of the exact schedulability test [6] designed for the uniprocessor EDF scheduling, as well as the [BAR] test [4] designed for the multiprocessor Global EDF scheduling.*

As we stated before, the range of the check points for Δ_k in Condition (27) can be reduced to a finite number. The following theorem extends the analysis of the [BAR] test (presented as **Theorem 3** in [4]) to clarify the range of the check points.

Theorem 2. *If Condition* (27) *is to be violated for any $\Delta_k \geq D_k$, then it is violated for some $\Delta_k \geq D_k$ satisfying Condition* (28)*, where $C_{\mathrm{carry-in}}$ denotes $\sum_{\tau_i \in \tau_{\mathrm{carry-in}}} C_i$.*

$$\Delta_k \leq \frac{h_k C_k - \sum_{\tau_i \in \tau} \{(D_i - T_i) U_i \times \min(v_i, h_k)\} + C_{\mathrm{carry-in}}}{h_k - \sum_{\tau_i \in \tau} U_i \times \min(v_i, h_k)} \tag{28}$$

Proof. Equation (2) and Condition (6) tell us:

$$\begin{aligned} I_1(\tau_i) &\leq \mathrm{hbf}(\tau_i, \Delta_k) \times \min(v_i, h_k) \\ &\leq (\Delta_k - D_i + T_i) \times U_i \times \min(v_i, h_k). \end{aligned}$$

Given also that the amount of the carry-in execution of τ_i is at most C_i, we can derive:

$$I_2(\tau_i) \leq I_1(\tau_i) + C_i \times \min(v_i, h_k).$$

To violate Condition (27), the following must be satisfied.

$$\begin{aligned} &\sum_{\tau_i \in \tau} \mathrm{hbf}(\tau_i, \Delta_k) \times \min(v_i, h_k) + C_{\mathrm{carry-in}} > w_k \times h_k \\ \Rightarrow &\sum_{\tau_i \in \tau} \{(\Delta_k - D_i + T_i) \times U_i \times \min(v_i, h_k)\} + C_{\mathrm{carry-in}} \\ &> (\Delta_k - C_k) \times h_k \\ \Rightarrow &\Delta_k \leq \frac{h_k C_k - \sum_{\tau_i \in \tau} \{(D_i - T_i) U_i \times \min(v_i, h_k)\} + C_{\mathrm{carry-in}}}{h_k - \sum_{\tau_i \in \tau} U_i \times \min(v_i, h_k)} \end{aligned}$$

Thus, the theorem is true. □

Theorem 2 implies that Condition (27) needs to be tested at only those values of $\Delta_k \geq D_k$ at which $\mathrm{hbf}(\tau_i, \Delta_k)$ changes for some τ_i. It follows that the schedulability test in Theorem 1 can be done in time pseudo-polynomial.

5 Related Work

In recent years, the preemptive EDF scheduling of sporadic task systems on identical multiprocessors has been much studied in algorithm design [14, 9, 37, 1, 26, 27] as well as schedulability analysis [23, 2, 3, 7, 10, 4, 5]. All these work, however, consider the sequential task model where any individual job uses at most one processor at any time instant. Meanwhile, we here considered the parallel task model where an individual job may be executed in parallel on more than one processor. Remember that this work generalized the result in [4], so our result is still valid for the sequential task model.

In parallel processing, several task models have been introduced [20, 11, 39]. According to [17], most parallel applications are *moldable*, that is, the number of processors to execute a parallel task is not fixed but determined before the execution. This number does not change until the completion of the parallel task. The scheduling of such moldable parallel task systems is often based on *Gang scheduling* or *Coscheduling* [34, 22, 19]. Gang scheduling and Coscheduling have also been extensively studied for many objectives [13, 18, 21, 41, 42, 43]. Performance comparison of several task dispatching policies in Gang scheduling was reported in [25]. The primary objective of these work is, however, to minimize *make-span* or *response time* of the given task system. Unlike general-purpose systems, timing constraints such as release times and deadlines must be concerned in real-time systems. We therefore combined the EDF policy and the Gang scheduling scheme, to realize the real-time scheduling of parallel task systems.

The problem of scheduling parallel tasks in general-purpose systems is summarized in [16, 17], and is proved to be NP-hard or NP-complete, regardless of preemptive and non-preemptive assumptions. As reported in [24], it is easily seen that the problem of scheduling parallel tasks

in real-time systems is also NP-hard or NP-complete, since we have additional constraints, namely, release times and deadlines. In this paper, we presented an efficient on-line algorithm, Gang EDF, for the real-time scheduling of parallel task systems. Particularly, we aimed at verifying if a given task system is successfully scheduled by Gang EDF.

Unlike general-purpose systems described above, real-time systems have not received much attention in the problem of scheduling parallel tasks. Han *et al.* in [24] proved that the preemptive fixed-priority scheduling of parallel task systems is NP-hard. Drozdowski in [15] studied the scheduling of linear-speedup parallel tasks with release times but not with deadlines. The EDF scheduling of moldable parallel tasks was investigated by Manimaran *et al.* in [32], but they presumed non-preemptive tasks, while we dealt with preemptive tasks. Kwon *et al.* in [28] explored the preemptive EDF scheduling of moldable and malleable parallel task systems with linear-speedup parallelism. In fact, the subject of [28] is similar to this work, but they focused on on-line heuristics to maximize the total amount of work that complete before deadlines, while we focused on schedulability analysis to guarantee a given task system schedulable. The optimal real-time scheduling of malleable parallel task systems was considered by Lee *et al.* in [30] and Collette *et al.* in [12]. The difference between them is that Lee *et al.* assumed arbitrary-deadline systems with linear-speedup parallelism, while Collette *et al.* did implicit-deadline systems with work-limited parallelism. Given that the malleability is not widely supported in practical applications [17], we restricted our attention to the moldable task model.

6 Conclusion

In this paper, we studied the Gang EDF scheduling of preemptive, sporadic, constrained-deadline, and moldable parallel task systems. Gang EDF dispatches the earliest-deadline jobs in those which can execute all the threads simultaneously at any time instants. In other words, later-deadline jobs may be dispatched ahead of earlier-deadline jobs, if the number of available processors is less than the requirement of the earlier-deadline jobs. Apart from this spacial limitation on the number of used processors, Gang EDF is subject to Global EDF.

In schedulability analysis, we derived the schedulability test of Gang EDF, with its basis on the Global EDF schedulability test (the [BAR] test) presented by Baruah in [4]. Our test is in fact a generalization of the [BAR] test. That is, for the sequential task model, our test can output the same result as the [BAR] test. To the best of our knowledge, this is the first research that has focused on the preemptive real-time scheduling of sporadic moldable parallel task systems and has provided the schedulability test of Gang EDF.

We still have many issues left open for future work, since the subject of this research is a new field. Though we focused on the subject under the assumption that the number of threads (used processors) for each parallel task is known or given by designers, it is an interesting challenge to dynamically determine the number so that we can obtain schedulability improvements. In this case, it is important to take into account speedup models, such as linear speedup and work-limited speedup. We will also attempt to develop new algorithms beyond Gang EDF. Given that Gang EDF is an integration of Gang scheduling and Global EDF, other global scheduling algorithms, such as EDF-US [37], EDZL [9], and EDCL [26], are worth being considered. Furthermore, finding optimal scheduling algorithms for the moldable parallel task model is a challenging issue.

References

[1] B. Andersson, K. Bletsas, and S. Baruah. Scheduling Arbitrary-Deadline Sporadic Task Systems Multiprocessors. In *Proc. of the IEEE Real-Time Systems Symposium*, pages 385–394, 2008.

[2] T.P. Baker. Multiprocessor EDF and Deadline Monotonic Schedulability Analysis. In *Proc. of the IEEE Real-Time Systems Symposium*, pages 120–129, 2003.

[3] T.P. Baker. An Analysis of EDF Schedulability on a Multiprocessor. *IEEE Transactions on Parallel and Distributed Systems*, 16:760–768, 2005.

[4] S. Baruah. Techniques for Multiprocessor Global Schedulability Analysis. In *Proc. of the IEEE Real-Time Systems Symposium*, pages 119–128, 2007.

[5] S. Baruah and T. Baker. Schedulability Analysis of Global EDF. *Real-Time Systems*, 38(3):223–235, 2008.

[6] S. Baruah and A. Mok. Preemptively Scheduling Hard-Real-Time Sporadic Tasks on One Processor. In *Proc. of the IEEE Real-Time Systems Symposium*, pages 182–190, 1990.

[7] M. Bertogna, M. Cirinei, and G. Lipari. Improved Schedulability Analysis of EDF on Multiprocessor Platforms. In *Proc. of the Euromicro Conference on Real-Time Systems*, pages 209–218, 2005.

[8] OpenMP Architecture Review Board. OpenMP C and C++ application program interface. http://www.openmp.org/. Version 1.0.

[9] S. Cho, S.K. Lee, A. Han, and K.J. Lin. Efficient Real-Time Scheduling Algorithms for Multiprocessor Systems. *IEICE Transactions on Communications*, E85-B(12):2859–2867, 2002.

[10] M. Cirinei and T.P. Baker. EDZL Scheduling Analysis. In *Proc. of the Euromicro Conference on Real-Time Systems*, pages 9–18, 2007.

[11] W. Cirne and F. Berman. A Model for Moldable Supercomputer Jobs. In *Proc. of the IEEE International Parallel and Distributed Processing Symposium*, 2001.

[12] S. Collette, L. Cucu, and J. Goossens. Integrating Job Parallelism in Real-Time Scheduling Theory. *Information Processing Letters*, 106:180–187, 2008.

[13] J. Corbalan, X. Martorell, and J. Labarta. Improving Gang Scheduling through Job Performance Analysis and Malleability. In *Proc. of the International Conference on Supercomputing*, pages 303–311, 2001.

[14] S.K. Dhall and C.L. Liu. On a Real-Time Scheduling Problem. *Operations Research*, 26:127–140, 1978.

[15] M. Drozdowski. Real-Time Scheduling of Linear Speedup Parallel Tasks. *Information Processing Letters*, 57:35–40, 1995.

[16] M. Drozdowski. *Scheduling Parallel Tasks – Algorithms and Complexity*, chapter 25. Handbook of SCHEDULING Algorithms, Models and Performance Analysis. CHAPMAN & HALL/CRC, 2004.

[17] P-F. Dutot, G. Mounie, and Denis Trystram. *Scheduling Parallel Tasks – Approximation Algorithms*, chapter 26. Handbook of SCHEDULING Algorithms, Models and Performance Analysis. CHAPMAN & HALL/CRC, 2004.

[18] D.G. Feitelson and M.A. Jette. Improved Utilization and Responsiveness with Gang Scheduling. In *Job Scheduling Strategies for Parallel Processing*, volume 1291 of *Lecture Notes in Computer Science*, pages 238–261, 1997.

[19] D.G. Feitelson and L. Rudolph. Gang Scheduling Performance Benefits for Fine-Grain Synchronization. *Journal of Parallel and Distributed Computing*, 16:306–318, 1992.

[20] D.G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. Sevcik, and P. Wong. Theory and Practice in Parallel Job Scheduling. In *Job Scheduling Strategies for Parallel Processing*, volume 1291 of *Lecture Notes in Computer Science*, pages 1–34, 1997.

[21] E. Frachtenberg, D.G. Feitelson, F. Petrini, and J. Fernandez. Adaptive Parallel Job Scheduling with Flexible Coscheduling. *IEEE Transactions on Parallel and Distributed Systems*, 16(11):1066–1077, 2005.

[22] E.F. Gehringer, D.P. Siewiorek, and Z. Segall. *Parallel Processing: The Cm* Experience*. Digital Press, 1987.

[23] J. Goossens, S. Funk, and S. Baruah. Priority-driven Scheduling of Periodic Task Systems on Multiprocessors. *Real-Time Systems*, 25:187–205, 2003.

[24] C. Han and K.-J. Lin. Scheduling Parallelizable Jobs on Multiprocessors. In *Proc. of the IEEE Real-Time Systems Symposium*, pages 59–67, 1989.

[25] H.D. Karatza. Performance of Gang Scheduling Strategies in a Parallel System. *Simulation Modelling Practice and Theory*, 17(2):430–441, 2009.

[26] S. Kato and N. Yamasaki. Global EDF-based Scheduling with Efficient Priority Promotion. In *Proc. of the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 197–206, 2008.

[27] S. Kato, N. Yamasaki, and Y. Ishikawa. Semi-Partitioned Scheduling of Sporadic Task Systems on Multiprocessors. In *Proc. of the Euromicro Conference on Real-Time Systems*, 2009.

[28] O.-H. Kwon and K.-Y. Chwa. Scheduling Parallel Tasks with Individual Deadlines. *Theoretical Computer Science*, 215(1):209–223, 1999.

[29] S. Kwon, Y. Kim, W.-C. Jeun, S. Ha, and Y. Paek. A Retargetable Parallel-Programming Framework for MPSoC. *ACM Transactions on Design Automation of Electronic Systems*, 13(3):Article 39, 2008.

[30] W.Y. Lee and H. Lee. Optimal Scheduling for Real-Time Parallel Tasks. *IEICE Transactions on Information and Systems*, E89-D(6):1962–1966, 2006.

[31] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *Journal of the ACM*, 20:46–61, 1973.

[32] G. Manimaran, C. Siva Ram Murthy, and K. Ramamritham. A New Approach for Scheduling of Parallelizable Tasks in Real-Time Multiprocessor Systems. *Real-Time Systems*, 15:39–60, 1998.

[33] G. Martin. Overview of the MPSoC Design Challenge. In *Proc. of the ACM/IEEE Design Automation Conference*, pages 274–279, 2006.

[34] J.K. Ousterhout. Scheduling Techniques for Concurrent Systems. In *Proc. of the IEEE International Conference on Distributed Computing Systems*, pages 22–30, 1982.

[35] K. Sankaralingam, R. Nagarajan, H. Liu, J. Huh, C.K. Kim, D. Burger, S.W. Keckler, and C.R. Moore. Exploiting ILP, TLP, and DLP with the Polymorphous TRIPS Architecture. In *Proc. of the International Symposium on Computer Architecture*, pages 422–433, 2003.

[36] J. Shirako, N. Oshiyama, Y. Wada, H. Shikano, K. Kimura, and H. Kasahara. Compiler Control Power Saving Scheme for Multi Core Processors. In *Languages and Compilers for Parallel Computing*, volume 4339 of *Lecture Notes in Computer Science*, pages 362–376, 2006.

[37] A. Srinivasan and S.K. Baruah. Deadline-based Scheduling of Peroidic Task Systems on Multiprocessors. *Information Processing Letters*, 84(2):93–98, 2002.

[38] M.B. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, J.-W. Lee, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, and A. Agarwal. The Raw Microprocessor: A Computational Fabric for Software Circuits and General-Purpose Programs. *IEEE Micro*, 22(2):25–35, 2002.

[39] D. Trystram. Scheduling Parallel Applications Using Malleable Tasks on Clusters. In *Proc. of the IEEE International Parallel and Distributed Processing Symposium*, 2001.

[40] D.W. Walker and J.J. Dongarra. MPI: A Standard Message Passing Interface. *Supercomputer*, 12:56–68, 1996.

[41] Y. Wiseman and D.G. Feitelson. Paired Gang Scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 14(6):581–592, 2003.

[42] Y. Zhang, H. Franke, J. Moreira, and A. Sivasubramaniam. An Integrated Approach to Parallel Scheduling using Gang-Scheduling, Backfilling and Migration. *IEEE Transactions on Parallel and Distributed Systems*, 14(3):236–247, 2003.

[43] Y. Zhang, A. Yang, A. Sivasubramaniam, and J. Moreira. Gang Scheduling Extensions for I/O Intensive Workloads. In *Job Scheduling Strategies for Parallel Processing*, volume 2862 of *Lecture Notes in Computer Science*, pages 183–207, 2003.