Controlling Preemption for Better Schedulability in Multi-Core Systems

Jinkyu Lee and Kang G. Shin
Dept. of Electrical Engineering and Computer Science, The University of Michigan, U.S.A.
{jinkyul, kgshin}@eecs.umich.edu

Abstract*—Interest in real-time multiprocessor scheduling has been rekindled as multi-core chips are increasingly used for embedded real-time systems. While tasks may be preemptive or non-preemptive (due to their transactional operations), deadline guarantees are usually made only for those task sets in each of which all tasks are preemptive or non-preemptive, not a mixture thereof, i.e., all or nothing. In this paper, we develop a schedulability analysis framework that guarantees the timing requirements of a given task set in which a task can be either preemptive or non-preemptive. As an example, we apply this framework to the prioritization policy of EDF (Earliest Deadline First), yielding schedulability tests of mpn-EDF (Mixed Preemptive/Non-preemptive EDF), which is a generalization of both fp-EDF (fully-preemptive EDF) and np-EDF (non-preemptive EDF). In addition to their deadline guarantees for any task set that is composed of a mixture of preemptive and non-preemptive tasks, the tests outperform the existing schedulability tests of np-EDF (a special case of mpn-EDF) by up to 109.1%. Using these tests, we also improve schedulability by disallowing preemptions of some preemptive tasks. For this, we develop an algorithm that optimally disallows preemption of a preemptive task under a certain assumption, and demonstrate via simulation that the algorithm discovers up to 30.9% additional task sets that are schedulable with the proposed scheduling scheme, but not with fp-EDF or np-EDF.

I. Introduction

With the increasing use of multi-core chips for embedded real-time applications due to their potential for high performance at low cost, there have been numerous real-time scheduling algorithms proposed for multi-core systems, which can be characterized by their prioritization and preemption policies. The prioritization policy determines each task's priority, such as EDF (Earliest Deadline First) and FP (Fixed Priority) [1], while the preemption policy determines the degree of restriction to preemptions, such as the non-preemptive policy that prohibits the preemption of an executing task, and the fully-preemptive policy that always allows a higher-priority task to preempt a lower-priority executing task. Despite the significant advance in scheduling theory to date, there still exists much room to improve real-time multi-core scheduling. For example, no exact feasibility condition has been identified for both fully-preemptive and non-preemptive scheduling of general periodic tasks. Besides, most schedulability analyses have focused on fully-preemptive scheduling.

While some tasks can be preempted at any time during their execution, other tasks should not be preempted due to their transactional operations, e.g., interrupts. Under the fully-preemptive (non-preemptive) policy, all tasks in a set are preemptive (non-preemptive), i.e., it is usually all or nothing. In contrast, we treat the preemption requirement of each task as a *variable*; it is safe to execute a given preemptive task as if it were non-preemptive, but the converse is not. In order to support task sets in each of which tasks have different preemption requirements, or in order to improve schedulability by using the preemption requirement as a *control knob*, more general preemption policies have been developed for real-time uniprocessor scheduling (see [2] for a survey). However, such general preemption policies have not been explored for real-time *multi-core* scheduling.

In this paper, we consider such general preemption policies, and focus on the following questions in real-time multi-core scheduling.

Q1. How can we guarantee the timing requirements of a given set of preemptive and non-preemptive tasks?

Q2. Can we improve schedulability by executing some preemptive tasks non-preemptively, but not conversely? If so, how can we find the optimal "assignment" of non-preemptiveness to each preemptive task?

We first define the MPN (Mixed Preemptive/Non-preemptive) policy, under which each task can be either preemptive or non-preemptive. This policy is a generalization of both fully-preemptive and non-preemptive policies. To address Q1, we choose a popular schedulability analysis [3] designed for scheduling algorithms under the fully-preemptive policy; by analyzing how the execution of a given preemptive task affects that of the other preemptive tasks under a given prioritization (e.g., EDF or FP) policy, we compute how long it takes for a given preemptive task to finish its execution (called the *task response time*). However, it is challenging to extend this analysis to scheduling algorithms that employ the MPN policy because we need to (i) develop a response time analysis framework for both preemptive and non-preemptive tasks, and (ii) analyze the effect of the execution of a preemptive/non-preemptive task on that of other preemptive/non-preemptive tasks (a total of four cases).

While the calculation of the response time of a preemptive task requires knowing when the last unit of its execution is

The work reported in this paper was supported in part by the NSF under grants CNS-0930813 and CNS-113820.

The work reported in this paper was also supported in part by National Research Foundation of Korea Grant funded by the Korean Government (Ministry of Education, Science and Technology) [NRF-2011-357-D00186].

finished, that of a non-preemptive task only needs to know when the *first unit* of its execution starts; once it starts, the remaining execution will run to completion without any interruption. Using these properties, we address (i), developing a schedulability analysis framework for *any* scheduling algorithm that employs the MPN policy. Then, since (ii) depends on the prioritization policy, we choose a target algorithm, mpn-EDF, which adopts MPN and EDF as its preemption and prioritization policies, respectively. Note that like the MPN policy, mpn-EDF is also a generalization of both fp-EDF (fully-preemptive EDF) and np-EDF (non-preemptive EDF). By carefully analyzing the four cases of (ii) under mpn-EDF, and incorporating them into (i), we finally develop schedulability tests of mpn-EDF (simple and improved tests). The proposed mpn-EDF schedulability tests not only guarantee the timing requirements of a given task set that consists of preemptive and non-preemptive tasks, but also outperform the existing schedulability tests of np-EDF [4,5] when they deal with np-EDF, a special case of mpn-EDF. Our simulation results show an up-to-109.1% improvement.

As to Q2, we first investigate how (ii) varies if a given preemptive task is not allowed to be preempted. Based on the results of this investigation, we develop an algorithm that "optimally" disallows preemption of each preemptive task, under the simple schedulability test of mpn-EDF. We then demonstrate via simulation that disallowing preemption of preemptive tasks is also effective even under the improved schedulability test in that it finds up to 30.9% additional task sets which are schedulable with neither np-EDF nor fp-EDF.

In summary, this paper makes the following contributions:

- Introduction of a new preemption policy, MPN, that handles tasks with different preemption requirements, which is, to the best of our knowledge, the first attempt in the area of real-time multi-core scheduling;
- Development of a schedulability analysis framework for any scheduling algorithm that employs the MPN policy, and derivation of a schedulability analysis of mpn-EDF;
- Demonstration of the superior average schedulability of our analysis of np-EDF (a special case of mpn-EDF) over existing analysis techniques; and
- Development of an algorithm by using the schedulability analysis of mpn-EDF to disallow preemption of preemptive tasks, and demonstration of its schedulability improvement over np-EDF and fp-EDF.

The remainder of the paper is organized as follows. In Section II, we present our system model, and recapitulate a schedulability analysis for fully-preemptive algorithms in [3]. In Section III, we develop a new schedulability analysis framework for any scheduling algorithm that employs the MPN policy, and perform a schedulability analysis of mpn-EDF. Section IV presents an algorithm of disallowing preemption of preemptive tasks for better schedulability. In Section V, we evaluate our schedulability analysis of mpn-EDF and the algorithm of disallowing preemption, via simulation. We summarize the related work in Section VI and conclude the paper in Section VII.

II. Background

In this section, we first introduce the system model, assumptions and notations to be used throughout the paper. Then, we scrutinize an existing schedulability analysis technique for fully-preemptive scheduling algorithms in [3], which will be a basis for our schedulability analysis framework for scheduling algorithms with the MPN preemption policy to be developed in Section III.

A. System model, assumptions and notations

Our focus in this paper is placed on a sporadic task model [6] in which a task $\tau_i \in \Phi$ is modeled as (T_i, C_i, D_i), where T_i is the minimum separation between two successive invocations, C_i the worst-case execution time, and D_i the relative deadline of τ_i. Our discussion is confined to implicit ($C_i \le D_i = T_i$) and constrained ($C_i \le D_i \le T_i$) deadline task sets. A task τ_i invokes a series of jobs, each separated from its predecessor/successor by at least T_i time units. We assume a quantum-based time and let the length of a quantum be one time unit, without loss of generality. All task parameters are specified in multiples of the quantum or time unit.

For each $\tau_i \in \mathcal{T}$, we introduce a new additional parameter Y_i, indicating whether τ_i is *preemptive* ($Y_i = 1$) or *non-preemptive* ($Y_i = 0$). That is, if $Y_i = 1$ ($Y_i = 0$), jobs of τ_i can (cannot) be preempted by any other higher-priority job at any time.

The system is assumed to be built with multi-core chips, each of which consists of m identical cores. We also focus on global work-conserving algorithms, i.e., a job can be executed on any core, and a core cannot be left idle if there is an unfinished ready job. For convenience, we will henceforth use the term "scheduling algorithm" to mean "global work-conserving scheduling algorithm." We also assume that a job cannot be executed in parallel.

We will use the terms *carry-in*, *body*, and *carry-out* jobs in an interval of interest, defined as follows.

- A *carry-in* job is released before the interval, but its deadline is within the interval;
- A *body* job has its release time and deadline within the interval; and
- A *carry-out* job is released within the interval, but its deadline is after the interval.

B. An existing schedulability analysis for fully-preemptive scheduling algorithms

As the basis for a schedulability analysis of MPN scheduling algorithms, we choose a response-time based schedulability analysis technique for fully-preemptive scheduling algorithms [3] due to its applicability to various prioritization policies (e.g., fp-EDF, fp-FP and potentially more) and schedulability performance (e.g., the authors [7] showed the test of fp-EDF [3] to be one of the best with respect to average schedulability).

The technique in [3] uses the notion of *interference* [8]. The interference to τ_k in an interval of $[a, b)$ (denoted by $I_k(a, b)$) represents the cumulative length of all intervals in $[a, b)$ such that a job of τ_k is ready to execute but cannot be executed due to other higher-priority ready jobs. Also, the interference of a task τ_i to another task τ_k in an interval of $[a, b)$ (denoted by $I_{k\leftarrow i}(a, b)$) represents the cumulative length of all intervals in $[a, b)$ such that a job of τ_i executes but a job of τ_k cannot, although it is ready for execution. Since a job of τ_k does not execute in a given time slot only when m other jobs execute, the following equation holds under any global work-conserving algorithm [8]:

$$I_k(a, b) = \frac{\sum_{\tau_i \in \Phi - \{\tau_k\}} I_{k\leftarrow i}(a, b)}{m}. \tag{1}$$

Then, a property regarding $I_k(a, b)$ and $I_{k\leftarrow i}(a, b)$ is derived, which is useful for reducing the pessimism in a schedulability analysis.

Lemma 1 (Lemma 4 in [8]): The following inequality holds for any global work-conserving algorithm:

$$I_k(a, b) \geq x \iff \sum_{\tau_i \in \Phi - \{\tau_k\}} \min\Big(I_{k\leftarrow i}(a, b), x\Big) \geq m \cdot x. \tag{2}$$

Proof: The proof can be found from [8], but is outlined here for completeness. A ready, unfinished job of τ_k does not execute in a given time slot when jobs of m other tasks execute (i.e., interfere with the job of τ_k) in the slot. Therefore, if we focus on the cumulative length of x over all intervals in $[a, b)$ such that any job of τ_k is ready to execute but it cannot, each task's interference with τ_k is upper-bounded by x, and the sum of all other tasks' interferences with τ_k should be no less than $m \cdot x$. The opposite direction can be proved by using the definition of $I_k(a, b)$ and $I_{k\leftarrow i}(a, b)$. ■

Using the concept of interference and Lemma 1, the technique in [3] calculates the maximum duration between the release and the completion of any job of task τ_k, i.e., called the *response time* of τ_k. We do this by computing the maximum amount of τ_i's interference with τ_k in an interval of length l starting from the release of any job of τ_k, which is denoted by $I^*_{k\leftarrow i}(l)$, and formally expressed as:

$$I^*_{k\leftarrow i}(l) \triangleq \max_{t \mid \text{the release time of any job of } \tau_k} I_{k\leftarrow i}(t, t+l). \tag{3}$$

We define $I^*_{k\leftarrow i}(l)$ only for $0 \leq l \leq D_k$, since we are interested in meeting the timing requirements.

If the sum of the execution time of τ_k and the maximum interference to τ_k in an interval of length l starting from the release of any job of τ_k is no longer than l, any job of τ_k finishes its execution within l time units after its release. $I^*_{k\leftarrow i}(l)$ is used to express this, leading to the following schedulability test for fully-preemptive scheduling algorithms.

Lemma 2 (Theorem 6 in [3]): When a task set Φ is scheduled by a fully-preemptive algorithm, an upper-bound of the response time of $\tau_k \in \Phi$ is $R_k = R_k^x$ such that $R_k^{x+1} \leq R_k^x$ holds in the following formula, starting from $R_k^0 = C_k$:

$$R_k^{x+1} \leftarrow C_k + \left\lfloor \frac{1}{m} \sum_{\tau_i \in \Phi - \{\tau_k\}} \min\Big(I^*_{k\leftarrow i}(R_k^x), R_k^x - C_k + 1\Big) \right\rfloor. \tag{4}$$

Then, if $R_k \leq D_k$ holds for all $\tau_k \in \Phi$, Φ is schedulable by the fully-preemptive algorithm.

Note that the iteration of Eq. (4) for τ_k halts if $R_k^x > D_k$, meaning that τ_k is deemed unschedulable.

Proof: The proof is given in [3], and the proof structure is the same as that of Eq. (13) in our new schedulability test of MPN scheduling algorithms to be presented in Theorem 1 in Section III. ■

Note that Eq. (4) is a response time analysis framework for a preemptive task since all tasks are preemptive under any fully-preemptive scheduling algorithm.

While the schedulability analysis in Lemma 2 can be applied to any preemptive (global work-conserving) scheduling algorithm, the main difficulty is to calculate $I^*_{k\leftarrow i}(l)$ for a given scheduling algorithm. Calculating the exact $I^*_{k\leftarrow i}(l)$ is generally intractable, and hence, the upper-bounds of $I^*_{k\leftarrow i}(l)$ are computed. Note that when the upper-bounds are calculated, we assume that there is no deadline miss. This is because the schedulability test in [3] aims to find necessary conditions for the "first" deadline miss, as most tests do. Therefore, we will assume this for derivation of all upper-bounds of $I^*_{k\leftarrow i}(l)$ in the rest of the paper.

The schedulability test in [3] calculates two upper-bounds of $I^*_{k\leftarrow i}(l)$: (i) the one for any scheduling algorithm (regardless of preemptive/non-preemptive scheduling) and (ii) the other for specific preemptive scheduling algorithms (e.g., fp-EDF and fp-FP). To develop (i), the test identifies a situation in which the amount of execution of jobs of given τ_i in a given interval is maximized (in Theorem 4 in [3]). Here, we use the notion of task τ_i's slack (denoted by S_i) that represents the minimum interval length between the finishing time and the deadline of any job of τ_i; in other words, any job of τ_i finishes its execution at least S_i time units ahead of its deadline. We will describe how to calculate S_i later in this section.

Fig. 1 shows an execution pattern of τ_i's jobs that maximizes the amount of execution in an interval starting at t. The first job of τ_i is a carry-in job, and starts its execution at t and ends at $t + C_i$. Here $t + C_i - D_i + S_i$ is the job's release time, i.e., the first job starts and finishes its execution as late as possible. Jobs of τ_i are then released and scheduled as soon as possible. In this case, the number of jobs fully executed in $[t, t + l)$ is calculated as:

$$N_i(l) = \left\lfloor \frac{l + D_i - S_i - C_i}{T_i} \right\rfloor. \tag{5}$$

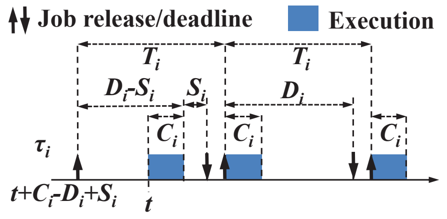


Figure 1. An execution pattern of jobs of τ_i that maximizes the amount of execution of τ_i's jobs in an interval starting at t

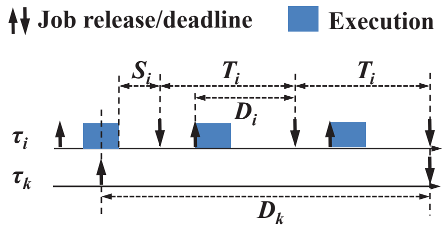


Figure 2. An execution pattern of τ_i's jobs that maximizes the amount of execution of higher-priority jobs of τ_i than a job of τ_k in an interval between the release and the deadline of the τ_k's job under fp-EDF

Then, the maximum amount of execution of jobs of τ_i in $[t, t+l)$ is calculated as (Eq. (8) in [3]):

$$W_i(l) = N_i(l) \cdot C_i + \min\Big(C_i, l + D_i - S_i - C_i - N_i(l) \cdot T_i\Big). \quad (6)$$

While it is guaranteed under any scheduling algorithm (regardless of preemptive/non-preemptive) that $I^*_{k\leftarrow i}(l) \le W_i(l)$ for any τ_k, τ_i and l, we can obtain another upper-bound of $I^*_{k\leftarrow i}(l)$ if we consider the property of a given scheduling algorithm.

Under fp-EDF, earlier-deadline jobs have higher priority (under EDF), and a job can be interfered only by higher-priority jobs (under the fully-preemptive policy). The amount of execution of a preemptive task τ_i's jobs with higher priority than a task τ_k's job is maximized when the deadlines of a job of τ_i and the τ_k's job are aligned as shown in Fig. 2. The number of body jobs of τ_i in an interval between the release time and the deadline of the τ_k's job is then calculated as

$$B_{k\leftarrow i} = \left\lfloor \frac{D_k + T_i - D_i}{T_i} \right\rfloor. \quad (7)$$

Then, the maximum amount of execution of higher-priority jobs of a preemptive task τ_i than a job of a task τ_k in an interval between the release and the deadline of the τ_k's job is calculated as (Eq. (9) in [3]):

$$E_{k\leftarrow i} = B_{k\leftarrow i} \cdot C_i + \min\Big(C_i, \max\big(0, D_k - B_{k\leftarrow i} \cdot T_i - S_i\big)\Big). \quad (8)$$

Finally, $I^*_{k\leftarrow i}(l)$ under fp-EDF is upper-bounded by $\min\big(W_i(l), E_{k\leftarrow i}\big)$ for any (τ_i, τ_k) pair and $0 \le l \le D_k$, so the following inequality holds under fp-EDF, for all $\tau_k \in \Phi$ and $0 \le l \le D_k$:

$$\sum_{\tau_i \in \Phi - \{\tau_k\}} \min\Big(I^*_{k\leftarrow i}(l), l - C_k + 1\Big) \text{ in Eq. (4)}$$
$$\le \sum_{\tau_i \in \Phi - \{\tau_k\}} \min\big(W_i(l), E_{k\leftarrow i}, l - C_k + 1\big) \quad (9)$$

Then, to check the schedulability of a given task set Φ under fp-EDF, we use Lemma 2 with the LHS of Eq. (9) replaced with the RHS. However, one may wonder how to apply the slack S_i when we compute $W_i(l)$ and $E_{k\leftarrow i}$. Depending on whether the slacks are utilized or not, we have two schedulability tests. The first one is to apply Lemma 2 with the upper-bounds in Eq. (9) only once with $S_i = 0, \forall \tau_i \in \Phi$. For a tighter but higher time-complexity analysis, we briefly summarize Section 4.3 of [3]. The basic idea is to repeat the application of Lemma 2 with the upper-bounds in Eq. (9); initially, S_i is set to 0 for all $\tau_i \in \Phi$, and at each iteration S_i is updated by $D_i - R_i$ for all $\tau_i | R_i \le D_i \in \Phi$. The iteration halts when there is no more update for any S_i of $\tau_i \in \Phi$. We call the two schedulability tests *simple* and *improved* tests, and they correspond to Theorem 6 with Eqs. (4) and (5), and Theorem 6 with Eqs. (8) and (9) in [3], respectively.

III. MPN-EDF Scheduling Algorithm and Its Schedulability Analysis

In this section, we first define the MPN preemption policy, in which each task can be either (artificially) preemptive or non-preemptive, and describe mpn-EDF, in which the MPN policy is incorporated into EDF. Then, we develop a schedulability analysis framework for any MPN scheduling algorithm, and perform a schedulability analysis of mpn-EDF.

A. MPN policy and mpn-EDF scheduling algorithm

We consider a preemption policy under which preemption decisions are made based on a task parameter Y_i of τ_i. That is, if $Y_i = 1$ ($Y_i = 0$), a job of τ_i can (cannot) be preempted by any other higher-priority job during its execution. We call this the *mixed preemptive/non-preemptive* (MPN) policy.

Let mpn-EDF denote a scheduling algorithm that adopts MPN and EDF as its preemption and prioritization policies, respectively. Algorithm 1 provides a formal description of mpn-EDF on a multi-core platform. Note that the scheduling overhead of mpn-EDF is not significant in that most steps in Algorithm 1 are also required for fp-EDF.

Then, it is trivial that the MPN policy and mpn-EDF are generalizations of non-preemptive and preemptive policies, and np-EDF and fp-EDF, respectively, as stated in the following lemma.

Lemma 3: The MPN policy subsumes the non-preemptive and preemptive policies, while the mpn-EDF scheduling algorithm subsumes np-EDF and fp-EDF.

Proof: The proof is straightforward. The non-preemptive (preemptive) policy is equivalent to the MPN policy with $Y_i = 0$ ($Y_i = 1$) for all $\tau_i \in \Phi$, and np-EDF

Algorithm 1 mpn-EDF scheduling algorithm

```
Job release: The following steps are performed whenever a job $J_{new}$ of $\tau_i$ is released at $t$:
 1: Set the absolute deadline of $J_{new}$: $d_{new} \leftarrow t + D_i$.
 2: if there is an idle core then
 3:     Start to execute $J_{new}$.
 4: else
 5:     Let $J_{curr}$ denote a currently executing job of a preemptive
        task $\tau_k$ (i.e., $Y_k = 1$), which has the latest deadline $d_{curr}$.
 6:     if $d_{curr} \leq d_{new}$ or all currently executing jobs are invoked
        by non-preemptive tasks (i.e., $Y_k = 0$) then
 7:         Put $J_{new}$ into the wait queue.
 8:     else
 9:         Stop executing $J_{curr}$, put $J_{curr}$ into the wait queue, and
            start to execute $J_{new}$.
10:     end if
11: end if

Job completion: The following step is performed whenever a currently executing job $J_{curr}$ finishes its execution,
 1: Start to execute a job with the earliest deadline in the wait
    queue.
```

(fp-EDF) is equivalent to mpn-EDF with $Y_i = 0$ ($Y_i = 1$) for all $\tau_i \in \Phi$. ■

B. Schedulability analysis of mpn-EDF

In Section II-B, a schedulability analysis of fp-EDF has been developed by addressing the following framework/upper-bound.

- F1. A response time analysis for a *preemptive* task (i.e., Lemma 2); and
- U1. An upper-bound of $I^*_{k\leftarrow i}(l)$ if both τ_k and τ_i are preemptive (i.e., $E_{k\leftarrow i}$ in Eq. (8)).

However, under any scheduling algorithm that employs the MPN preemption policy for a mixture of preemptive and non-preemptive tasks, we need to address the following framework/upper-bounds in addition to F1 and U1.

- F2. A response time analysis for a *non-preemptive* task;
- U2. An upper-bound of $I^*_{k\leftarrow i}(l)$ if τ_k is non-preemptive but τ_i is preemptive;
- U3. An upper-bound of $I^*_{k\leftarrow i}(l)$ if both τ_k and τ_i are non-preemptive; and
- U4. An upper-bound of $I^*_{k\leftarrow i}(l)$ if τ_k is preemptive but τ_i is non-preemptive.

We now address the above issues, starting from F2. To do this, we first introduce a non-preemptive task's property. By definition, any job of a non-preemptive task cannot be interrupted by any other job, and thus, the following property holds.

Observation 1: Once a job of a non-preemptive task starts its first unit of execution, the execution should not be interrupted by any other job. Therefore, if a job of τ_k finishes its first unit of execution at t, it finishes its entire execution no later than $t + C_k - 1$.

Based on the above observation, we can derive an upper-bound of the response time of a given non-preemptive task by calculating when the first unit of execution of any job of the task is finished. Then, the following lemma presents a condition for an upper-bound on the response time.

Lemma 4: The response time of a non-preemptive task τ_k is upper-bounded by $l + C_k - 1$ if the following inequality holds:

$$1 + \left\lfloor \frac{1}{m} \sum_{\tau_i \in \Phi - \{\tau_k\}} \min\left(I^*_{k\leftarrow i}(l), l\right) \right\rfloor \leq l. \tag{10}$$

Proof: We prove the lemma by contradiction. Suppose that Eq. (10) holds for a given l, but the response time of τ_k is longer than $l + C_k - 1$.

In this case, there exists t such that $I_k(t, t+l) \geq l$; otherwise, at least one unit of execution of any job of τ_k is performed within $[t, t+l)$, and then the response time is upper-bounded by $l + C_k - 1$. By Lemma 1 and the definition of $I^*_{k\leftarrow i}(l)$, the following inequality holds:

$$\begin{aligned} & I_k(t, t+l) \geq l \\ \Longleftrightarrow \quad & \sum_{\tau_i \in \Phi - \{\tau_k\}} \min\left(I_{k\leftarrow i}(t, t+l), l\right) \geq m \cdot l \\ \Longrightarrow \quad & \sum_{\tau_i \in \Phi - \{\tau_k\}} \min\left(I^*_{k\leftarrow i}(l), l\right) \geq m \cdot l. \end{aligned} \tag{11}$$

Applying the final result of Eq. (11) to Eq. (10), we show the contradiction, i.e., $1 + l \leq l$. ■

Then, we can develop F2 using Lemma 4, and by merging F1 and F2, we develop a schedulability analysis framework for any global work-conserving scheduling algorithm that employs the MPN policy, as stated in the following theorem.

Theorem 1: When a task set Φ is scheduled by an MPN scheduling algorithm, an upper-bound of the response time of a preemptive task $\tau_k | Y_k = 1 \in \Phi$ is $R_k = R_k^x$ such that $R_k^{x+1} \leq R_k^x$ holds in the following formula, starting from $R_k^0 = C_k$:

$$R_k^{x+1} \leftarrow C_k + \left\lfloor \frac{1}{m} \sum_{\tau_i \in \Phi - \{\tau_k\}} \min\left(I^*_{k\leftarrow i}(R_k^x), R_k^x - C_k + 1\right) \right\rfloor, \tag{12}$$

and an upper-bound of the response time of a non-preemptive task $\tau_k | Y_k = 0 \in \Phi$ is $R_k = F_k^x + C_k - 1$ such that $F_k^{x+1} \leq F_k^x$ holds in the following formula, starting from $F_k^0 = 1$:

$$F_k^{x+1} \leftarrow 1 + \left\lfloor \frac{1}{m} \sum_{\tau_i \in \Phi - \{\tau_k\}} \min\left(I^*_{k\leftarrow i}(F_k^x), F_k^x\right) \right\rfloor. \tag{13}$$

Then, if $R_k \leq D_k$ holds for all $\tau_k \in \Phi$, Φ is schedulable by the algorithm.

Note that the iteration of Eq. (13) (Eq. (12)) for a non-preemptive (preemptive) task τ_k halts if $F_k^x + C_k - 1 > D_k$ ($R_k^x > D_k$), meaning that τ_k is deemed unschedulable.

Proof: By Lemma 2, R_k derived by Eq. (12) is an upper-bound on the response time of a preemptive task τ_k.

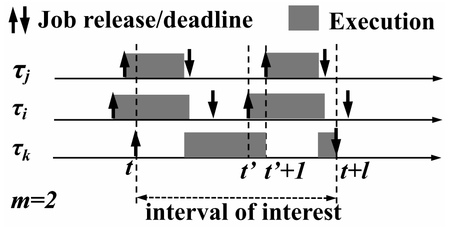


Figure 3. A case where a carry-out job of a non-preemptive task τ_i can interfere with a job of a preemptive τ_k although the priority of the carry-out job of τ_i is lower than that of the job of τ_k under mpn-EDF

Also, by Lemma 4, it is guarantee that the response time of τ_k is upper-bounded by $R_k = F_k^x + C_k - 1$ if $F_k^{x+1} \le F_k^x$. Thus, if $R_k \le D_k$ holds for all $\tau_k \in \Phi$, Φ is schedulable. ∎

In order to develop a schedulability test of mpn-EDF, the next step is to calculate U2, U3 and U4 under mpn-EDF. Note that $E_{k\leftarrow i}$ in Eq. (8) can be an upper-bound of $I^*_{k\leftarrow i}(l)$ when τ_i is preemptive (i.e., $Y_i = 1$) regardless of Y_k. Therefore, we use $E_{k\leftarrow i}$ for U2.

To address U3, we observe how a job of a non-preemptive task interferes with a job of another non-preemptive task.

Observation 2: When both jobs J_A and J_B are non-preemptive, J_A can interfere with J_B only in two cases: (i) the priority of J_A is higher than that of J_B; and (ii) the priority of J_A is lower than that of J_B but the execution of J_A starts before the release of J_B.

Note that J_A cannot interfere with J_B in other than cases (i) and (ii) since J_A cannot be preempted once it starts execution.

Consider an interval $[t, t+l)$ where t is the release time of a job of τ_k and $0 \le l \le D_k$. According to Observation 2, we may consider two cases depending on whether priority inversion occurs or not. The first case is without priority inversion; only higher-priority jobs of τ_i can interfere with the job of τ_k. As shown in the analysis of fp-EDF, in this case $I^*_{k\leftarrow i}(l)$ is upper-bounded by $E_{k\leftarrow i}$ in Eq. (8). The second case is with priority inversion; there should exist a job of τ_i whose execution starts before the interval $[t, t+l)$ and whose deadline is later than $t + D_k$ (i.e., lower-priority than the job of τ_k) according to Observation 2. Therefore, among jobs of τ_i, only one job of τ_i can interfere with the job of τ_k in $[t, t+l)$, and the maximum interference is $C_i - 1$ because at least one unit of execution should be performed before t, the release time of the job of τ_k.

When it comes to U4, one may think that the upper-bound on U3 can be also used for U4. However, this is not true because priority inversion occurs more extensively. Consider an interval $[t, t+l)$ where t is the release time of a job of τ_k and $0 \le l \le D_k$. A carry-out job of τ_i whose deadline is later (i.e., which has lower-priority) than that of the job of τ_k can interfere with the job of τ_k in $[t, t+l)$; note that this carry-out job of τ_i cannot start its execution before the interval because it is released within the interval (by the definition of carry-out jobs). Fig. 3 represents the case with $m = 2$ and three tasks τ_i, τ_j and τ_k. In $[t', t'+1)$, both jobs of a non-preemptive task τ_i and a preemptive task τ_k execute. When a higher-priority job of τ_j is released at $t'+1$, the job of τ_k has to pause its execution due to the higher-priority job of τ_j, but the carry-out job of τ_i does not pause due to its non-preemptiveness. In this case the carry-out job of τ_i interferes with the job of τ_k after $t'+1$ regardless of its priority, and in the worst case the pattern of the interference of τ_i on τ_k is the same as that of the maximum execution in Fig. 1. Therefore, we use $W_i(l)$ for U4,[1] which is an upper-bound of $I^*_{k\leftarrow i}(l)$ in any case.

Note that the case in Fig. 3 cannot occur when both τ_k and τ_i are non-preemptive (i.e., U3). This is because the job of τ_k in the figure cannot be preempted once it starts execution. Therefore, the job of τ_k cannot be preempted at $t'+1$, meaning that a job of τ_i released at t' cannot block the execution of the job of τ_k.

In summary, upper-bounds of $I^*_{k\leftarrow i}(l)$ for the four cases are as follows:

U1. $E_{k\leftarrow i}$ if both τ_k and τ_i are preemptive;
U2. $E_{k\leftarrow i}$ if τ_k is non-preemptive and τ_i is preemptive;
U3. $E_{k\leftarrow i}$ or $C_i - 1$ depending on situations, if both τ_k and τ_i are non-preemptive; and
U4. $W_i(l)$ if τ_k is preemptive and τ_i is non-preemptive.

Then, the only remaining issue is how to handle two cases in U3. While it is safe to set U3 as $\max(E_{k\leftarrow i}, C_i - 1)$, we can do better using the following simple observation.

Observation 3: Consider an interval $[t, t+l)$ where t is the release time of a job of τ_k and $0 \le l \le D_k$. If τ_k is non-preemptive, under mpn-EDF, m is an upper-bound on the number of non-preemptive tasks τ_i ($\neq \tau_k$), each of which invokes a job such that the job has lower-priority than the job of τ_k but interferes with the job of τ_k. This is because the number of jobs executed in any given time slot is upper-bounded by m, and all the jobs of such tasks start their execution before t and continue after t, meaning they execute in $[t-1, t)$.

Based on this observation, we can tightly upper-bound U3 in a safe manner. Considering $I^*_{k\leftarrow i}(l)$ is upper-bounded by $W_i(l)$ in Eq. (6) in any case, the following lemma presents upper-bounds on $\sum_{\tau_i \in \Phi - \{\tau_k\}} \min\left(I^*_{k\leftarrow i}(l), l - C_k + 1\right)$ in Eq. (12) and $\sum_{\tau_i \in \Phi - \{\tau_k\}} \min\left(I^*_{k\leftarrow i}(l), l\right)$ in Eq. (13) under mpn-EDF.

Lemma 5: Under mpn-EDF, the following inequalities hold for all $\tau_k \in \Phi$ and $0 \le l \le D_k$:

If τ_k is preemptive (i.e., $Y_k = 1$),

[1]Actually, one unit of execution can be deducted since the carry-out job of τ_i does not interfere with the job of τ_k in $[t', t'+1)$. We use a safe upper-bound $W_i(l)$ for simplicity of presentation.

$$\sum_{\tau_i \in \Phi - \{\tau_k\}} \min\Big(I^*_{k\leftarrow i}(l), l - C_k + 1\Big) \text{ in Eq. (12)}$$
$$\leq \sum_{\tau_i | Y_i = 1 \in \Phi - \{\tau_k\}} \min\big(W_i(l), E_{k\leftarrow i}, l - C_k + 1\big)$$
$$+ \sum_{\tau_i | Y_i = 0 \in \Phi - \{\tau_k\}} \min\big(W_i(l), l - C_k + 1\big), \quad (14)$$

and if τ_k is non-preemptive (i.e., $Y_k = 0$),

$$\sum_{\tau_i \in \Phi - \{\tau_k\}} \min\Big(I^*_{k\leftarrow i}(l), l\Big) \text{ in Eq. (13)}$$
$$\leq \sum_{\tau_i \in \Phi - \{\tau_k\}} \min\big(W_i(l), E_{k\leftarrow i}, l\big)$$
$$+ \sum_{\text{m largest } \tau_i | Y_i = 0 \in \Phi - \{\tau_k\}} \max\Big(0,$$
$$\min\big(W_i(l), C_i - 1, l\big) - \min\big(W_i(l), E_{k\leftarrow i}, l\big)\Big). \quad (15)$$

Proof: By U1 and U4, and $W_i(l)$, the RHS of Eq. (14) is a safe upper-bound on the LHS. To safely upper-bound the LHS of Eq. (15), we initially add the upper-bound of U2 (which is equivalent to U3 for the case of no priority inversion) for every task. Then, we choose m non-preemptive tasks τ_i which have the largest values of the upper-bound of U3 for the case of priority inversion (i.e., $\min\big(W_i(l), C_i - 1, l\big)$) minus that for the case of no priority inversion (i.e., $\min\big(W_i(l), E_{k\leftarrow i}, l\big)$), and add each difference when it is positive. Then, the RHS of Eq. (15) is a safe upper-bound on the LHS even if any combination of at most m non-preemptive tasks τ_i belong to the case of priority inversion to a non-preemptive task τ_k. ■

Then, to check the schedulability of a given task Φ under mpn-EDF, we use Theorem 1 with the upper-bounds in Lemma 5. Then, the way of using the slack values $\{S_i\}_{\tau_i \in \Phi}$ is the same as that of fp-EDF. We can obtain a *simple* schedulability test of mpn-EDF by setting all S_i to zero, and also an *improved* test of mpn-EDF by iterations for slack reclamation. Then, the schedulability tests of mpn-EDF have the following property.

Lemma 6: The schedulability tests of mpn-EDF in Theorem 1 with the upper-bounds in Lemma 5 with/without slack reclamation (simple/improved tests) generalize the corresponding schedulability tests for fp-EDF in Lemma 2 with the upper-bounds in Eq. (9) with/without slack reclamation, (i.e., Theorem 6 with Eqs. (4) and (5), and Theorem 6 with Eqs. (8) and (9) in [3]).

Proof: The proof is straightforward; the schedulability analysis of fp-EDF is equivalent to that for mpn-EDF with $Y_i = 1$ for all $\tau_i \in \Phi$. ■

Since mpn-EDF also generalizes np-EDF, we highlight that the schedulability test of mpn-EDF in Theorem 1 with the upper-bounds in Lemma 5 is actually a new schedulability test of np-EDF by setting all Y_i to zero. We will demonstrate in Section V that our np-EDF schedulability test outperforms existing schedulability tests of np-EDF.

Algorithm 2 ASSIGNMENT OF $\{Y_i\}_{\tau_i \in \Phi}$

```
1:  while true do
2:      Calculate $R_i, \forall \tau_i \in \Phi$ using Theorem 1 with the upper-bounds in Lemma 5. If $R_i \leq D_i, \forall \tau_i \in \Phi$, return SCHEDULABLE with $\{Y_i\}_{\tau_i \in \Phi}$.
3:      for $\tau_i \in \Phi$ do
4:          if $R_i > D_i$ then
5:              if $Y_i = 0$ then
6:                  Return UNSCHEDULABLE.
7:              else
8:                  $Y_i \leftarrow 0$
9:              end if
10:         end if
11:     end for
12: end while
```

IV. OPTIMAL ASSIGNMENT OF $\{Y_i\}$ FOR MPN-EDF

While allowance/disallowance of preemption is a specification of each task, it does not violate the specification to execute preemptive tasks as if it were non-preemptive. To utilize this for improving schedulability, we first study how the response time of other tasks vary when we execute a preemptive task as a non-preemptive one (i.e. $Y_i = 1 \rightarrow 0$). Based on this, we develop an algorithm that finds an assignment of $\{Y_i\}$. Then, we prove its optimality when the underlying schedulability test of mpn-EDF is the simple one.

The following lemma presents the effect of making a preemptive task non-preemptive on the response time of other tasks.

Lemma 7: Suppose we apply the simple schedulability test of mpn-EDF without slack reclamation, i.e., S_j is set to 0 and does not change for all $\tau_j \in \Phi$. Also, suppose a single preemptive task $\tau_i \in \Phi$ is made non-preemptive (i.e., $Y_i = 1 \rightarrow 0$). Then, $R_k \leq R'_k$ holds, where R_k and R'_k denote the upper-bounds of the response time of τ_k ($\neq \tau_i$) when $Y_i = 1$ and $Y_i = 0$, respectively.

Proof: We consider two cases, i.e., τ_k is preemptive or non-preemptive. If τ_k is preemptive, making a preemptive task τ_i ($\neq \tau_k$) non-preemptive does not decrease the upper-bound on $I^*_{k\leftarrow i}(l)$. (i.e., U1≤U4). If τ_k is non-preemptive, the same holds (i.e., U2≤U3). Since the upper-bound on the interference gets larger, $R_k \leq R'_k$ holds. ■

While our control knob is to make some preemptive tasks non-preemptive, Lemma 7 states the fact that any unschedulable task (whose upper-bound of the response time is larger than it relative deadline) cannot be schedulable by making other preemptive tasks non-preemptive when the simple schedulability test is applied. Therefore, the only way to improve schedulability is to make unschedulable preemptive tasks themselves non-preemptive. Note that if we make a preemptive task τ_k non-preemptive, an upper-bound of $I^*_{k\leftarrow i}(l)$ remains or decreases (i.e., U1=U2, and U4≥U3). Algorithm 2 disallows the preemption of each preemptive task (assignment of $\{Y_i\}_{\tau_i \in \Phi}$) when the response time is greater than the relative deadline (i.e., $R_i > D_i$). Note that in Algorithm 2, we do not change tasks with $Y_i = 0$ to $Y_i = 1$ since it violates their specification. The following lemma proves the optimality of Algorithm 2.

Lemma 8 (Optimality of Algorithm 2): Suppose the simple schedulability test of mpn-EDF is applied, i.e., Theorem 1 with the upper-bounds in Lemma 5 without slack reclamation (S_j is set to 0 and does not change for all $\tau_j \in \Phi$). If Algorithm 2 deems Φ unschedulable, any $\Phi' \triangleq \{\tau'_i\}$ such that τ'_i is the same as τ_i except Y'_i $(\leq Y_i)$ cannot be deemed schedulable by the simple schedulability test of mpn-EDF.

Proof: Suppose that there exists a task set Φ' which is schedulable by the simple schedulability test of mpn-EDF, but Algorithm 2 returns unschedulable. We divide Φ into two disjoint task sets: $\Phi_A = \{\tau_i \in \Phi | Y_i = Y'_i\}$ and $\Phi_B = \{\tau_i \in \Phi | Y_i = 1 \text{ and } Y'_i = 0\}$. Since Φ' is schedulable, at the first iteration of Steps 2–11, tasks in Φ_A satisfy $R_i \leq D_i$ by Lemma 7, and tasks in Φ_B may violate $R_i \leq D_i$. Therefore, at the first iteration, only some tasks in Φ_B can change its Y_i as 0. Similarly, at each iteration, tasks in Φ_A always satisfy $R_i \leq D_i$ by Lemma 7, and each task in Φ_B either satisfies $R_i \leq D_i$ or eventually becomes non-preemptive (i.e., $Y_i = 0$). This means that the algorithm finds another schedulable assignment or tests Φ' eventually. In both cases, the algorithm returns schedulable, which is a contradiction. ■

Then, we can find the optimal disallowance of preemptions with low time-complexity; while a naive approach needs to consider $O(2^n)$ assignments, Algorithm 2 finds the optimal assignment by considering only $O(n)$ assignments, where n is the number of preemptive tasks in a task set Φ.

However, such optimality does not necessarily hold when we apply the improved schedulability test of mpn-EDF, which uses slack reclamation. This is because if we make a preemptive task τ_i non-preemptive, its response time may get decreased, meaning that the slack value may also get increased. This can help reduce the response time of another task τ_k. Despite its non-optimality, Algorithm 2 with the improved test effectively finds a large number of additional schedulable task sets which are not schedulable by the corresponding improved tests of np-EDF and fp-EDF, which will be demonstrated in Section V.

V. Evaluation

In this section, we evaluate the schedulability analysis of mpn-EDF and the algorithm that disallows preemption based on this analysis. We first describe how task sets are generated and then present the average performance improvement of the schedulability test of np-EDF (i.e., a special case of mpn-EDF) over existing np-EDF tests [4, 5]. We also compare the schedulability performance of mpn-EDF with disallowance of preemption, with fp-EDF and np-EDF.

A. Generation of task sets

We generate task sets based on the technique in [9], which has also been widely used elsewhere [10, 11]. There are three input parameters: (a) the task type (constrained or implicit deadlines), (b) the number of cores ($m = 2, 4$ and 8), and (c) individual task utilization (C_i/T_i) distributions (bimodal with parameter:[2] 0.1, 0.3, 0.5, 0.7, or 0.9, or exponential with parameter:[3] 0.1, 0.3, 0.5, 0.7, or 0.9). For each task, T_i is uniformly distributed in $[1, T_{max} = 1000]$, C_i is chosen based on the given bimodal or exponential parameter, and D_i is uniformly distributed in $[C_i, T_i]$ for constrained deadline task systems or D_i is equal to T_i for implicit deadline task systems.

For each combination of (a), (b) and (c), we repeat the following procedure and generate 10,000 task sets, thus resulting in 100,000 task sets for any given m and the type of task sets.

1. Initially, we generate a set of $m + 1$ tasks.
2. In order to exclude unschedulable sets, we check whether the generated task set can pass a necessary feasibility condition [12].
3. If it fails to pass the feasibility test, we discard the generated task set and return to Step 1. Otherwise, we include this set for evaluation. Then, this task set serves as a basis for the next new set; we create a new set by adding a new task into an already created and tested set, and return to Step 2.

B. Average schedulability

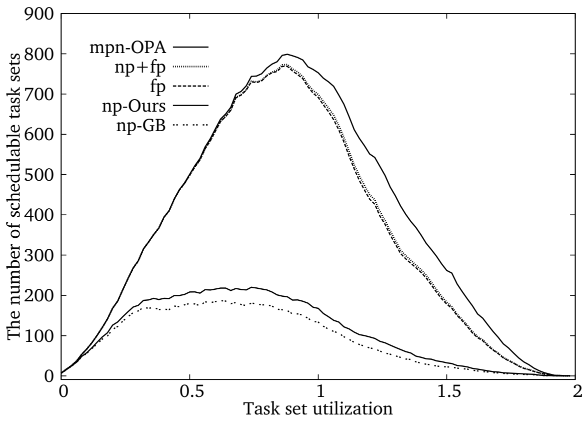
We show the number of task sets schedulable by the following schedulability tests:

- The only existing np-EDF tests [4, 5] (denoted by np-GB), i.e., schedulable by at least one of the tests;
- Our schedulability test for np-EDF, i.e., Theorem 1 with the upper-bounds in Lemma 5 when $\{Y_i = 0\}$ (denoted by np-Ours);
- The existing fp-EDF test [3], which is equivalent to our schedulability test for fp-EDF, i.e., Theorem 1 with the upper-bounds in Lemma 5 when $\{Y_i = 0\}$ (denoted by fp);
- np-Ours and fp (denoted by np+fp), i.e., schedulable by at least one of np-Ours and fp; and
- Our schedulability test of mpn-EDF with disallowance preemption by Algorithm 2 when the initial preemption requirement for each task is $\{Y_i = 1\}$ (denoted by mpn-OPA).

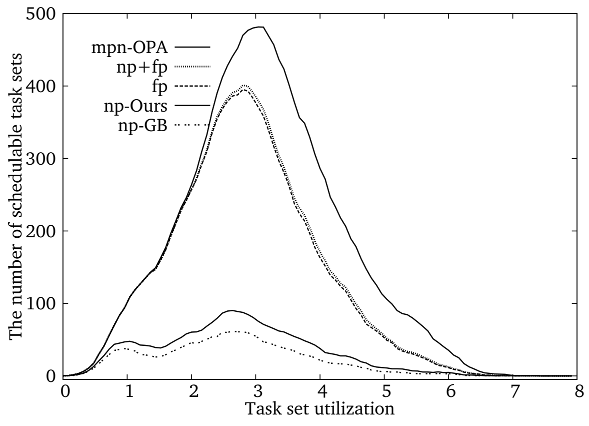
Note that we present our schedulability tests with the best performance; in other words, we present the improved test with slack reclamation for np-Ours, fp, and mpn-OPA.

[2]For a given bimodal parameter p, a value for C_i/T_i is uniformly chosen in $[0, 0.5)$ with probability p, and in $[0.5, 1]$ with probability $1 - p$.

[3]For a given exponential parameter $1/\lambda$, a value for C_i/T_i is chosen according to an exponential distribution whose probability density function is $\lambda \cdot \exp(-\lambda \cdot x)$.

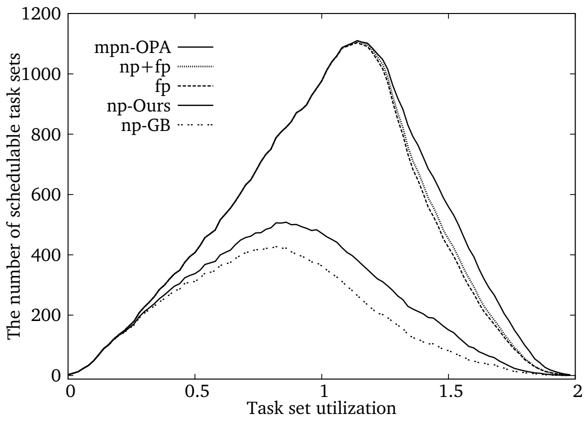


(a) m=2

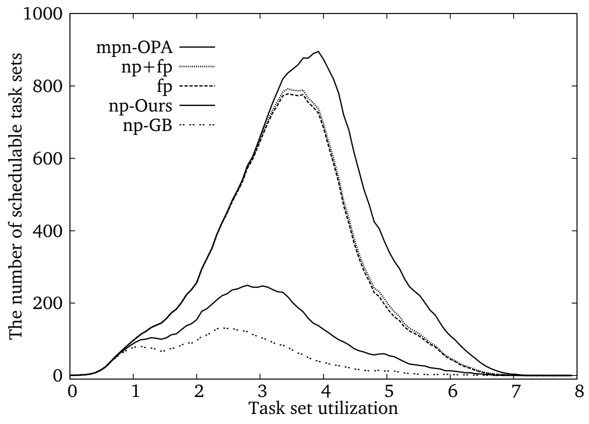


(b) m=8

Figure 4. Schedulability results of constrained deadline task sets



(a) m=2



(b) m=8

Figure 5. Schedulability results of implicit deadline task sets

Figs. 4 and 5 show average schedulability results for constrained and implicit deadline task sets over different number of processors. Of different m, we choose to show for $m = 2$ and 8 since the simulation results with different values of m have similar behaviors. Each figure consists of five lines, each plot showing the number of task sets proven schedulable by each test, with task set utilization (i.e., $U_{sys} \triangleq \sum_{\tau_i \in \Phi} C_i/T_i$) in $[U_{sys} - 0.01 \cdot m, U_{sys} + 0.01 \cdot m)$.

As shown in Figs. 4 and 5, np-Ours outperforms np-GB regardless of the type of task sets and the number of processors (m). However, the degree of improvement gets larger as the number of processors is increasing; np-Ours finds 20.0%, 34.2% and 46.6% additional constrained deadline task sets, which are deemed unschedulable by np-GB, respectively for $m = 2, 4$ and 8. The improvements are more significant in implicit deadline task sets: 25.0%, 79.9%, and 109.1%, respectively, for $m = 2, 4$ and 8. However, of those task sets which are deemed schedulable by np-GB, only up to 0.2% task sets are deemed unschedulable by np-Ours in any case.

There are two reasons why np-Ours outperforms np-GB. First, while np-GB finds the time of completion of the last execution, np-Ours uses the framework of Eq. (13), which identifies the time instant of the first execution of a given job J. Once we find this time instant, any other job executed after that instant cannot block the execution of J, reducing the pessimism of calculating the amount of interference. Second, we have shown in Observation 2 that a non-preemptive job J_k can be blocked only by (i) higher-priority jobs and (ii) jobs that start their execution before the release time of J_k. These two cases, incorporated into the fact that the number of jobs of (ii) is upper-bounded by m, yield a tighter (i.e., less pessimistic) interference bound than those used in np-GB.

We now compare mpn-OPA with np+fp. As shown in the figures, mpn-OPA can find additional schedulable task sets which are schedulable by neither np-Ours nor fp. The fraction of such additional task sets also gets increased, as the number of processors becomes larger; mpn-OPA can find 10.2% (5.0%), 20.9% (12.5%), and 30.9% (21.3%) additional schedulable constrained (implicit) deadline task sets which are deemed schedulable by neither np-Ours nor fp, respectively, for $m = 2, 4$ and 8.

In summary, the proposed schedulability analysis of mpn-EDF significantly improves the schedulability of np-EDF, which is a special case of mpn-EDF. The disallowance of preemption enables discovery of a large number of additional schedulable task sets by exploiting non-preemptiveness as a control knob.

VI. Related Work

For real-time *uniprocessor* systems, researchers have proposed more general preemption policies than the fully-preemptive and non-preemptive policies to meet different goals (e.g., see a survey [2]). Some of them [13, 14] have aimed to improve the schedulability of FP by using the preemption requirement as a control knob, because the fully-preemptive policy is not an optimal preemption policy for FP on a uniprocessor platform. Several studies [15–18] have also focused on expressing broader preemption requirements, such as non-preemptive execution parts and the limit on the number of preemptions, and performed their schedulability analyses for uniprocessor systems. However, little has been done on schedulability analysis for more general preemption policies than fully-preemptive and non-preemptive policies for increasingly popular real-time *multicore* platforms, which is the subject of this paper.

VII. Conclusion

We have proposed the MPN preemption policy, a generalization of preemptive and non-preemptive policies for real-time multicore platforms, and then developed a schedulability analysis framework for MPN scheduling algorithms. We have chosen mpn-EDF as an example and carried out its schedulability analysis, showing that it not only generalizes an existing fp-EDF schedulability analysis, but also outperforms existing np-EDF schedulability analyses. We have also presented a suboptimal assignment algorithm of disallowing preemptions, and demonstrated that the algorithm efficiently finds additional schedulable task sets which are schedulable by neither fp-EDF nor np-EDF.

Although we have only shown the application of the proposed schedulability analysis framework to mpn-EDF, it can be applied to other existing prioritization policies once the interference between tasks under each prioritization policy is upper-bounded. In future, we plan to analyze the schedulability of other MPN scheduling algorithms (e.g., mpn-FP). We would also like to develop a schedulability analysis framework for more general preemption policies than the MPN policy. For example, instead of accommodating only preemptive and non-preemptive tasks, we may also consider a task with limited preemptions in terms of the number and/or the duration of preemptions, which have been already studied for *uniprocessor* scheduling.

References

[1] C. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[2] G. Buttazzo, M. Bertogna, and G. Yao, "Limited preemptive scheduling for real-time systems: a survey," *To appear in IEEE Transactions on Industrial Informatics*.

[3] M. Bertogna and M. Cirinei, "Response-time analysis for globally scheduled symmetric multiprocessor platforms," in *Proceedings of IEEE Real-Time Systems Symposium (RTSS)*, 2007, pp. 149–160.

[4] S. Baruah, "The non-preemptive scheduling of periodic tasks upon multiprocessors," *Real-Time Systems*, vol. 32, no. 1, pp. 9–20, 2006.

[5] N. Guan, W. Yi, Z. Gu, Q. Deng, and G. Yu, "New schedulability test conditions for non-preemptive scheduling on multiprocessor platforms," in *Proceedings of IEEE Real-Time Systems Symposium (RTSS)*, 2008, pp. 137–146.

[6] A. Mok, "Fundamental design problems of distributed systems for the hard-real-time environment," Ph.D. dissertation, Massachusetts Institute of Technology, 1983.

[7] M. Bertogna and S. Baruah, "Tests for global EDF schedulability analysis," *Journal of systems architecture*, vol. 57, pp. 487–497, 2011.

[8] M. Bertogna, M. Cirinei, and G. Lipari, "Improved schedulability analysis of EDF on multiprocessor platforms," in *Proceedings of Euromicro Conference on Real-Time Systems (ECRTS)*, 2005, pp. 209–218.

[9] T. P. Baker, "Comparison of empirical success rates of global vs. paritioned fixed-priority and EDF scheduling for hand real time," Dept. of Computer Science, Florida State University, Tallahasee, Tech. Rep. TR-050601, 2005.

[10] M. Bertogna, M. Cirinei, and G. Lipari, "Schedulability analysis of global scheduling algorithms on multiprocessor platforms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, pp. 553–566, 2009.

[11] J. Lee, A. Easwaran, and I. Shin, "Maximizing contention-free executions in multiprocessor scheduling," in *Proceedings of IEEE Real-Time Technology and Applications Symposium (RTAS)*, 2011, pp. 235–244.

[12] T. P. Baker and M. Cirinei, "A necessary and sometimes sufficient condition for the feasibility of sets of sporadic hard-deadline tasks," in *Proceedings of IEEE Real-Time Systems Symposium (RTSS)*, 2006, pp. 178–190.

[13] Y. Wang and M. Saksena, "Scheduling fixed-priority tasks with preemption threshold," in *Proceedings of IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 1999, pp. 318–335.

[14] M. Bertogna, G. Buttazzo, and G. Yao, "Improving feasibility of fixed priority tasks using non-preemptive regions," in *Proceedings of IEEE Real-Time Systems Symposium (RTSS)*, 2011, pp. 251–260.

[15] S. Baruah, "The limited-preemption uniprocessor scheduling of sporadic task systems," in *Proceedings of Euromicro Conference on Real-Time Systems (ECRTS)*, 2005, pp. 137–144.

[16] G. Yao, G. Buttazzo, and M. Bertogna, "Bounding the maximum length of non-preemptive regions under fixed priority scheduling," in *Proceedings of IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2009, pp. 351–360.

[17] M. Bertogna and S. Baruah, "Limited preemption EDF scheduling of sporadic task systems," *IEEE Transactions on Industrial Informatics*, vol. 6, no. 4, pp. 579–591, 2010.

[18] G. Yao, G. Buttazzo, and M. Bertogna, "Feasibility analysis under fixed priority scheduling with limited preemptions," *Real-Time Systems*, vol. 47, no. 3, pp. 198–223, 2011.