# Improved Schedulability Test for Non-Preemptive Fixed-Priority Scheduling on Multiprocessors

Hyeongboo Baek and Jinkyu Lee, *Member, IEEE*

***Abstract*—Non-preemptive scheduling is essential to tasks that inherently disallow any preemption and useful for tasks that exhibit extremely large preemption/migration overhead; however, studies of non-preemptive scheduling have not matured for real-time tasks subject to timing constraints. In this letter, we propose an improved schedulability test for non-preemptive fixed-priority scheduling (NP-FP), which offers timing guarantees for a set of real-time tasks executed on a multiprocessor platform. To this end, we first carefully investigate the NP-FP properties, and present an observation why the existing technique is pessimistic in calculating interference from higher-priority tasks. We then develop a new technique that tightly upper bounds the amount of the interference, and show how to incorporate the technique into the existing schedulability test. Via simulations, we demonstrate that our proposed test improves schedulability performance of NP-FP up to 18%, compared with the state-of-the-art existing tests.**

***Index Terms*—Non-preemptive fixed-priority scheduling (NP-FP), real-time embedded systems, real-time tasks, schedulability analysis.**

## I. Introduction

As MULTIPROCESSOR architectures have been increasingly adopted in real-time embedded systems, the primary concern in designing the systems becomes how to effectively utilize the enlarged computing resources to operate real-time tasks while satisfying their timing constraints (i.e., meeting deadlines of real-time tasks). There have been a number of studies that develop real-time scheduling algorithms and their schedulability tests to efficiently allocate computing resources to real-time tasks while guaranteeing their timing constraints. As a result, studies of pre-emptive scheduling have been sufficiently matured. However, the same cannot be said true for those of non-preemptive scheduling, although non-preemptive scheduling is essential to tasks with extremely large preemption/migration overhead and inherently non-preemptive tasks (e.g., interrupts and transactional operations) [1].

In this letter, our goal is to develop an improved schedulability test for global non-preemptive fixed-priority scheduling (NP-FP) on a multiprocessor platform, where global scheduling allows each task to execute in any processor. Although NP-FP is one of the fundamental non-preemptive scheduling algorithms, there still exists room for improvement of its schedulability analysis on a multiprocessor platform. To this end, we first find a new observation with an example—why the existing technique is pessimistic in calculating *interference*, the amount of execution of other jobs that prevents the execution of the job of interest. In this observation, we demonstrate that if we carefully consider the properties of NP-FP, we can upper bound the amount of the interference more tightly, compared to the existing studies. Based on this observation, we propose a new technique for calculating a tighter upper bound of the interference under NP-FP, and present how to incorporate the new technique into the existing schedulability test, yielding an improved schedulability test for NP-FP.

To demonstrate the schedulability improvement of the proposed schedulability test, we systematically generate a number of task sets, and measure the number of task sets newly covered by the proposed schedulability test (which have not been deemed schedulable by any existing test for NP-FP). We find conditions of settings in which the schedulability improvement by the proposed test is significant, and demonstrate the amount of schedulability improvement is up to 18%.

Manuscript received October 22, 2019; accepted January 11, 2020. Date of publication January 14, 2020; date of current version November 24, 2020. This work was supported by the National Research Foundation of Korea funded by the Korea Government (MSIT) under Grant 2019R1A2B5B02001794 and Grant 2019R1F1A1059663. This manuscript was recommended for publication by H. Tomiyama. *(Corresponding author: Jinkyu Lee.)*

Hyeongboo Baek is with the Department of Computer Science and Engineering, Incheon National University, Incheon 22012, South Korea (e-mail: hbbaek@inu.ac.kr).

Jinkyu Lee is with the Department of Computer Science and Engineering, Sungkyunkwan University, Suwon 16419, South Korea (e-mail: jinkyu.lee@skku.edu).

Digital Object Identifier 10.1109/LES.2020.2966681

## II. System Model, Assumptions, and Notations

In this letter, we consider $\tau$, a set of $n$ sporadic real-time tasks ($\tau_i \in \tau$) [2] scheduled on $m$ identical processors. A task $\tau_i = (T_i, C_i, D_i)$ invokes a series of jobs, each of which is separated from its predecessor by at least $T_i$ time units (*called* the minimum separation or period), and is supposed to finish its execution up to $C_i$ time units (*called* the worst-case execution time) within $D_i$ time units (*called* the relative deadline) from its release. The absolute deadline is a job parameter, which indicates the time instant until which the job should complete its execution; the absolute deadline of a job of $\tau_i$ is set to "the release time of the job"$+D_i$. A task $\tau_i \in \tau$ is said to be schedulable if every job invoked by $\tau_i$ completes its execution no later than its absolute deadline, and a task set $\tau$ is said to be schedulable if every task $\tau_i \in \tau$ is schedulable. A single job is not allowed to execute on more than one processor at the same time. Without loss of generality, we assume quantum-based time where a time unit represents a quantum of length 1.

We target the NP-FP scheduling algorithm. That is, a job cannot be pre-empted once it starts its execution, and jobs are prioritized by fixed task priorities of their invoking tasks. When it comes to notations, let $\tau^{\mathrm{HI}(\tau_k)}$ and $\tau^{\mathrm{LO}(\tau_k)}$ denote sets

of tasks in $\tau$, whose priorities are higher and lower than $\tau_k$, respectively. LHS and RHS stand for the left-hand side and right-hand side, respectively.

## III. Improved Schedulability Test for NP-FP

In this section, we recapitulate the existing schedulability test for NP-FP, called response time analysis (RTA). We then propose an improved schedulability condition by exploiting our own observations for NP-FP. Finally, we present how to incorporate the improved condition into the existing RTA.

### A. Existing RTA for NP-FP

Different from preemptive scheduling, a job under non-preemptive scheduling does not pause once it starts to execute. Therefore, RTA for NP-FP can judge whether a job of $\tau_k$ of interest is schedulable or not, by checking whether the job finishes the first unit of its execution until $(D_k - C_k + 1)$ time units after its release time [4].

Let $I_k(\ell)$ (where $\ell \leq D_k - C_k + 1$) denote the cumulative length of intervals such that a job of $\tau_k$ of interest cannot execute in an interval between the release time of the job and $\ell$ time units after the release time. Then, a job of $\tau_k$ of interest is schedulable if the following inequality holds with at least one $\ell$ ($\leq D_k - C_k + 1$) (shown in [4] and [5] with a different form):

$$1 + I_k(\ell) \leq \ell. \tag{1}$$

This is because, (1) implies that the job of $\tau_k$ of interest executes for at least one time unit within the interval of length $\ell$, which also implies that the job of $\tau_k$ of interest finishes its execution no later than the interval of length $\ell + C_k - 1$ (considering NP-FP is *non-preemptive* scheduling). Therefore, if there is at least one $\ell$ ($\leq D_k - C_k + 1$) that satisfies (1), the job of $\tau_k$ of interest is schedulable.

To upper bound $I_k(\ell)$ under NP-FP, the existing study calculates the largest amount of execution of a task in a consecutive interval of length $\ell$. Let $W_i(\ell)$ denote the maximum amount of execution of jobs of $\tau_i$ in a consecutive interval of length $\ell$, calculated as follows [3]:

$$W_i(\ell) = \left\lfloor \frac{\ell + D_i - C_i - S_i}{T_i} \right\rfloor \cdot C_i + \min\left(C_i, \ell + D_i - C_i - S_i - \left\lfloor \frac{\ell + D_i - C_i - S_i}{T_i} \right\rfloor \cdot T_i\right) \tag{2}$$

where $S_i$ ($\geq 0$) implies a slack value of $\tau_i$, meaning that every job of $\tau_i$ finishes its execution $S_i$ ahead of its absolute deadline. As shown in Fig. 1, $W_i(\ell)$ describes the situation where the execution of the first and last jobs start as late and early as possible, respectively, and the interval of interest of length $\ell$ starts at the beginning of the first job's execution. Then, $W_i(\ell)$ is an upper bound of the amount of execution of jobs of $\tau_i$ in a consecutive interval of length $\ell$ [3].

While it is trivial that a task $\tau_i$ with a higher-priority than $\tau_k$ can prevent the execution of the job of $\tau_k$ of interest in an interval of length $\ell$ during at most $W_i(\ell)$, even a task $\tau_i$ with a lower priority than $\tau_k$ can block the execution of the job of $\tau_k$ of interest when the scheduler is non-preemptive. This happens when a job of $\tau_i$ starts its execution before the release of the job of $\tau_k$ of interest; this is a property of non-preemptive scheduling. In this case, the number of such blocking jobs (which have a lower priority than $\tau_k$) is at most $m$ (i.e., the number of processors), and the blocking time by each job of $\tau_i$ is up to $(C_i - 1)$ time units.

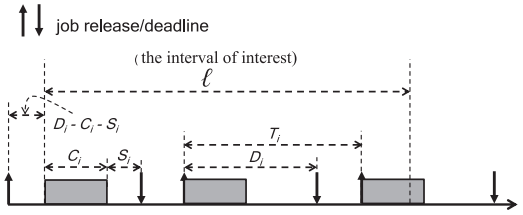


Fig. 1. Worst-case scenario where $W_i(\ell)$ is maximized [3].

Considering that a job of $\tau_k$ cannot be executed in a time slot only if there are $m$ other jobs to be executed in that slot, we can upper bound $I_k(\ell)$ under NP-FP as follows [5]:

$$I_k(\ell) \leq \frac{1}{m} \cdot \left( \sum_{\tau_i \in \tau^{\mathrm{HI}}(\tau_k)} \min(W_i(\ell), \ell) + \sum_{\tau_i \in \tau^{\mathrm{LO}}(\tau_k) | m \text{ largest } C_i} \min(C_i - 1, \ell) \right). \tag{3}$$

Using (3), the following lemma records a schedulability test (called RTA) for NP-FP [5].

*Lemma 1 [5, Lemma 4]:* $\tau$ is schedulable by NP-FP on an $m$-processor platform, if every task $\tau_k \in \tau$ satisfies the following inequality with at least one $\ell \leq D_k - C_k + 1$:

$$1 + \text{ the RHS of (3)} \leq \ell. \tag{4}$$

*Proof:* According to (3), $I_k(\ell)$ is upper bounded by the RHS of (3). Then, the lemma holds by (1) after $I_k(\ell)$ in (1) is replaced by the RHS of (3). See [5] for the detailed proof. ■

Then, RTA effectively finds the value of $\ell$ (for each $\tau_k$) satisfying (4) as follows [5]. Initially, RTA tests whether the inequality holds for $\tau_k$, with $\ell = 1$ and $S_i = 0$ for every $\tau_i \in \tau$. If the inequality holds for $\tau_k$, $\tau_k$ is deemed schedulable. Otherwise, RTA repetitively resets $\ell$ to the previous value of the LHS of the inequality, until the inequality holds (schedulable task) or $\ell > D_k - C_k + 1$ (unschedulable task). If there is any unschedulable task after conducting the above procedure for every $\tau_i \in \tau$ with $S_i = 0$, RTA then updates every $S_i$ of every schedulable task to $(D_i - C_i + 1 - F_i)$ where $F_i$ is the $\ell$ value that satisfies (4) for $\tau_i$ after the above procedure. It repeats calculating $\ell$ of every task with the updated slack values, until all tasks are deemed schedulable (schedulable task set) or there is no more slack value update (unschedulable task set).

### B. Derivation of New Schedulability Conditions for NP-FP

The upper bound of $I_k(\ell)$ [i.e., the RHS of (3)] under the existing RTA for NP-FP can be overestimated due to misunderstanding lower-priority jobs' execution behavior. That is, it successfully upper bounds the amount of execution of lower-priority jobs contributing to $I_k(\ell)$, but it fails to utilize the fact that the execution of lower-priority jobs can block a higher-priority job's execution only when they start their execution

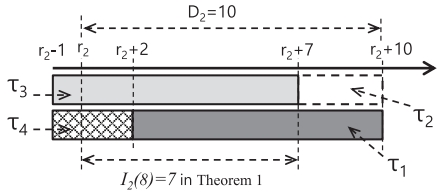


Fig. 2. Example of why a job of $\tau_2$ in Example 1 is schedulable by NP-FP.

before the release of the higher-priority job. The following observation records such failure, where $r_k$ denotes the release time of a job of $\tau_k$ of interest.

*Observation 1:* Although blocking of a job of $\tau_i$ on $\tau_k$ cannot contribute later than $(r_k + C_i - 1)$, the existing RTA allows the blocking to contribute at any time slot in $[r_k, r_k + D_k)$ by the operation of summation of all interference and blocking [i.e., the RHS of (3)], which yields a pessimistic calculation of $I_k(\ell)$.

Here, is an example of how Observation 1 yields a pessimistic RTA.

*Example 1:* Consider a task set $\tau = \{\tau_1(T_1 = 10, C_1 = 8, D_1 = 10), \tau_2(10, 3, 10), \tau_3(100, 8, 100), \tau_4(100, 3, 100)\}$ scheduled by NP-FP scheduling on a two-processor platform (i.e., $m = 2$). The task index is sorted by the priority; $\tau_1$ and $\tau_4$ have the highest and lowest priorities, respectively. According to the existing RTA in Lemma 1, the sequence of the tested target interval length $\ell$ for $\tau_2$ increases in the form of $1 \to 2 \to 4 \to 6 \to 8$. Then, $\tau_2$ is deemed unschedulable, due to $W_1(\ell = 8) = 8$ and the fact that the blocking of each job of $\tau_3$ and $\tau_4$ in $\ell = 8$ are 7 and 2. This yields $I_k(8) = (8 + 7 + 2)/2 = 8 \iff 1 + I_k(8) > 8$, which means there is no $\ell$ satisfying (4). This calculation operates based on an assumption that the blocking of $\tau_3$ (and $\tau_4$) is performed for at most 7 (and 2) at any time slot in $[r_2, r_2 + D_2)$. However, as shown in Fig. 2, $\tau_2$ is schedulable even in the worst-case as it starts its execution no later than $r_2 + 7$. This is because $\tau_2$ has only one higher-priority task (i.e., $\tau_1$), and thus there should be at least one blocking job executing in parallel with $\tau_1$ to interfere $\tau_2$. Also, blocking of $\tau_3$ (and $\tau_4$) cannot be performed after $r_2 + 7$ (and $r_2 + 2$). Otherwise, it executes after a job of $\tau_2$ finishes its execution or in parallel with the job of $\tau_2$, which cannot contribute to prevent $\tau_2$'s execution. Then, the amount of interference (including blocking) can be naturally upper bounded by the longer blocking length between $\tau_3$ and $\tau_4$, which is 7.

We now derive schedulability conditions from Observation 1 and Example 1 as follows. Let $n_k$ denote the number of tasks belonging to $\tau^{\mathrm{HI}(\tau_k)}$. Example 1 implies that how much the execution of the $m$ highest priority tasks is prevented is determined by the possible blocking length of lower-priority tasks in $[r_k, r_k + D_k)$. That is, there should be $m$ higher-priority or blocking jobs in a time slot for $\tau_k$'s execution to be hindered. Since each job cannot execute in parallel, $(m - n_k)$ additional blocking jobs are needed for $n_k$ higher-priority tasks to prevent the execution of $\tau_k$ at each time slot.

Therefore, the job of interest of $\tau_k$ can start its execution no later than the earliest time instant among the finishing times of $(m - n_k)$ lower-priority blocking tasks, each of which occupies a processor that is not used by any of the $n_k$ highest priority tasks. This means, $I_k(\ell)$ cannot be larger than the $(m - n_k)$th largest $(C_i - 1)$ among $\tau_i \in \tau^{\mathrm{LO}(\tau_k)}$. For example, $I_2(\ell)$ in Example 1 for $\tau_2$ is upper bounded by the $(m - n_2) = 1st$ largest $(C_i - 1)$ among $\tau_i \in \tau^{\mathrm{LO}(\tau_2)}$, which is $(C_3 - 1) = 7$. Although $\tau_4$ occupies the processor with the higher-priority task $\tau_1$, the job of interest of $\tau_2$ can start its execution no later than the finishing time of blocking by $\tau_3$, which is $(r_2 + 7)$ as shown in Fig. 2.

Based on the reasoning, we develop another upper bound of $I_k(\ell)$ for the $m$ highest priority tasks as follows.

*Theorem 1:* Under NP-FP, the following inequality holds for every $\tau_k$ with $n_k < m$:

$$I_k(\ell) \le (m - n_k)\text{th longest } (C_i - 1) \text{ among } \tau_i \in \tau^{\mathrm{LO}(\tau_k)}. \quad (5)$$

Note that there exists no such task described in the RHS of (5), the RHS is 0.

*Proof:* We prove that the execution of $\tau_k$ with $n_k < m$ scheduled by NP-FP is prevented at most in $[r_k, r_k + X)$, where $X$ is the $(m - n_k)$th longest $(C_i - 1)$. Suppose that the execution of $\tau_k$ is prevented in $[r_k, r_k + X + Y)$, where $Y > 0$, meaning that there should be $m$ tasks (other than $\tau_k$), each of which occupies a processor in $[r_k, r_k + X + Y)$. Since $\tau_k$ has $n_k$ higher-priority tasks and each task can occupy only one processor in each time slot, the other $(m - n_k)$ processors should be occupied by $\tau_k$'s lower-priority tasks (as blocking). Since every job of a lower-priority task $\tau_i$ can block the job of $\tau_k$ during at most $(C_i - 1)$ time units only if it starts its execution before the job of $\tau_k$'s release, a lower-priority task $\tau_i$ can occupy a processor only in $[r_k, r_k + C_i - 1)$. Therefore, there should exist a lower-priority task $\tau_i$ that cannot occupy a processor after $r_k + X$. This means, the supposition contradicts, which proves the lemma. ■

### C. Improved RTA Exploiting the Derived Conditions

Incorporating Theorem 1 into Lemma 1, we develop a new improved RTA for NP-FP. Although the upper bound of $I_k(\ell)$ calculated by (5) is tighter than that by (3) in most cases, the opposite holds in some cases. For example, let us assume that $C_1$ and $C_3$ in Example 1 are changed to 1 and 9, respectively, and focus on the schedulability of $\tau_2$. According to (5), $I_2(\ell)$ is 8 regardless of the value of $\ell$ so that $\tau_2$ is deemed unschedulable. On the other hand, if we apply the existing RTA with (3), the sequence of the tested target interval length $\ell$ is $1 \to 2 \to 4 \to 5$. In the case of $\ell = 5$, $I_2(5)$ calculated by (3) is 4, meaning that (4) is satisfied and therefore $\tau_2$ is schedulable. Therefore, we take the tighter upper bound of $I_k(\ell)$ between the RHS of (3) and that of (5). The following theorem proposes a new improved RTA for NP-FP.

*Theorem 2:* $\tau$ is schedulable by NP-FP on an $m$-processor platform, if every $\tau_k$ with $n_k < m$ satisfies the following inequality and every $\tau_k$ with $n_k \ge m$ satisfies (4), with at least one $\ell \le D_k - C_k + 1$

$$1 + \min[\text{the RHS of (3), the RHS of (5)}] \le \ell. \quad (6)$$

*Proof:* According to (3) and (5), $I_k(\ell)$ is upper bounded by $\min$ [the RHS of (3), the RHS of (5)]. Then, the theorem holds by (1) after $I_k(\ell)$ in (1) is replaced by $\min$ [the RHS of (3), the RHS of (5)]. ■

Then, the new test finds the value of $\ell$ satisfying (6) using the same mechanism as the existing RTA. According

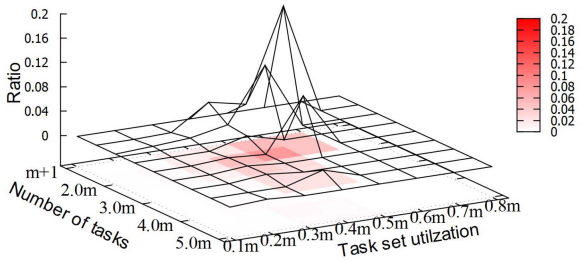


Fig. 3. Ratio between the number of task sets newly schedulable by Ours which have not been deemed schedulable by any of the five existing schedulability tests and the number of task sets deemed schedulable by at least one test among the five tests, for $m = 8$.

to Theorem 1, $\tau_2$ in Example 1 is now deemed schedulable since $\ell = 8$ in RTA satisfies (6), i.e., $1 + I_2(8) = 1 + 7 \leq 8$.

## IV. Evaluation

In this section, we evaluate the performance of our schedulability test for NP-FP proposed in Theorem 2 (denoted by Ours) compared to five existing tests, each denoted by LeSh [5] (i.e., Lemma 1 in this letter), GYG [6], DBM [7], GYD [8], and Lee [1] (i.e., he state-of-the-art schedulability test for NP-FP); note that there exist some other schedulability tests for NP-FP, but they are dominated by the five existing tests. To demonstrate the effectiveness of our approach, we measure how many schedulable task sets are newly found by Ours, which have not been proven to be schedulable by any of the existing five schedulability tests.

To generate task sets for a multiprocessor platform, we use UUnifast-discard [9] exploited in a number of existing studies. The following three input parameters are typically considered for UUnifast-discard: 1) the number of processors $m$ (2, 4, 8, and 16); 2) the number of tasks $n$ ($m+1$, 1.5 m, 2.0 m, 2.5 m, 3.0 m, 3.5 m, 4.0 m, 4.5 m, and 5.0 m); and 3) the task set utilization $U = \sum_{\tau_i \in \tau} C_i/T_i$ (0.1 m, 0.2 m, 0.3 m, 0.4 m, 0.5 m, 0.6 m, 0.7 m, and 0.8 m). For a given combination of $m$, $n$, and $U$, each task's $T_i$ is uniformly chosen in [1, 1000], $C_i$ is computed based on $\sum_{\tau_i \in \tau} C_i/T_i = U$, and $D_i$ is set to $T_i$. As a result, 1000 task sets are generated for each input combination, thereby generating $4 \cdot 9 \cdot 8 \cdot 1000 = 288\,000$ task sets in total.

Fig. 3 shows the ratio between the number of task sets newly schedulable by Ours which have not been deemed schedulable by any of the five existing schedulability tests and the number of task sets deemed schedulable by at least one test among the five tests, and presents the ratios for each combination of $n$ and $U$ for $m = 8$ by considering the rate-monotonic (RM) scheduling as a target FP scheduling algorithm; results under other settings exhibit a similar trend.

The first observation shown in Fig. 3 is that the ratio is almost zero for low (e.g., lower than 0.3 m) and high (e.g., higher than 0.7 m) task set utilization. This is because task sets with low task set utilization are mostly deemed schedulale by the five existing tests, and thus there is little room for better schedulability. For example, 977 (out of 1000) task sets are deemed schedulable by the five existing tests for $n = m + 1$, $U = 0.1$ m, and $m = 8$. With the similar reason, most task sets with high task set utilization are deemed not schedulable by Ours as well as the five existing tests (e.g., no schedulable tasks for $n = 5.0$ m, $U = 0.8$ m, and $m = 8$).

The first observation leads to another one that the ratio sharply increases for a setting with the lower number of tasks (e.g., lower than 2.0 m) and medium task set utilization (e.g., from 0.3 m to 0.7 m). For example, the ratio peaks as 0.18 for $n = 1.5$ m, $U = 0.6$ m, and $m = 8$, meaning that the proposed test improves the schedulability performance with 18% (2 versus 11 task sets) under this setting. Since the number of samples is small, we generate 100 times more task sets (i.e., 100 000 generated task sets) for some settings. Then, the ratio becomes 9.2% (i.e., 91 versus 990 task sets) for the same setting, and 15% (i.e., 5 versus 34 task sets) for $n = 1.5$ m, $U = 0.7$ m, and $m = 8$. This phenomenon stems from the property of Ours that the schedulability of the at most $m$ highest priority tasks can be improved. That is, the improved schedulability of the $m$ highest priority tasks can much affect to the schedulability of the task set containing the smaller number of tasks rather than the larger number of tasks.

Since Ours is developed on top of LeSh (i.e., Lemma 1), the schedulability performance difference between Ours and LeSh demonstrates the improvement by Theorem 1. The ratio between Ours and Lesh peaks as 132% (29 versus 22 task sets) and 129% (i.e., 2601 versus 2016 task sets) for $n = 2.0$ m, $U = 0.5$ m, and $m = 8$ with 1000 and 100 000 generated task sets for the setting, respectively.

## V. Conclusion

In this letter, we derived a simple but effective schedulability condition for NP-FP, and demonstrated its effectiveness in finding additional task sets schedulable by NP-FP, by incorporating the condition to the existing RTA. In the future, we will improve the proposed condition for better schedulability.

## References

[1] J. Lee, "Improved schedulability analysis using carry-in limitation for non-preemptive fixed-priority multiprocessor scheduling," *IEEE Trans. Comput.*, vol. 66, no. 10, pp. 1816–1823, Oct. 2017.

[2] A. K.-L. Mok, "Fundamental design problems of distributed systems for the hard-real-time environment," Ph.D. dissertation, Dept. Elect. Eng. Comput. Sci., Mass. Inst. Technol., Cambridge, MA, USA, 1983.

[3] M. Bertogna and M. Cirinei, "Response-time analysis for globally scheduled symmetric multiprocessor platforms," in *Proc. IEEE Real Time Syst. Symp. (RTSS)*, Tucson, AZ, USA, 2007, pp. 149–160.

[4] J. Lee and K. G. Shin, "Controlling preemption for better schedulability in multi-core systems," in *Proc. IEEE Real Time Syst. Symp. (RTSS)*, San Juan, Puerto Rico, 2012, pp. 29–38.

[5] J. Lee and K. G. Shin, "Improvement of real-time multi-core schedulability with forced non-preemption," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 5, pp. 1233–1243, May 2014.

[6] N. Guan, W. Yi, Z. Gu, Q. Deng, and G. Yu, "New schedulability test conditions for non-preemptive scheduling on multiprocessor platforms," in *Proc. IEEE Real Time Syst. Symp. (RTSS)*, Barcelona, Spain, 2008, pp. 137–146.

[7] R. I. Davis, A. Burns, J. Marinho, V. Nelis, S. M. Petters, and M. Bertogna, "Global and partitioned multiprocessor fixed priority scheduling with deferred preemption," *ACM Trans. Embedded Comput. Syst.*, vol. 14, no. 3, pp. 1–28, 2015.

[8] N. Guan, W. Yi, Q. Deng, Z. Gu, and G. Yu, "Schedulability analysis for non-preemptive fixed-priority multiprocessor scheduling," *J. Syst. Archit.*, vol. 57, no. 5, pp. 536–546, 2011.

[9] R. I. Davis and A. Burns, "Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems," in *Proc. IEEE Real Time Syst. Symp. (RTSS)*, 2009, pp. 398–409.