Supplement of "Improvement of Real-Time Multi-Core Schedulability with Forced Non-Preemption"

Jinkyu Lee, Department of Computer Science and Engineering, Sungkyunkwan University, South Korea.

Kang G. Shin, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, MI, U.S.A.

Details of Section 2.2

The technique in [4] uses the notion of *interference* [27]. 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 the execution of other 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 [27]:

$$I_k(a, b) = \frac{\sum_{\tau_i\in\Phi-\{\tau_k\}} I_{k\leftarrow i}(a, b)}{m}. \tag{9}$$

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 8 (Lemma 4 in [27]): The following inequality holds for any global work-conserving algorithm:

$$I_k(a, b) \geq x \iff \sum_{\tau_i\in\Phi-\{\tau_k\}} \min\Big(I_{k\leftarrow i}(a, b), x\Big) \geq m \cdot x. \tag{10}$$

Proof: The proof can be found from [27], 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 8, the technique in [4] 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:

$$I^*_{k\leftarrow i}(l) \triangleq \max_{t\,|\text{ the release time of any job of }\tau_k} I_{k\leftarrow i}(t, t+l). \tag{11}$$

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 greater 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 9 (Theorem 6 in [4]): 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 $R_k = R^x_k$ such that $R^{x+1}_k \leq R^x_k$ holds in the following expression, starting from $R^0_k = C_k$:

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

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. (12) for τ_k halts if $R^x_k > D_k$, meaning that τ_k is deemed unschedulable.

Proof: The proof is given in [4], and the proof structure is the same as that of Eq. (4) in our new schedulability test of MPN scheduling algorithms to be presented in Theorem 1 in Section 4. □

Note that Eq. (12) 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 9 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 [4] 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 [4] 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 [4]). 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

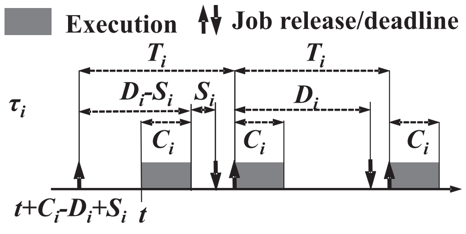


Fig. 2. An execution pattern of jobs of τ_i that maximizes the amount of execution of τ_i's jobs in an interval starting at t

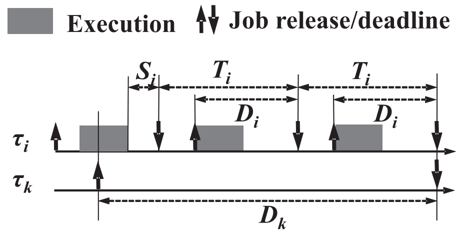


Fig. 3. An execution pattern of τ_i's jobs that maximizes the amount of execution of higher-priority jobs of τ_i than a job of τ_k in an interval between the release and the deadline of the τ_k's job under fp-EDF

its deadline. We will describe how to calculate S_i later in this section.

Fig. 2 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:

$$N_i(l) = \left\lfloor \frac{l + D_i - S_i - C_i}{T_i} \right\rfloor. \tag{13}$$

Then, the maximum amount of execution of jobs of τ_i in $[t, t + l)$ is calculated as (Eq. (8) in [4]):

$$W_i(l) = N_i(l) \cdot C_i + \min\Big(C_i, l + D_i - S_i - C_i - N_i(l) \cdot T_i\Big). \tag{14}$$

While it is guaranteed under any scheduling algorithm (regardless of preemptive/non-preemptive) that $I^*_{k\leftarrow i}(l) \leq W_i(l)$ 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 priorities (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 priorities 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. 3. 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

$$B_{k\leftarrow i} = \left\lfloor \frac{D_k + T_i - D_i}{T_i} \right\rfloor. \tag{15}$$

Then, the maximum amount of execution of jobs of a task τ_i with higher priorities 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 [4]):

$$E_{k\leftarrow i} = B_{k\leftarrow i} \cdot C_i + \min\Big(C_i, \max\big(0, D_k - B_{k\leftarrow i} \cdot T_i - S_i\big)\Big). \tag{16}$$

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 $0 \leq l \leq D_k$, so the following inequality holds under fp-EDF, for all $\tau_k \in \Phi$ and $0 \leq l \leq D_k$:

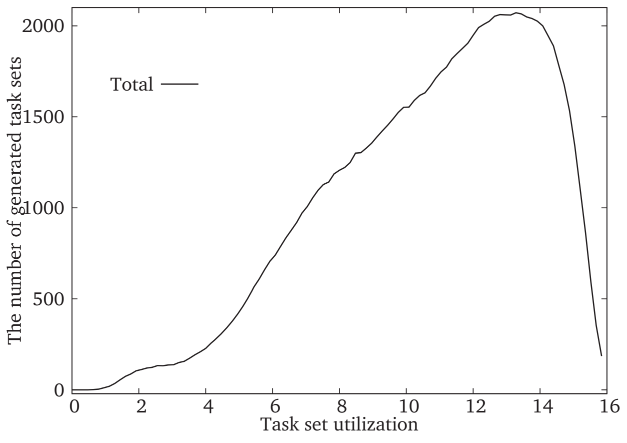
$$\begin{aligned} &\sum_{\tau_i \in \Phi - \{\tau_k\}} \min\Big(I^*_{k\leftarrow i}(l), l - C_k + 1\Big) \text{ in Eq. (12)} \\ \leq &\sum_{\tau_i \in \Phi - \{\tau_k\}} \min\big(W_i(l), E_{k\leftarrow i}, l - C_k + 1\big). \end{aligned} \tag{17}$$

Now, we calculate an upper-bound of $I^*_{k\leftarrow i}(l)$ under fp-FP. Different from fp-EDF, fp-FP maintains task-level priorities; if the priority of τ_i is higher (lower) than that of τ_k, a job of τ_i can (cannot) interfere with a job of τ_k, regardless of their job parameters. Therefore, $I_{k\leftarrow i}(l)$ is upper-bounded by $W_i(l)$ and zero, respectively when τ_i's priority is higher and lower than τ_k. Using the upper-bounds, the following inequality holds under fp-FP, for all $\tau_k \in \Phi$ and $0 \leq l \leq D_k$:

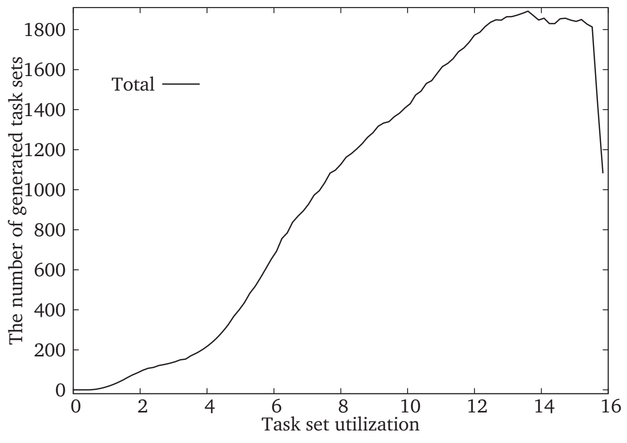
$$\begin{aligned} &\sum_{\tau_i \in \Phi - \{\tau_k\}} \min\Big(I^*_{k\leftarrow i}(l), l - C_k + 1\Big) \text{ in Eq. (12)} \\ \leq &\sum_{\tau_i \in \mathsf{HP}(\tau_k)} \min\big(W_i(l), l - C_k + 1\big). \end{aligned} \tag{18}$$

Then, to check the schedulability of a given task set Φ under fp-EDF or fp-FP, we use Lemma 9 with the LHS of Eq. (17) or (18) 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 for both fp-EDF and fp-FP. The first one is to apply Lemma 9 with the upper-bounds in Eq. (17) or (18) 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 [4]. The basic idea is to repeat the application of Lemma 9 with the upper-bounds in Eq. (17) or (18); 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 $\tau_i | R_i \leq D_i \in \Phi$. 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 [4], respectively.

The total time-complexity of the simple and improved tests is then $O(n^2 \cdot \max_{\tau_i \in \Phi} D_i)$ and $O(n^3 \cdot \max_{\tau_i \in \Phi} D_i^2)$, respectively [4].



(a) $m = 16$, constrained deadline task sets



(b) $m = 16$, implicit deadline task sets

Fig. 4. Generated task sets

Details of Section 6.1

There are three input parameters: (a) the task type (constrained or implicit deadlines), (b) the number of cores ($m = 2, 4, 8$ and 16), and (c) individual task utilization (C_i/T_i) distributions (bimodal with parameter:[6] 0.1, 0.3, 0.5, 0.7, or 0.9, or exponential with parameter:[7] 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 [28].
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.

Then, Fig. 4 shows the total number of generated task sets with different task set utilization (i.e., $U_{sys} \triangleq \sum_{\tau_i \in \Phi} C_i/T_i$) in $[U_{sys} - 0.01 \cdot m, U_{sys} + 0.01 \cdot m)$. Among the generated task sets, Figs. 5 and 6 to be presented plot the number of task sets proven schedulable by each schedulability analysis, with task set utilization in $[U_{sys} - 0.01 \cdot m, U_{sys} + 0.01 \cdot m)$. In all figures including Figs. 4, 5 and 6, we selectively choose m values since the task set generation or simulation results with different values of m have similar behaviors.

6. 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 $1 - p$.

7. 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 $\lambda \cdot \exp(-\lambda \cdot x)$.

TABLE 4
Statistics according to the number of tasks for constrained task sets with $m = 4$ and $1.5 \leq U_{sys} \leq 2.0$.

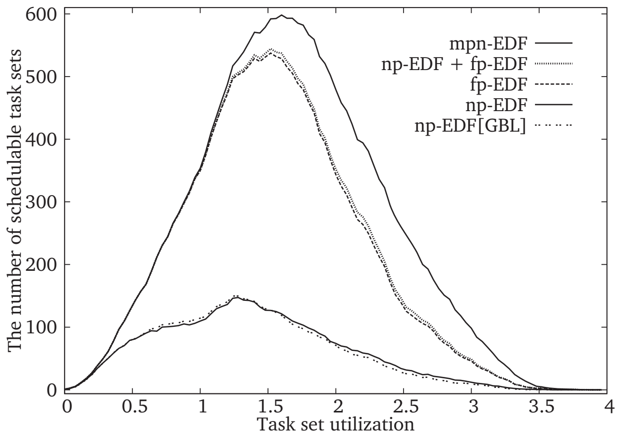
# of tasks	5,6	7,8	9,10	11,12	13,14	15,16
$\frac{\text{mpn-EDF}-\text{fp-EDF}}{\text{fp-EDF}}$ (%)	15.5	34.5	48.5	33.3	21.1	18.2
# of average non-preemptive tasks	1.10	1.13	1.11	1.05	1.0	1.0

Details of Section 6.2

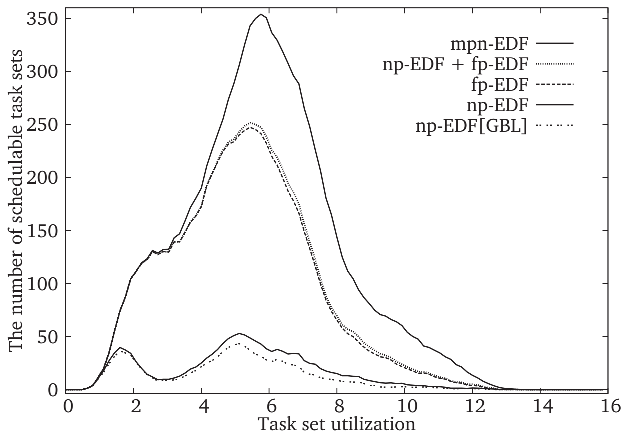
Fig. 5 illustrates the average schedulability results under EDF with different preemption policies, which were discussed in Section 6.2.

To show the effect of the number of tasks on the schedulability improvement by mpn-EDF, we focus on constrained deadline task sets with $m = 4$ and $1.5 \leq U_{sys} \triangleq \sum_{\tau_i \in \Phi} C_i/T_i \leq 2.0$, and investigate two statistics over different numbers of tasks as shown in Table 4: (i) the ratio of the difference between the number of schedulable task sets under mpn-EDF and fp-EDF, to the number of schedulable task sets under fp-EDF (denoted by $\frac{\text{mpn-EDF}-\text{fp-EDF}}{\text{fp-EDF}}$) and (ii) the average number of non-preemptive tasks of each task set under mpn-EDF when the task set is schedulable with mpn-EDF but not with fp-EDF (denoted by # of non-preemptive tasks). The former represents the schedulability improvement by mpn-EDF, while the latter shows how many tasks should be non-preemptive in order to make the task set schedulable.

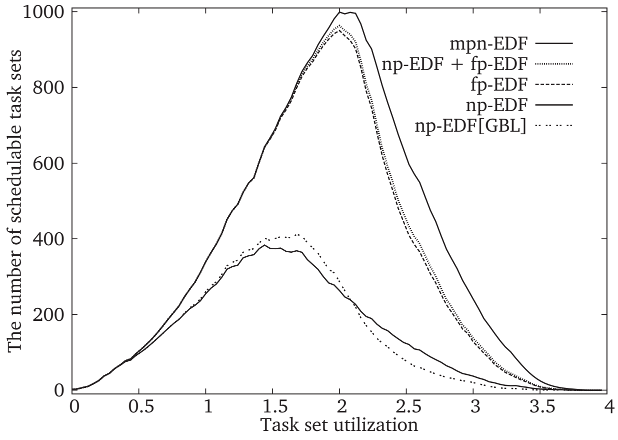
As shown in the table, mpn-EDF enables each task set to be schedulable by making the small number of tasks non-preemptive (i.e., no larger than 1.13 in any case in the table). This is because disallowing preemption of each task makes a negative impact on other tasks' schedulability; if the preemptions of more tasks are



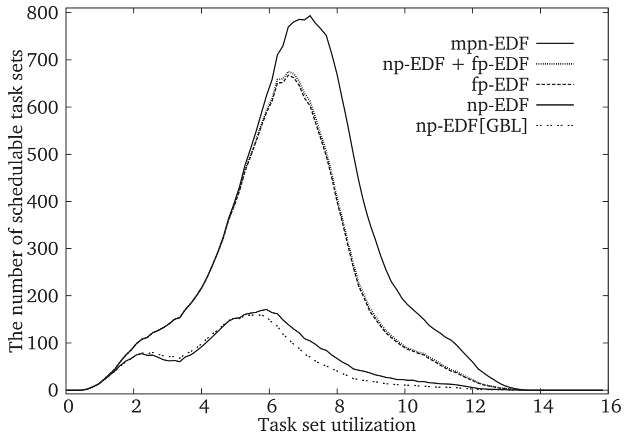
(a) $m = 4$, constrained deadline task sets



(b) $m = 16$, constrained deadline task sets



(c) $m = 4$, implicit deadline task sets



(d) $m = 16$, implicit deadline task sets

Fig. 5. Schedulability results under EDF with different preemption policies

disallowed, the other tasks that were originally schedulable with fp-EDF are more likely to be unschedulable. Therefore, mpn-EDF is less effective on task sets each of whose number of unschedulable tasks with fp-EDF is larger.

On the other hand, the interference-based schedulability framework in Section 2.2 is known to have greater pessimism with more tasks due to over-estimation of each task's interference from carry-in jobs [26]. Therefore, the more tasks in each task set, the more room for improvement.

Due to these two conflicting effects, the improvement by mpn-EDF increases with the number of tasks, and peaks with 9 and 10 tasks (in which the second factor dominates). After that, the improvement decreases (in which the first factor dominates).

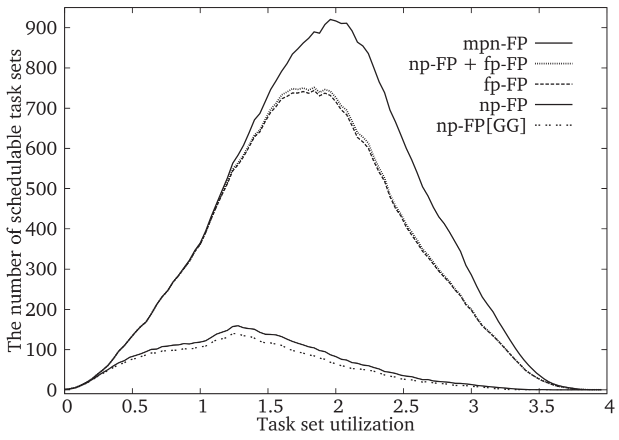
Details of Section 6.3

Similar to the EDF case, we also show the number of task sets schedulable by the following schedulability tests of FP with different preemption policies:

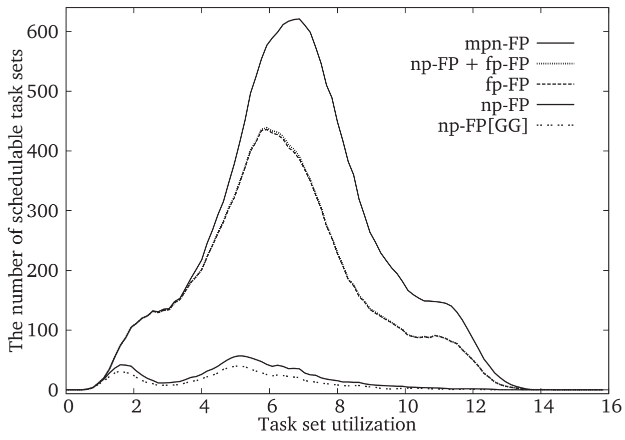
- The only existing np-FP tests [6, 8] (denoted by np-FP[GG]), i.e., schedulable by at least one of the two tests;
- Our schedulability test for np-FP, i.e., Theorem 1 with the upper-bounds in Lemma 4 when $Y_i = 0$ for all $\tau_i \in \Phi$ (denoted by np-FP);
- The existing fp-FP test [4], which is equivalent to our schedulability test for fp-FP, i.e., Theorem 1 with the upper-bounds in Lemma 4 when $Y_i = 1$ for all $\tau_i \in \Phi$ (denoted by fp-FP);
- np-FP and fp-FP (denoted by np-FP+fp-FP), i.e., schedulable by at least one of np-FP and fp-FP; and
- Our schedulability test of mpn-FP with disallowance preemption by Algorithm 2 when the initial preemption requirement is $Y_i = 1$ for all $\tau_i \in \Phi$ (denoted by mpn-FP).

As in Section 6.2, we present our schedulability tests with the best performance, i.e., applying slack reclamation. For fixed-priority of tasks, we deploy DM (Deadline Monotonic) [29], in which the priority of a task τ_k is higher than that of another task τ_i if $D_k < D_i$.

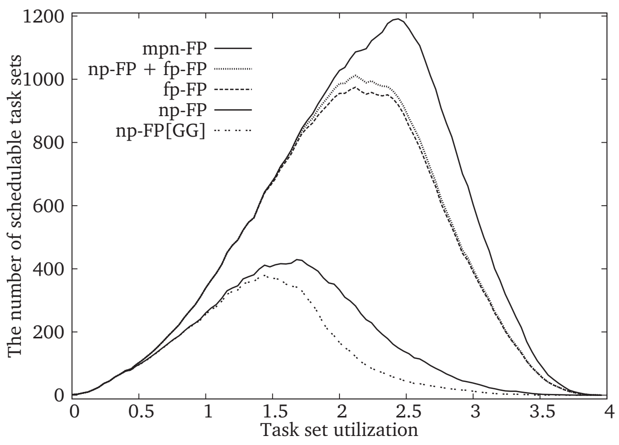
Fig. 6 plots the number of schedulable task sets by each schedulable test with different m and the type of task sets. If we focus on the schedulability under np-FP, np-FP finds 11.3%, 17.3%, 27.2% and 54.5% additional schedulable constrained deadline task sets (likewise 26.7%, 34.6%, 39.5% and 46.3% additional schedulable



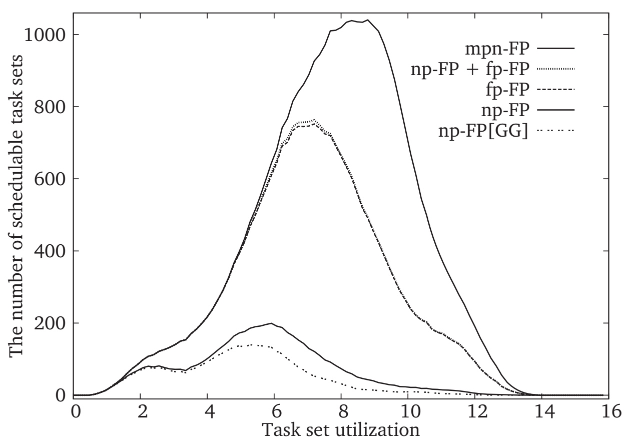
(a) $m = 4$, constrained deadline task sets



(b) $m = 16$, constrained deadline task sets



(c) $m = 4$, implicit deadline task sets



(d) $m = 16$, implicit deadline task sets

Fig. 6. Schedulability results under FP with different preemption policies

implicit deadline task sets), which are not schedulable by np-FP[GG].

The schedulability trend between mpn-FP and np-FP+fp-FP is similar to that between mpn-EDF and np-EDF+fp-EDF. That is, the number of additional schedulable task sets covered by mpn-FP is significant, and the number is increasing as the number of cores becomes larger. The numeric values of task sets which are schedulable by mpn-FP but not by np-FP+fp-FP are 5.3%, 16.0%, 30.9% and 47.6% with implicit task sets (8.7%, 20.5%, 32.6% and 44.3% with constrained task sets) for $m = 2, 4, 8$ and 16, respectively.