# New Slack-Monotonic Schedulability Analysis of Real-Time Tasks on Multiprocessors

Risat Mahmud Pathan and Jan Jonsson
Chalmers University of Technology
SE-412 96, Göteborg, Sweden
{risat, janjo}@chalmers.se

***Abstract*—In this paper, multiprocessor scheduling of a set of real-time periodic tasks with implicit deadlines is addressed. We propose two static priority-assignment policies, called policy $P_{bound}$ and policy $P_{search}$. Common for both policies is that a subset of a given task set is assigned the slack-monotonic priority while each of the other tasks is assigned the highest static-priority. The tasks are scheduled on *m* processors using preemptive global multiprocessor scheduling algorithm.**

**First, we show that the utilization bound of global multiprocessor scheduling using our proposed policy $P_{bound}$ is higher than any other existing state-of-the-art static-priority multiprocessor scheduling. Second, using the utilization information of individual task of a given task set, we propose our second priority assignment policy $P_{search}$. We show that global multiprocessor scheduling using policy $P_{search}$ dominates that of using $P_{bound}$ in the sense that any task set schedulable using $P_{bound}$ is also schedulable using $P_{search}$ and not conversely.**

## I. Introduction

Consider the following problem: **Given a collection of *n* implicit-deadline periodic tasks, is it possible to meet all the deadlines of the tasks scheduled on *m* identical, unit-capacity processors?** Real-time task scheduling on multiprocessors is primarily based on either *global* or *partitioned* approach. In global scheduling, a task is allowed to execute on any processor even when it is resumed after preemption. In partitioned scheduling, the task set is grouped in different task partitions during design time and each partition has a fixed processor on which only the tasks of that partition are allowed to execute. The main design goal of many static-priority global [3], [5], [6], [7], [1], [8] and partitioned [4], [12], [16] scheduling algorithms is to derive a *schedulability condition* that is when satisfied implies that all the deadlines are met. One of the most popular and expressive ways to derive a schedulability condition of a scheduling algorithm is in terms of its *utilization bound*.

The utilization bound of a scheduling algorithm $A$ is a number $\texttt{UB}_A$ such that all the tasks will meet their deadlines when scheduled by $A$ on $m$ processors, if the total utilization of the task set is not greater than $\texttt{UB}_A$. It has already been proved that neither global nor partitioned static-priority scheduling can have a utilization bound greater than $0.5m$ on $m$ processors [3], [16]. There exists a static-priority partitioned scheduling algorithm, called `R-BOUND-MP-NFR`, having utilization bound of $0.5m$ [4]. However, the best state-of-the-art utilization bound of global multiprocessor scheduling is $\frac{m^2}{3m-2}$ for $m \leq 5$ (`RM-US` scheduling [3]) and $\frac{2m}{3+\sqrt{5}}$ for $m > 5$ (`SM-US` scheduling [1]). In this paper, we propose *two* static priority-assignment policies, called $P_{bound}$ and $P_{search}$, and prove that global multiprocessor scheduling using either of these policies dominates both `RM-US` and `SM-US` scheduling.

Both of our proposed priority assignment policies are based on *slack-monotonic hybrid priority assignment* scheme that works as follows: if the utilization of a task is not greater than a *threshold utilization*, then the task is given slack-monotonic priority, otherwise, the task is given the highest static-priority. According to slack-monotonic priority assignment, the smaller the difference between a task's period and worst-case execution time (WCET), i.e., called slack, the higher is the assigned priority.

For each of our proposed priority assignment policies, $P_{bound}$ and $P_{search}$, we present the corresponding schedulability analysis of the Global multiprocessor Scheduling algorithms, called $\texttt{GS}_{bound}$ and $\texttt{GS}_{search}$, respectively. The threshold utilizations for policy $P_{bound}$ is determined based on the schedulability analysis of a class of task sets, called "special" task sets (presented in Section V). The threshold utilization we use for policy $P_{bound}$ is $\frac{3m-2-\sqrt{5m^2-8m+4}}{2m-2}$ where $m$ is the number of processors, $m \geq 2$. We prove that, the utilization bound of $\texttt{GS}_{bound}$ is $m \cdot min\{\frac{1}{2}, \frac{3m-2-\sqrt{5m^2-8m+4}}{2m-2}\}$. It is easy to see that the utilization bound of $\texttt{GS}_{bound}$ is higher than (hence, dominates) that of both `RM-US` [3] and `SM-US` [1] scheduling.

Note that, the threshold utilization used for $P_{bound}$ depends only on the number of processors and does not use any information (e.g., utilization) of the individual task of a given task set. Using the utilization information of individual task of a given task set in addition to the information about the number of processors, we propose our second slack-monotonic hybrid priority-assignment policy $P_{search}$. In order to determine the threshold utilization for policy $P_{search}$, we borrowed the idea of determining the job-level priorities of priority-driven $\texttt{EDF}^{(k)}$ scheduling from [13] and applied it to the static-priority setting at the task-level. In $\texttt{EDF}^{(k)}$ scheduling, jobs of the $k$ highest utilization tasks are given the highest priority and the jobs of the remaining $(n-k)$ lowest utilization tasks are given the Earliest-Deadline-First (EDF) priority (in fact, one such $k$ is searched). Based on policy $P_{search}$, we show that scheduling algorithm $\texttt{GS}_{search}$ dominates $\texttt{GS}_{bound}$ in the sense that if a task set is schedulable using $\texttt{GS}_{bound}$ then it is also schedulable using $\texttt{GS}_{search}$ and not conversely. Consequently, $\texttt{GS}_{search}$ dominates both RM-US and SM-US since the utilization bound of $\texttt{GS}_{bound}$ is higher than that of RM-US and SM-US. To this end, we make the following major contributions:

1) We propose priority-assignment policy $P_{bound}$ and show that the utilization bound of $\texttt{GS}_{bound}$ is $m \cdot min\{\frac{1}{2}, \frac{3m-2-\sqrt{5m^2-8m+4}}{2m-2}\}$. This utilization bound is higher than that of the state-of-the-art algorithms RM-US and SM-US. Therefore, $\texttt{GS}_{bound}$ dominates both RM-US and SM-US.
2) We propose priority assignment policy $P_{search}$ and show that $\texttt{GS}_{search}$ *dominates* $\texttt{GS}_{bound}$ scheduling algorithm. We prove the domination of $\texttt{GS}_{search}$ over $\texttt{GS}_{bound}$ by showing that any task set schedulable using $\texttt{GS}_{bound}$ is also schedulable using $\texttt{GS}_{search}$ and not conversely.
3) We show through simulation experiments that more than 92% of the one million randomly-generated $\texttt{GS}_{search}$ schedulable task sets are *not* SM-US schedulable. This percentage increases up to 99.99% when we increase the number of processors from 4 to 32 in our experiments. Therefore, $\texttt{GS}_{search}$ scheduling *scales* very well in comparison to SM-US with increasing number of processors.

The rest of the paper is organized as follows. First, other works related to multiprocessor scheduling are discussed in Section II. The system model we use in this work is presented in Section III. Useful definitions and some important prior results that we use are presented in Section IV. The schedulability analysis of “special” task sets and the two priority assignment policies are presented in Section V and Section VI, respectively. In Section VII, we prove the domination of $\texttt{GS}_{search}$ over $\texttt{GS}_{bound}$ and domination of $\texttt{GS}_{bound}$ over both RM-US and SM-US scheduling, and present our simulation results. Finally, we conclude the paper in Section VIII.

## II. Related Work

The well-known Rate-Monotonic (RM) static-priority assignment is optimal for uniprocessor [15] and is not optimal for global multiprocessor scheduling due to so called “Dhall’s effect” [11]. Andersson, Jonsson and Baruah proposed a RM hybrid static-priority assignment policy for RM-US$[m/(3m-2)]$ algorithm that has the utilization bound of $m^2/(3m-2)$ on $m$ processors [3]. In RM-US$[m/(3m-2)]$ algorithm, the Dhall’s effect is avoided by assigning the highest priority to the tasks having utilization greater than $m/(3m-2)$ while the rest of the tasks are given the traditional RM priority.

Analysis of static-priority multiprocessor global scheduling is also addressed by Baker [5] based on the minimum amount of interference in an interval that can cause a tasks deadline to be missed. Baker’s analysis is general for any fixed-priority scheduling and arbitrary deadline task systems. Baker [5] showed that, for implicit-deadline task sets the utilization bound of RM scheduling is $\frac{m(1-u_{max})}{2} + u_{min}$, where $u_{max}$ and $u_{min}$ are the maximum and minimum utilization of any task in the task set, respectively. RM scheduling is studied for uniform multiprocessors in [6], and it is shown that the utilization bound is $\frac{m}{3}$ for implicit-deadline tasks on $m$ unit-capacity processors. Using an analysis of the worst-case workload in an interval, similar to that of the Baker’s work in [5], Bertogna *et al.* [7] showed that, the utilization bound for deadline-monotonic scheduling of implicit deadline tasks is $\frac{m(1-u_{max})}{2} + u_{max}$. The work in [7] is further improved in [8], where an iterative algorithm for separately testing the schedulability of each task is proposed.

Based on slack-monotonic hybrid priority assignment scheme, a static priority-assignment policy is proposed by Andersson for SM-US scheduling algorithm [1]. In SM-US scheduling, each task having utilization greater than $2/(3+\sqrt{5})$ is given the highest static-priority and the rest of the tasks are given slack-monotonic priorities. The utilization bound of the SM-US scheduling algorithm is $2m/(3+\sqrt{5}) \approx 0.3819m$ [1]. Recently, static-priority assignment scheme for global scheduling is proposed by Davis and Burns [10] which has been shown (using simulation) to have better performance than many other fixed-priority scheduling algorithms.

In summary, the current state-of-the-art utilization bound for global multiprocessor scheduling is $\frac{m^2}{3m-2}$ for $m \leq 5$ (RM-US scheduling) and $\frac{2m}{3+\sqrt{5}}$ for $m > 5$ (SM-US scheduling). In this paper, we show that our proposed scheduling algorithm $\texttt{GS}_{bound}$ has a higher utilization bound than that of both RM-US and SM-US. Moreover, we show that $\texttt{GS}_{search}$ dominates $\texttt{GS}_{bound}$ which also implies $\texttt{GS}_{search}$ dominates both RM-US and SM-US.

## III. System Model

In this paper, we consider global preemptive scheduling of $n$ independent periodic tasks in set $\Gamma = \{\tau_1, \tau_2, \ldots \tau_n\}$ on $m$ identical, unit-capacity processors. Each task $\tau_i \in \Gamma$ is characterized by a pair ($C_i$, $T_i$), where $C_i$ represents the WCET and $T_i$ is the period of task $\tau_i$. An instance of each task, called *job*, is released at each period $T_i$ and requires at most $C_i$ units of execution time before the next period.

Without loss of generality we assume that tasks are sorted based on *decreasing* priority order, that is, task $\tau_k$ has lower priority than task $\tau_i$ for $i \leq k$. Since the execution of a task $\tau_k$ can be interfered only by the higher-priority tasks, whether task $\tau_k$ meets its deadline or not depends on the tasks in $\{\tau_1, \tau_2, \ldots \tau_k\}$ but are completely unaffected by the tasks in $\{\tau_{k+1}, \tau_{k+2}, \ldots \tau_n\}$. We find it useful to define the task set $\Gamma^k \overset{\text{def}}{=} \{\tau_1, \tau_2, \ldots \tau_k\}$ for $k = 1, 2 \ldots n$. We sometimes, hereafter, also use the notation $\Gamma^k$ to denote the collection of all the jobs of the tasks in $\Gamma^k$.

We define the *utilization* of a task $\tau_i$ as $u_i = C_i/T_i$ and the *total utilization* of the task set $\Gamma^k$ as $U^k = \sum_{i=1}^{k} u_i$ for $k = 1, 2 \ldots n$. Note that, $U^n$is the total utilization of $\Gamma^n$=$\Gamma$.

We define the *maximum utilization* $u^k_{max}$ and the *minimum utilization* $u^k_{min}$ of a periodic task system $\Gamma^k$ to be the largest and smallest utilization of any task in $\Gamma^k$, respectively.

## IV. Useful Definitions and Prior Results

The schedulability analysis presented in this paper is based on the schedulability analysis of a class of task sets called "special" task sets. A task set is said to be "special on $m$ processors" based on *two* particular properties (defined shortly in Definition 2). We prove that a task set that is special on $m$ processors is schedulable using global slack-monotonic scheduling, denoted by $\mathtt{GS}_{\mathtt{SM}}$, on $m$ processors (proved in Theorem 6).

It will be evident later that the schedulability analysis of special task sets is based on the amount of *work* done by $\mathtt{GS}_{\mathtt{SM}}$ within a particular time interval. To formally characterize the amount of work done by any scheduling algorithm $A$ over a time interval of length $t$ on $m$ processors each having speed $s$, we use the following well-known definition of *work* (also used in [3], [6], [13]).

**Definition 1** ($W(A, m, s, \Gamma^k, t)$)**.** *Let the jobs of the task set $\Gamma^k$ are to be executed using any algorithm $A$ on an identical multiprocessor platform where each processor has speed $s$. For time instant $t \geq 0$, let $W(A, m, s, \Gamma^k, t)$ denote the amount of work done by algorithm $A$ on jobs of the task set $\Gamma^k$ over the interval $[0, t)$, while executing on $m$ processors each of speed $s$.*

Scheduling algorithm $A$ on a multiprocessor platform is called *work-conserving algorithm*, if it never idles a processor when there are jobs awaiting execution. Note that, *global static-priority scheduling is work-conserving* by definition. The following Theorem 1 provides a lower bound on the amount of work completed by a work-conserving scheduling algorithm $A$ on an identical multiprocessor platform that has each of the $m$ processors $(2 - \frac{1}{m})$ times faster compared to that of completed by any algorithm $A'$ on a platform with $m$ identical processors. Theorem 1 is based on the work in [17] that exploits **resource augmentation** for on-line scheduling of real-time jobs.

**Theorem 1** (Based on [17])**.** *For the set of all jobs of task systems $\Gamma^k$, any time-instant $t \geq 0$, any work-conserving algorithm $A$, and any algorithm $A'$, it is the case that*

$$W(A, m, s \cdot (2 - \frac{1}{m}), \Gamma^k, t) \geq W(A', m, s, \Gamma^k, t) \qquad (1)$$

Theorem 1 will be useful later to find the lower bound on the amount of work completed by time instant $t$ using global slack-monotonic scheduling algorithm $\mathtt{GS}_{\mathtt{SM}}$.

We will use the following function in Eq. (2) in rest of this paper:

$$F_m(x) \;= \frac{m(1-x)}{2-x} + x \qquad (2)$$

where $m \in \mathbb{Z}^+$ and $0 \leq x \leq \frac{m}{2m-1}$. Two important features of the function in Eq. (2) are given in Lemma 2.

**Lemma 2.** *Consider $a$, $b$, $x$, $c$ and $d$ such that $0 \leq a \leq b \leq x \leq c \leq d \leq \frac{m}{2m-1}$ for any integer $m > 0$. The following two inequalities hold:*

$$min\{F_m(b), F_m(c)\} \leq F_m(x) \qquad (3)$$

$$min\{F_m(a), F_m(d)\} \leq min\{F_m(b), F_m(c)\} \qquad (4)$$

*Proof:* Proof is given in Appendix. ■

## V. "Special" task set and its Schedulability

In this section, first we formally define the two properties of a task system that is "special" on $m$ processors. Then, we present an upper and lower bound on the amount of work within a given interval for $\mathtt{GS}_{\mathtt{SM}}$ scheduling (subsections V-A and V-A). Finally, based on the difference between these two bounds and using the two properties of the special task system, we prove that all the deadlines of the special task system are met using $\mathtt{GS}_{\mathtt{SM}}$ scheduling on $m$ unit-capacity processors (subsection V-C). Then based on the schedulability analysis of special task system of this section, we propose the two slack-monotonic hybrid priority-assignment policies $P_{bound}$ and $P_{search}$ and the corresponding schedulability analysis of algorithms $\mathtt{GS}_{bound}$ and $\mathtt{GS}_{search}$, respectively, in Section VI.

**Definition 2** (Special Task System)**.** *A periodic task system $\Gamma^k$ is special on $m$ processor if it satisfies the following two properties:*

***Property 1:*** $u^k_{max} \leq \frac{m}{2m-1}$

***Property 2:*** $U^k \leq min\{F_m(u^k_{min}), F_m(u^k_{max})\}$

According to Property 1, the maximum utilization of any task in set $\Gamma^k$, that is special on $m$ processors, is not greater than $\frac{m}{2m-1}$. According to Property 2, the total utilization of the special task system $\Gamma^k$ is not greater than the minimum of $F_m(u^k_{min})$ and $F_m(u^k_{max})$.

### A. Upper Bound on Work of $\mathtt{GS}_{\mathtt{SM}}$

Over the interval $[0, t)$, exactly $\lfloor \frac{t}{T_i} \rfloor$ complete jobs of task $\tau_i$ are scheduled by $\mathtt{GS}_{\mathtt{SM}}$ and the $(\lfloor \frac{t}{T_i} \rfloor + 1)^{th}$ job of $\tau_i$ may be scheduled for at most $min(t - \lfloor \frac{t}{T_i} \rfloor T_i, C_i)$ time units. Thus, the maximum amount of work to be completed by the jobs of the tasks $\{\tau_1, \ldots \tau_{j-1}\}$ within $[0, t)$ is at most $\sum_{i=1}^{j-1} \lfloor \frac{t}{T_i} \rfloor C_i \; + min(t - \lfloor \frac{t}{T_i} \rfloor T_i \;, C_i)$ and this quantity is upper bounded by the right hand side of Eq. (5) given in Lemma 3 (proved in [1], [9]).

**Lemma 3** (From [1], [9])**.** *The maximum amount of work $\sum_{i=1}^{j-1} \lfloor \frac{t}{T_i} \rfloor C_i \; + min(t - \lfloor \frac{t}{T_i} \rfloor T_i \;, C_i)$ that need to be completed by tasks $\{\tau_1, \tau_2 \ldots \tau_{j-1}\}$ over an interval $[0, t)$ is upper bounded by the following inequality in Eq.* (5)

$$\sum_{i=1}^{j-1} [\ \lfloor \frac{t}{T_i} \rfloor C_i\ + min(t - \lfloor \frac{t}{T_i} \rfloor T_i\ , C_i)\ ]$$

$$\leq \sum_{i=1}^{j-1}\ \ [\ C_i + (t - C_i)u_i] \qquad (5)$$

We will use Eq. (5) as the upper bound of the amount of work that need to be completed by the higher priority jobs of task $\tau_j$ within a particular time interval $[0, t]$.

## *B. Lower Bound on Work of* GS$_{SM}$

In this subsection, first we show in Lemma 4 that there exists an algorithm, called OPT, that can feasibly schedule special task set $\Gamma^k$ on $m$ processors each of speed $\frac{m}{2m-1}$. Then, we will show that the amount of work completed by GS$_{SM}$ on $m$ unit-capacity processors within an interval is no less than that of completed by OPT on $m$ processors each of speed $\frac{m}{2m-1}$ (i.e., we derive a lower bound on the amount of work actually completed by GS$_{SM}$).

**Lemma 4.** *Consider the task set $\Gamma^k$ that is special on $m$ processors. Task set $\Gamma^k$ is feasible on $m$ processors each with speed $\frac{m}{2m-1}$.*

*Proof:* By Property 1 of special task system we have $u^k_{max} \leq \frac{m}{2m-1}$. Therefore, we have $u_i \leq \frac{m}{2m-1}$ since $u_i \leq u^k_{max}$ for all $i = 1, 2, \ldots k$. To prove this lemma, we need to show that the following inequality holds

$$U^k \leq \frac{m^2}{2m-1} \qquad (6)$$

This is because, if $u_i \leq \frac{m}{2m-1}$ and $U^k \leq \frac{m^2}{2m-1}$, then the task system $\Gamma^k$ can be scheduled to meet all the deadlines on $m$ processors each of speed $\frac{m}{2m-1}$ using the algorithm, called OPT, as follows

> The processor sharing schedule, called OPT, assigns a fraction $u_i$ of a processor to $\tau_i$ at each time-instant, and thus ensures that each instance of task $\tau_i$ completes $C_i$ units of execution within its deadline $T_i$.

The algorithm OPT is also used in the schedulability analysis of RM-US in [3] where the above inequality in Eq. (6) was straightforward to prove for so called “light task system”. However, showing that the inequality in Eq. (6) also holds for “special task system” is not as straightforward as for RM-US. To show that inequality in Eq. (6) holds we, in fact, show that the inequality in Eq. (7) holds.

$$min\{F_m(u^k_{min}), F_m(u^k_{max})\}\ \leq\ \frac{m^2}{2m-1} \qquad (7)$$

The proof that Eq. (7) holds is given in Lemma 10 in the Appendix. Since $\Gamma^k$ is special on $m$ processors, using Property 2 of special task system we have

$$U^k \leq min\{F_m(u^k_{min}), F_m(u^k_{max})\} \qquad (8)$$

From Eq. (7) and Eq. (8), the inequality in Eq. (6) follows immediately. ■

The outcome of Lemma 4 is that there exists a scheduling algorithm, called OPT, through the use of which a task system $\Gamma^k$ that is special on $m$ unit-capacity processors is feasible on $m$ identical processors each of speed $\frac{m}{2m-1}$. Now, since $\frac{m}{2m-1} \times (2 - \frac{1}{m}) = 1$, using Theorem 1 together with the existence of an scheduling algorithm OPT plus the fact that GS$_{SM}$ is work-conserving, we get

$$W(\text{GS}_{\text{SM}}, m, 1, \Gamma^k, t) \geq W(\text{OPT}, m, \frac{m}{2m-1}, \Gamma^k, t) \qquad (9)$$

for all $t \geq 0$.

Note that according to algorithm OPT, $W(\text{OPT}, m, \frac{m}{2m-1}, \Gamma^k, t) = t \cdot \sum_{i=1}^{k} u_i = t \cdot U^k$ for any time instant $t \geq 0$. Thus, the lower bound on the amount of work actually completed by GS$_{SM}$ within $[0, t)$ for executing the jobs of the task set $\Gamma^k$ is given in Eq. (10) as follows.

$$W(\text{GS}_{\text{SM}}, m, 1, \Gamma^k\ , t) \geq t \cdot \sum_{i=1}^{k} u_i = t \cdot U^k \qquad (10)$$

We will use Eq. (10) as the lower bound of the actual amount of work completed by GS$_{SM}$ scheduling within a particular time interval $[0, t]$.

## *C. Slack-Monotonic Schedulability Analysis*

In this subsection, we prove that a task system $\Gamma^k$ that is special on $m$ processors is feasible using GS$_{SM}$ on $m$ unit-capacity processors. First, we prove in Lemma 5 that all the jobs of the lowest priority task $\tau_j$ of task set $\Gamma^j$, which is special on $m$ processors, complete by their deadlines using GS$_{SM}$ scheduling. Then, it follows using induction on $j = 1, 2, \ldots k$ that the special task system $\Gamma^k$ is feasible on $m$ processors using GS$_{SM}$.

**Lemma 5.** *Consider task set $\Gamma^j$ that is special on $m$ processors. All the jobs of task $\tau_j$ meet their deadlines when $\Gamma^j$ is scheduled using GS$_{SM}$ on $m$ unit-capacity processors.*

*Proof:* We will prove this Lemma using induction. Lets assume that all the $(l-1)$ jobs of $\tau_j$ have met their deadlines using GS$_{SM}$ scheduling algorithm. We will prove that the $l^{th}$ job of $\tau_j$ also meets the deadline. Using induction on $l \geq 1$, the correctness of Lemma 5 then immediately follows.

Since each job of task $\tau_j$ is released at each period $T_j$, the $l^{th}$ job arrives at time $(l-1)T_j$ and requires $C_j$ units of execution time before its deadline $lT_j$. Remember from Lemma 4 that task set $\Gamma^j$ is schedulable using algorithm OPT on $m$ processors each having speed $\frac{m}{2m-1}$. Thus, from Eq. (10) we have,

$$W(\text{GS}_{\text{SM}}, m, 1, \Gamma^j, (l-1)T_j) \geq (l-1)T_j \cdot \sum_{i=1}^{j} u_i$$

$$= (l-1)T_j \cdot \sum_{i=1}^{j-1} u_i + (l-1)T_j u_j \qquad (11)$$

According to Eq. (11), the minimum amount of work completed by GS$_{SM}$ before the $l^{th}$ job of $\tau_j$ arrives at time $(l-1)T_j$ is $(l-1)T_j \cdot \sum_{i=1}^{j-1} u_i + (l-1)T_j u_j$. Note that, prior to time instant $(l-1)T_j$, the amount of work generated for task $\tau_j$ is exactly $(l-1)T_j u_j$. Since we assume that all the $(l-1)$ jobs of task $\tau_j$ have met their deadlines (inductive hypothesis), the total work executed by $\mathsf{GS}_{\mathsf{SM}}$ for the higher-priority tasks $\tau_1, \tau_2, \ldots \tau_{j-1}$ is at least $(l-1)T_j \sum_{i=1}^{j-1} u_i$ prior to the time instant $(l-1)T_j$.

Lemma 3 ensures that the maximum amount of work that can be completed by all the higher-priority tasks $\tau_1, \tau_2, \ldots \tau_{j-1}$ over the interval $[0, lT_j)$ is bounded from above by $\sum_{i=1}^{j-1}[C_i + (lT_j - C_i)u_i]$. In the previous paragraph, we saw that at least $(l-1)T_j \sum_{i=1}^{j-1} u_i$ of this work is completed prior to time instant $(l-1)T_j$. Therefore, at most

$$\sum_{i=1}^{j-1}[(C_i + (lT_j - C_i)u_i)] - (l-1)T_j \sum_{i=1}^{j-1} u_i = \sum_{i=1}^{j-1}(C_i + (T_j - C_i)u_i)$$

amount of work remains to be executed after time instant $(l-1)T_j$ for all the higher priority tasks $\tau_1, \tau_2, \ldots \tau_{j-1}$.

The amount of processors capacity left unused by tasks $\tau_1, \tau_2, \ldots \tau_{j-1}$ during the interval $[(l-1)T_j, lT_j)$ on the $m$ multiprocessor platform is therefore at least

$$m \cdot T_j - \sum_{i=1}^{j-1}(C_i + (T_j - C_i)u_i) \qquad (12)$$

Not all of this capacity is available to the $l^{th}$ job of $\tau_j$ if several processors are available at the same time. In the worst case (i.e., all the $m$ processors are available at the same time) at least $\frac{1}{m}$ of this available capacity can be used by $\tau_j$. Consequently, the amount of processing capacity available to the $l^{th}$ job of $\tau_j$ during the interval $[(l-1)T_j, lT_j)$ on the multiprocessor platform is at least

$$\frac{1}{m}\left[m \cdot T_j - \sum_{i=1}^{j-1}(C_i + (T_j - C_i)u_i)\right]$$

To guarantee that the $l^{th}$ job of $\tau_j$ meets its deadline, we need this capacity to be at least as large as the execution time of $\tau_j$; that is, we must have,

$$C_j \le \frac{1}{m}\left[m \cdot T_j - \sum_{i=1}^{j-1}(C_i + (T_j - C_i)u_i)\right] \qquad (13)$$

In the remaining part of this proof, we show that Eq. (13) holds; which guarantees that the $l^{th}$ job of $\tau_j$ meets its deadline. Since task set $\Gamma^j$ is special on $m$ processors, according to Property 2 of special task set we have

$$U^j \le \mathit{min}\{F_m(u^j_{min}), F_m(u^j_{max})\} \qquad (14)$$

For task $\tau_j \in \Gamma^j$, we have $u^j_{min} \le u_j \le u^j_{max}$. Thus, according to Property 1 of special task system $\Gamma^j$, we have $0 \le u^j_{min} \le u_j \le u^j_{max} \le \frac{m}{2m-1}$. And using Eq. (3) of Lemma 2, we have

$$\mathit{min}\{F_m(u^j_{min}), F_m(u^j_{max})\} \le F_m(u_j) \qquad (15)$$

From Eq. (14) and Eq. (15), we have $U^j \le F_m(u_j)$ which is equivalent to

$$\equiv \sum_{i=1}^{j} u_i \le \frac{m(1-u_j)}{2-u_j} + u_j \qquad \textit{[from Eq. (2)]}$$

$$\equiv \sum_{i=1}^{j-1} u_i(2-u_j) \le m(1-u_j)$$

$$\equiv u_j \le 1 - \frac{1}{m}\sum_{i=1}^{j-1} u_i(2-u_j)$$

$$\equiv u_j \le \frac{1}{m}\left[m - \sum_{i=1}^{j-1}\left[u_i + u_i(1-u_j)\right]\right]$$

$$\equiv \frac{C_j}{T_j} \le \frac{1}{m}\left[m - \sum_{i=1}^{j-1}\left[\frac{C_i}{T_i} + \frac{C_i}{T_i}\left(\frac{T_j - C_j}{T_j}\right)\right]\right]$$

$\Rightarrow$ (According to slack-monotonic priority assignment, we have $T_i - C_i \le T_j - C_j$)

$$\equiv \frac{C_j}{T_j} \le \frac{1}{m}\left[m - \sum_{i=1}^{j-1}\left[\frac{C_i}{T_i} + \frac{C_i}{T_i}\left(\frac{T_i - C_i}{T_j}\right)\right]\right]$$

$$\equiv C_j \le \frac{1}{m}\left[m \cdot T_j - \sum_{i=1}^{j-1}\left[\frac{C_i T_j}{T_i} + C_i - \frac{C_i^2}{T_i}\right]\right]$$

$$\equiv C_j \le \frac{1}{m}\left[m \cdot T_j - \sum_{i=1}^{j-1}\left[C_i + (T_j - C_i)u_i\right]\right] \equiv \text{Eq. (13)}$$

Since the inequality in Eq. (13) is true, we can conclude that the $l^{th}$ job of task $\tau_j$ meets its deadline using $\mathsf{GS}_{\mathsf{SM}}$. ∎

Based on Lemma 5 we prove in Theorem 6 that the task set $\Gamma^k$ that is special on $m$ processors is feasible using $\mathsf{GS}_{\mathsf{SM}}$.

**Theorem 6.** *Task system $\Gamma^k$ that is special on $m$ processors is feasible using $\mathsf{GS}_{\mathsf{SM}}$ on $m$ processors.*

*Proof:* Using induction on $j$ and applying Lemma 5 for task set $\Gamma^j$ for $j = 1, 2, \ldots k$, it is easy to see that the special task system $\Gamma^k$ is feasible on $m$ processors using $\mathsf{GS}_{\mathsf{SM}}$. ∎

Based on the $\mathsf{GS}_{\mathsf{SM}}$ feasibility of "special" task systems, we propose two slack-monotonic hybrid priority-assignment policies $P_{bound}$ and $P_{search}$ for an arbitrary task set $\Gamma$.

## VI. Hybrid Priority Assignment

In this section, we propose the two hybrid priority assignment policies $P_{bound}$ and $P_{search}$. Common for both policies is that the priorities to the tasks are assigned based on some threshold utilization $u_{ts}$ such that all the tasks having utilization not greater than $u_{ts}$ are given the slack-monotonic priorities and each task having utilization greater than $u_{ts}$ is given the highest priority. Using such hybrid policy, the task set $\Gamma$ is visualized as the union of two sets $\Gamma = \Gamma_L \cup \Gamma_H$ such that the tasks in set $\Gamma_L$ have the slack-monotonic priorities and each task in set $\Gamma_H$ has the highest priority. It will be evident shortly that the value of $u_{ts}$ for policy $P_{bound}$ only depends on the number of processors. However, the value of $u_{ts}$ for policy $P_{search}$ depends not only on the number of processors but also on the utilization information of the individual task of a given task set.

Before the thresholds for the two priority assignment policies are presented, we present two general conditions, denoted as **C1** and **C2**, in Lemma 7 that can imply the feasibility of a task set based on priority assignment policy $P_{bound}$ ($P_{search}$). The proof technique in Lemma 7 is based on the notion of *predictable scheduling algorithm* that is proposed by Ha and Liu in [14] and used in [1] as follows.

**Predictability (from [14]):** A job is characterized by its arrival time, its deadline, its minimum execution time and its maximum execution time. The execution time of a job is unknown but it is no less than and greater than its minimum and maximum execution time, respectively. A scheduling algorithm $A$ is *predictable* if for every set $J$ of jobs, the following fact

> scheduling all jobs in $J$ by $A$ with execution times equal to their maximum execution times causes all the deadlines to be met

implies that

> scheduling all jobs in $J$ by $A$ with execution times being at least their minimum execution times and at most their maximum execution times causes all the deadlines to be met.

This notion of predictable scheduling algorithm implies that we only need to analyze the schedulability of the jobs considering the WCET of the jobs. Since a periodic task set generates a set of jobs, the notion of predictability can be extended in a straightforward manner to algorithms for scheduling periodic task systems. Ha and Liu's work also implies that global static-priority scheduling of periodic tasks on multiprocessors is predictable [3], [1].

**Lemma 7.** *Let $u_{ts}$ be the threshold utilization that is used to determine the sets $\Gamma_L$ and $\Gamma_H$ such that $\Gamma = \Gamma_L \cup \Gamma_H$ assuming the priority assignment policy $P_{bound}$ ($P_{search}$). The task set $\Gamma$ is schedulable using $\mathtt{GS}_{bound}$ ($\mathtt{GS}_{search}$) if the following two conditions **C1** and **C2** are satisfied*

(**C1**) $|\Gamma_H| < m$

(**C2**) $\Gamma_L$ *is special on* $(m - |\Gamma_H|)$ *processors*

*Proof:* We will show that if **C1** and **C2** are true for priority assignment policy $P_{bound}$ ($P_{search}$) that uses $u_{ts}$ as the threshold utilization, then the task set $\Gamma$ is schedulable using the $\mathtt{GS}_{bound}$ ($\mathtt{GS}_{search}$) scheduling. Consider the following task set $\Gamma'_H$ such that

$$\Gamma'_H = \{\tau'_i \mid \tau_i \in \ \Gamma_H \text{ and } \ T'_i = C'_i = T_i\}$$

For each task $\tau_i \in \Gamma_H$, there is a corresponding task $\tau'_i$ in set $\Gamma'_H$ such that the period and the WCET of $\tau'_i$ are equal to $T_i$. Therefore, each task $\tau'_i \in \Gamma'_H$ has utilization 1. Note that, $|\Gamma'_H| = |\Gamma_H|$ and we let $k = |\Gamma'_H| = |\Gamma_H|$.

Now consider the task set $\Gamma' = \Gamma_L \cup \Gamma'_H$ that is to be scheduled on $m$ processors using $\mathtt{GS}_{bound}$ ($\mathtt{GS}_{search}$). According to policy $P_{bound}$ ($P_{search}$) that uses the threshold utilization $u_{ts}$, each of the tasks in $\Gamma'_H$ is given the highest priority and the tasks in $\Gamma_L$ are given the slack-monotonic priorities.

When scheduling the task set $\Gamma'$ using $\mathtt{GS}_{bound}$ ($\mathtt{GS}_{search}$), then $k = |\Gamma'_H|$ processors are devoted to tasks in set $\Gamma'_H$ since these are the highest priority tasks each with utilization 1. All these tasks in $\Gamma'_H$ are schedulable on $k$ processors (one task is executed in one dedicated processor) since $|\Gamma'_H| = |\Gamma_H| = k < m$ according to **C1**.

According to **C2**, the tasks in set $\Gamma_L$ are special on $(m-|\Gamma_H|) = (m-k)$ processors. Remember that according to Theorem 6, task set $\Gamma_L$ that is special on $(m-k)$ processors is schedulable using global slack-monotonic scheduling $\mathtt{GS}_{\mathtt{SM}}$ on $(m-k)$ processors. Consequently, the task set $\Gamma'$ is schedulable on $m$ processors using $\mathtt{GS}_{bound}$ ($\mathtt{GS}_{search}$) scheduling.

However, since global static-priority algorithm $\mathtt{GS}_{bound}$ ($\mathtt{GS}_{search}$) is predictable, jobs of the tasks in set $\Gamma_H$, in fact, complete earlier than the jobs of the tasks in set $\Gamma'_H$. In other words, no jobs in task system $\Gamma$ finishes later than the corresponding job in $\Gamma'$ for predictable scheduling algorithm $\mathtt{GS}_{bound}$ ($\mathtt{GS}_{search}$). Therefore, all the deadlines of the tasks in $\Gamma$ are met using $\mathtt{GS}_{bound}$ ($\mathtt{GS}_{search}$) scheduling where the tasks are given the priorities based on policy $P_{bound}$ ($P_{search}$) whenever the conditions **C1** and **C2** are satisfied. ■

Based on these two general conditions (**C1** and **C2**) of Lemma 7, the schedulability analysis of $\mathtt{GS}_{bound}$ and $\mathtt{GS}_{search}$, assuming the priority assignment policies $P_{bound}$ and $P_{search}$, are presented in subsection VI-A and subsection VI-B, respectively.

## A. Priority Assignment Policy $P_{bound}$

In this section, we present the threshold utilization used for slack-monotonic hybrid priority assignment policy $P_{bound}$ and derive the corresponding utilization bound for $\mathtt{GS}_{bound}$ scheduling. The value of $u_{ts}$ is defined based on the solution of the equation $F_m(u_{ts}) = m \cdot u_{ts}$ where $m$ is some constant, $m > 1$. One of the solutions of $F_m(u_{ts}) = m \cdot u_{ts}$ is $u_{ts} = \frac{3m-2-\sqrt{5m^2-8m+4}}{2m-2}$ for $m > 1$. The value of $u_{ts}$ for policy $P_{bound}$ is $u_{ts} = B(m)$ for $m > 0$ where $B(m)$ is defined as follows:

$$B(m) = \begin{cases} 1 & \text{if } m = 1 \\ \frac{3m-2-\sqrt{5m^2-8m+4}}{2m-2} & \text{if } m > 1 \end{cases} \tag{16}$$

The two following inequalities in Eq. (17) and Eq. (18) hold for $B(m)$ and $B(m')$ where $m \geq m' \geq 1$

$$B(m) \leq \frac{m}{2m-1} \leq \frac{m'}{2m'-1} \tag{17}$$

$$B(m) \leq B(m') \tag{18}$$

The proof that Eq. (17) and Eq. (18) hold are given in Lemma 11 and Lemma 12 in the Appendix. Now the schedulability condition in terms of utilization bound of $\mathtt{GS}_{bound}$ scheduling is given in Theorem 8.

**Theorem 8.** *A task set $\Gamma$ is schedulable using $\texttt{GS}_{bound}$ if $U^n \leq m \cdot min\{1/2, B(m)\}$ for $m \geq 2$.*

*Proof:* Given the task set $\Gamma$ and the number of processors $m$, we can determine the two subsets $\Gamma_L$ and $\Gamma_H$ such that $\Gamma = \Gamma_L \cup \Gamma_H$ based on the threshold utilization $u_{ts} = B(m)$. Note that, based on policy $P_{bound}$, the tasks in set $\Gamma_L$ and $\Gamma_H$ are given the slack-monotonic and the highest static priorities, respectively. We will show that if the total utilization $U^n \leq m \cdot min\{1/2, B(m)\}$, then the two general conditions **C1** and **C2** of Lemma 7 hold; which guarantees the schedulability of $\Gamma$ using $\texttt{GS}_{bound}$.

**(C1 holds)** It is easy to see that $B(m) \geq min\{1/2, B(m)\}$. Then it follows that each task in $\Gamma_H$ has utilization greater than $min\{1/2, B(m)\}$ since each task in $\Gamma_H$ has utilization greater than $u_{ts} = B(m)$ using policy $P_{bound}$. If the total utilization (i.e., $U^n$) of the task set $\Gamma$ is not greater than $m \cdot min\{1/2, B(m)\}$, then the number of tasks that are given the highest priority is less than $m$. In other words, we have $|\Gamma_H| < m$, and this implies that condition **C1** of Lemma 7 holds.

**(C2 holds)** To show that **C2** of Lemma 7 holds, we have to show that $\Gamma_L$ is special on $m'$ processors where $m' = (m - |\Gamma_H|)$. Let $\texttt{UL}$ be the total utilization of all the tasks in $\Gamma_L$. Also let $u_{maxL}$ and $u_{minL}$ be the maximum utilization and minimum utilization of any task in set $\Gamma_L$, respectively. To show that $\Gamma_L$ is special on $m'$ processors, we show that Property 1 and Property 2 (given in Definition 2) of special task set are satisfied. In other words, we have to show that the following two inequalities hold.

**Property 1** $\quad u_{maxL} \leq \dfrac{m'}{2m'-1}$

**Property 2** $\quad \texttt{UL} \leq min\{F_{m'}(u_{minL}), F_{m'}(u_{maxL})\}$

**(Property 1 holds for $\Gamma_L$)** Using policy $P_{bound}$, no task in $\Gamma_L$ has utilization greater than the threshold utilization $u_{ts} = B(m)$. So, we have $u_{maxL} \leq B(m)$. Moreover, from Eq. (17), we have $B(m) \leq \frac{m'}{2m'-1}$. Consequently, $u_{maxL} \leq \frac{m'}{2m'-1}$ and thus Property 1 is satisfied for $\Gamma_L$.

**(Property 2 holds for $\Gamma_L$)** The total utilization of the tasks in $\Gamma_H$ is greater than $(|\Gamma_H| \cdot min\{1/2, B(m)\})$ because each task in $\Gamma_H$ has utilization greater than $u_{ts} = B(m)$ and $B(m) \geq min\{1/2, B(m)\}$. Since the total utilization (i.e., $U^n$) of the task set $\Gamma$ is not greater than $m \cdot min\{1/2, B(m)\}$, the total utilization of the tasks in $\Gamma_L$ is at most $m' \cdot min\{1/2, B(m)\}$ where $m' = (m - |\Gamma_H|)$. Therefore, Eq. (19) holds.

$$\texttt{UL} \leq m' \cdot min\{1/2, B(m)\} \tag{19}$$

Based on the threshold utilization $u_{th} = B(m)$ of policy $P_{bound}$, we have $u_{maxL} \leq B(m)$. Moreover, from Eq. (18), we have $B(m) \leq B(m')$. Thus, $u_{maxL} \leq B(m')$. Furthermore, from Eq. (17), we have $B(m') \leq \frac{m'}{2m'-1}$ (by replacing $m$ by $m'$ in the left-hand side inequality in Eq. (17)). Therefore, $u_{maxL} \leq B(m') \leq \frac{m'}{2m'-1}$. Because $0 \leq u_{minL} \leq u_{maxL}$, the following inequality in Eq. (20) holds.

$$0 \leq u_{minL} \leq u_{maxL} \leq B(m') \leq \frac{m'}{2m'-1} \tag{20}$$

Based on Eq. (20) and from Eq. (4) of Lemma 2 we have

$$\begin{aligned} &min\{F_{m'}(0)\ ,\ F_{m'}(B(m'))\} \\ &\quad \leq min\{F_{m'}(u_{minL}), F_{m'}(u_{maxL})\} \end{aligned} \tag{21}$$

From the function definition given in Eq. (2) we have

$$F_{m'}(0) = \frac{m'(1-0)}{2-0} + 0 = m'/2 \tag{22}$$

From Eq. (16), for $m' = 1$ and setting $x = B(m')$ we have $F_{m'}(B(m'))= 1 = m'$. And for $m' > 1$ we have $F_{m'}(B(m')) = m' \cdot B(m')$ because one solution for $x$ of $F_{m'}(x) = m'x$ is $x = B(m')$. Thus, for any $m'$

$$F_{m'}(B(m')) \geq m' \cdot min\{1, B(m')\} \tag{23}$$

It follows from Eq. (22) and Eq. (23) that

$$min\{F_{m'}(0), F_{m'}(B(m'))\} \geq m' \cdot min\{1/2, B(m')\} \tag{24}$$

Now from Eq. (24) and using the fact that $B(m) \leq B(m')$ from Eq. (18), we have

$$m' \cdot min\{1/2,\ B(m)\} \leq min\{F_{m'}(0), F_{m'}(B(m'))\} \tag{25}$$

Thus it now follows from Eq. (19) and Eq. (25) that

$$\texttt{UL} \leq min\{F_{m'}(0), F_{m'}(B(m'))\ \} \tag{26}$$

Finally, from Eq. (21) and Eq. (26), we have

$$\texttt{UL} \leq min\{F_{m'}(u_{minL}), F_{m'}(u_{maxL})\ \} \tag{27}$$

Therefore, Property 2 is satisfied for task set $\Gamma_L$. Consequently, $\Gamma_L$ is special on $m'$ processors (i.e., **C2** holds). ■

## *B. Priority Assignment Policy $P_{search}$*

Inspired by the priority assignment scheme for $\texttt{EDF}^{(k)}$ scheduling in [13], the hybrid priority assignment policy $P_{search}$ is defined as follows.

1) *Each of the $k$ highest utilization tasks is given the highest priority, and*
2) *the remaining $(n-k)$ lowest utilization tasks are given the slack-monotonic priorities*

*for some $k$ such that $0 \leq k < m$.*

The schedulability condition of $\texttt{GS}_{search}$ scheduling assuming policy $P_{search}$ is given in Theorem 9.

**Theorem 9.** *A task set $\Gamma$ is schedulable using $\texttt{GS}_{search}$ if the set of $(n-k)$ lowest utilization tasks of $\Gamma$ is special on $(m-k)$ processors for some $k$ and $k < m$.*

*Proof:* Using policy $P_{search}$ the $(n-k)^{th}$ highest utilization of the task in set $\Gamma$ is used as the threshold utilization $u_{ts}$, for some $k$, $0 \leq k < m$. This threshold utilization decides the tasks in set $\Gamma_L$ and $\Gamma_H$ that are given the slack-monotonic and the highest priorities such that $\Gamma = \Gamma_L \cup \Gamma_H$. Note that, using policy $P_{search}$, the number of tasks having the highest priority is $|\Gamma_H| = k$ for some $k$ and $k < m$. Consequently, condition **C1** of Lemma 7 is satisfied for policy $P_{search}$. According to Lemma 7, to guarantee the feasibility of task set $\Gamma$ using $\texttt{GS}_{search}$ scheduling, the value of $k$ has to be chosen such that the condition **C2** of Lemma 7 holds as well. In other words, task set $\Gamma$ is feasible using $\texttt{GS}_{search}$ scheduling whenever $\Gamma_L$ is special on $(m-k)$ processors such that $\Gamma_L$ contains all the $(n-k)$ lowest utilization tasks. ■

Deriving a $k$, if one exists, that satisfies Theorem 9 is straightforward. One such example algorithm, called $P_{search}(\Gamma)$, that finds (if exists) the value of $k$ is presented in Figure 1. The algorithm $P_{search}(\Gamma)$ returns **True** if it can find some $k$ such that the set of $(n-k)$ lowest utilization tasks of $\Gamma$ is special on $(m-k)$ processors such that $k < m$, otherwise, it returns **False**.

**Algorithm** $P_{search}(\Gamma)$

```
1.  $\Gamma_H = \emptyset$
2.  $\Gamma_L = \Gamma$
3.  For $k = 0$ to $(m-1)$
4.    If $\Gamma_L$ is special on $(m-k)$ processors Then
5.       Print "All tasks in $\Gamma_L$ are assigned SM priority"
6.       Print "All tasks in $\Gamma_H$ are assigned the highest priority"
7.       Return True
8.    End If
9.    Find $\tau_{ts}$ such that $u_{ts}$ is the largest utilization in set $\Gamma_L$
10.   $\Gamma_H = \Gamma_H \cup \{\tau_{ts}\}$
11.   $\Gamma_L = \Gamma - \Gamma_H$
12. End For
13. Print "Priority Assignment Fails"
14. Return False
```

Figure 1. $\wp$ Priority Assignment Algorithm

In line 1–2, the algorithm $P_{search}(\Gamma)$ in Figure 1 initializes local variables $\Gamma_L$ and $\Gamma_H$ as $\Gamma_L = \Gamma$ and $\Gamma_H = \emptyset$ to consider first whether all the tasks in $\Gamma$ are special on $m$ processors (this condition is checked during the first iteration of the **For** loop in line 3–12).

The **For** loop in line 3–12 iterates at most $m$ times for the iterative variable $k$ that iterates form 0 to $(m-1)$. In each iteration of the **For** loop, we check whether the $(n-k)$ lowest utilization tasks in set $\Gamma_L$ are special on $(m-k)$ processors. Note that in order to determine whether $\Gamma_L$ is special on $(m-k)$ processors, we need to verify that both Property 1 and Property 2 (Definition 2) of special task system are satisfied. If the task set $\Gamma_L$ is special on $(m-k)$ processors (condition at line 4 is true), then slack-monotonic priorities are assigned to the tasks in $\Gamma_L$ (line 5), each of the tasks in $\Gamma_H$ is assigned the highest static-priority (line 6) and the algorithm returns **True** (line 7).

During a particular iteration of the **For** loop, if the task set $\Gamma_L$ is not special on $(m-k)$ processors (condition at line 4 is false), then the highest utilization task $\tau_{ts} \in \Gamma_L$ is extracted from $\Gamma_L$ (line 9) and is included in set $\Gamma_H$ (line 10). Note that during the $k^{th}$ iteration of the **For** loop, the largest utilization of the tasks in $\Gamma_L$ is the $(n-k)^{th}$ largest utilization of the tasks in $\Gamma$. It is easy to see that, during each iteration of the **For** loop, total $k$ largest utilization tasks are in set $\Gamma_H$ and the rest $(n-k)$ lowest utilization tasks are in set $\Gamma_L$. If the task set $\Gamma_L$ is not special on $(m-k)$ processors for some $k$ such that $0 \leq k < m$, then $P_{search}$ fails to assign the static-priorities to the tasks in $\Gamma$ (line 13) and the algorithm returns **False** (line 14).

## VII. Performance Evaluation

In this section, first we show that $\texttt{GS}_{bound}$ dominates both the best state-of-the-art scheduling algorithms RM-US and SM-US (note that, if $m \leq 5$, RM-US is the best one, otherwise, SM-US is the best one). To show the dominance of $\texttt{GS}_{bound}$ over RM-US and SM-US, we show that the utilization bound of $\texttt{GS}_{bound}$ is higher than that of both RM-US and SM-US. Second, we show that $\texttt{GS}_{search}$ dominates $\texttt{GS}_{bound}$. To show the dominance of $\texttt{GS}_{search}$ over $\texttt{GS}_{bound}$, we show that any task set schedulable using $\texttt{GS}_{bound}$ is also schedulable using $\texttt{GS}_{search}$ but the converse is not true.

**Dominance of $\texttt{GS}_{bound}$ over RM-US and SM-US:** The utilization bound of $\texttt{GS}_{bound}$ is $m \cdot min\{1/2, B(m)\}$ where $B(m) = \frac{3m-2-\sqrt{5m^2-8m+4}}{2m-2}$ for $m \geq 2$ (Theorem 8). The utilization bound of RM-US is $\frac{m^2}{3m-2}$ [1]. It is easy to see that, $m \cdot min\{1/2, B(m)\} > \frac{m^2}{3m-2}$ for any $m > 2$ (they are equal only for $m = 2$). The utilization bound of SM-US is $\frac{2m}{3+\sqrt{5}}$ [1]. It is also easy to see that, $m \cdot min\{1/2, B(m)\} > \frac{2m}{3+\sqrt{5}}$ for any $m \geq 2$. Consequently, the utilization bound of $\texttt{GS}_{bound}$ dominates that of both RM-US and SM-US.

**Dominance of $\texttt{GS}_{search}$ over $\texttt{GS}_{bound}$:** We will show that any task set $\Gamma$ that is schedulable using $\texttt{GS}_{bound}$ is also schedulable using $\texttt{GS}_{search}$. Assume a contradiction where task set $\Gamma$ is not schedulable using $\texttt{GS}_{search}$ but schedulable using $\texttt{GS}_{bound}$. If $\Gamma$ is not schedulable using $\texttt{GS}_{search}$, then there exist no $k$ such that $k < m$ and the set of $(n-k)$ lowest utilization tasks is special on $(m-k)$ processors (contrapositive of Theorem 9).

When $\Gamma$ is schedulable using $\texttt{GS}_{bound}$ scheduling, the proof of the schedulability condition of $\texttt{GS}_{bound}$ in Theorem 8 guarantees that there exists a task set $\Gamma_L$ that is special on $(m - |\Gamma_H|)$ processors and $|\Gamma_H| < m$. So, there exists some $k$ such that $k < m$ and the set of $(n-k)$ lowest utilization tasks is special on $(m-k)$ processors (contradiction!). Therefore, any task set schedulable using $\texttt{GS}_{bound}$ is also schedulable using $\texttt{GS}_{search}$.

We will now show, using an example, that the converse is not true; that is, there is a task set that is schedulable using $\texttt{GS}_{search}$ but not schedulable using $\texttt{GS}_{bound}$.

**Example:** Consider $n = 11$ tasks in set $\Gamma = \{\tau_1, \ldots \tau_{11}\}$ such that $u_1 = \ldots = u_{10} = 0.40$ and $u_{11} = 0.15$. Thus, the total utilization of $\Gamma$ is $U^n = 4.15$. The task set $\Gamma$ is to be scheduled on $m = 10$ processors.

The entire task set $\Gamma$ is special on $m = 10$ processors (i.e., Property 1 and Property 2 of special task system are satisfied) and hence schedulable using $\texttt{GS}_{search}$. Notice that Property 1 of special task system is satisfied for $\Gamma$ because $u^n_{max} = 0.4 < m/(2m-1)$ for $m = 10$. And Property 2 of special task system is satisfied for $\Gamma$ as well. This is because for $m = 10$, $u^n_{max}$=0.40 and $u^n_{min}$=0.15, we have $F_m(u^n_{min}) \approx 4.745$ and $F_m(u^n_{max})$ = 4.150. Consequently, $min\{F_m(u^n_{min}), F_m(u^n_{max})\} = 4.150$ which implies $U^n \leq min\{F_m(u^n_{min}), F_m(u^n_{max})\}$ (i.e., Property 2). Since Property 1 and Property 2 are satisfied, $\Gamma$ is special on $m = 10$ processors. Thus, according to Theorem 9, the set of $(n-k)$ lowest utilization tasks (i.e., entire task set $\Gamma$ ) is special on $(m-k)$ processors for the choice of $k = 0$.

However, the schedulability condition of $\texttt{GS}_{bound}$ in Theorem 8 is not satisfied for task set $\Gamma$ (i.e., utilization bound $m \cdot min\{1/2, B(m)\} = 4.116 < U^n$). Consequently, we can not guarantee the schedulability of $\Gamma$ using $\texttt{GS}_{bound}$. In summary, $\texttt{GS}_{search}$ dominates $\texttt{GS}_{bound}$ which dominates both RM-US and SM-US scheduling.

## A. Simulation Results

To quantitatively estimate the degree of dominance of $\texttt{GS}_{search}$ over the state-of-the-art algorithm, we conducted experiments using randomly generated task sets for $m = 4, 8, 16$ and 32 processors. For simulation purpose, we considered SM-US as the competitive state-of-the-art algorithm and estimated the dominance of $\texttt{GS}_{search}$ over SM-US scheduling. The conclusions from our experiments is that $\texttt{GS}_{search}$ provides an order-of-magnitude performance improvements in terms of number of scheduled task sets.

*1) Simulation Setup:* We run a number of 12 experiments. Each experiment has three simulation parameters: $m$, $minU$, $maxU$. Each experiment is characterized by the value of $m$ and the range $(minU, maxU]$. The value of $m$ denotes the number of processors we consider for an experiment. Each experiment is carried out for a number of task sets. The utilization $u_i$ of a randomly generated task $\tau_i$ of a task set is uniformly distributed within ($minU$, $maxU$].

The number of processors we consider in our experiments are $m$ = 4, 8, 16, and 32. This parameter $m$ is used to measure the impact of increasing number of processors (scalability) on number of schedulable task sets.

We consider three different utilization ranges (0, 0.5], (0.25, 0.75] and (0, 1] for ($minU$, $maxU$]. The utilization ranges (0, 0.5], (0.25, 0.75] and (0, 1] are used to experiment with *light*, *medium* and *mixed* tasks, respectively. The *four* values of $m$ and the *three* utilization ranges for ($minU$, $maxU$] constitute our 12 different experiments.

For each $m$ a total of 1000000 task sets, that are *all schedulable* using $\texttt{GS}_{search}$ on $m$ processors, are randomly-generated using a similar approach as in [8]. The 1000000 task sets are generated according to the following procedure:

1) Initially, we generate $m+1$ tasks.
2) Then we verify if the generated task set is schedulable using $\texttt{GS}_{search}$ (i.e., Theorem 9 is used for verification).
3) If this task set is $\texttt{GS}_{search}$ schedulable, then
   a) it is counted as one of the 1000000 task sets, and
   b) by adding one new task, we extend this (old) task set to a new task set and return to Step 2.
4) If this task set is not $\texttt{GS}_{search}$ schedulable, then we discard this task set and go to Step 1.

For each experiment, we calculated the *dominance* of $\texttt{GS}_{search}$ over SM-US as the percentage of the 1 million task sets that are *not schedulable* using SM-US scheduling. First, the total number of task sets (of the 1000000 $\texttt{GS}_{search}$ schedulable task sets) that are also schedulable by SM-US scheduling is counted in variable $P_{searchcount}$. Second, the *dominance* of the $\texttt{GS}_{search}$ over SM-US scheduling, denoted by $\texttt{DOM}_{\texttt{Psearch}}$, is given as follows:

$$\texttt{DOM}_{\texttt{Psearch}} = \frac{1000000 - P_{searchcount}}{1000000} \times 100\%$$

The higher the value of $\texttt{DOM}_{\texttt{Psearch}}$, the higher is the degree of dominance of $\texttt{GS}_{search}$ over SM-US scheduling. For example, if $\texttt{DOM}_{\texttt{Psearch}}$ = 20%, then 20% (20000 task sets) of the 1000000 task sets are not schedulable using SM-US scheduling.

*2) Simulation Result:* The result of the 12 experiments are given in Table I. Each of the shaded cells in Table I represents the value of $\texttt{DOM}_{\texttt{Psearch}}$ for the simulation parameters — number of processors and utilization range — given in the corresponding second row and first column of Table I.

| | $(minU, maxU]$ | | |
|---|---|---|---|
| | (0, 0.5] | (0.25, 0.75] | (0, 1] |
| $m = 4$ | 48.69 % | 99.91 % | 92.06 % |
| $m = 8$ | 38.01 % | 99.97 % | 96.95 % |
| $m = 16$ | 29.16 % | 99.99 % | 99.21 % |
| $m = 32$ | 23.87 % | 100.0 % | 99.99 % |

Table I
VALUE OF $\texttt{DOM}_{\texttt{Psearch}}$ FOR THE 12 EXPERIMENTS

The impact of utilization ranges $(minU, maxU]$ on the dominance of $\texttt{GS}_{search}$ over SM-US is significant. The value of dominance $\texttt{DOM}_{\texttt{Psearch}}$ for medium (i.e. $u_i$ is in [0.25, 0.75]) and mixed (i.e. $u_i$ is in [0, 1]) tasks is significantly higher (see the third and forth columns of Table I) than that of for the light tasks (i.e. $u_i$ is in [0, 0.5]). The SM-US scheduling fails to schedule more than 92% of the randomly generated one million task sets (having medium/mixed tasks) that are schedulable using $\texttt{GS}_{search}$.

This significant performance gain of $\texttt{GS}_{search}$ scheduling in terms of number of schedulable medium/mixed task sets results from our schedulability analysis of the “special” task systems. Remember that the threshold utilization used for SM-US is constant (i.e., $2/(3+\sqrt{5})$) while that of for

GS$_{search}$ scheduling is searched as one of the utilizations of the tasks of a given task set. Thus, a task $\tau_i$ with utilization greater than $2/(3+\sqrt{5})$ is always given the highest priority in SM-US scheduling while this task may be assigned slack-monotonic priority in GS$_{search}$ scheduling. Such exploitation of the tasks' utilization to determine the threshold utilization for GS$_{search}$ results in fewer number of tasks that are given the highest priority in order to avoid the Dhall's effect. In other words, GS$_{search}$ is more effective than SM-US in dealing with tasks having large utilization. Thus, GS$_{search}$ scheduling significantly outperforms SM-US scheduling for medium/mixed tasks in terms of number of schedulable task sets.

The number of processors used in an experiment has an impact on the value of dominance as well. As the number of processors increases, the scalability of GS$_{search}$ scheduling in terms of dominance increases for experiments with medium/mixed tasks (notice the increasing trend of dominance value in third and forth columns of Table I). This is because as $m$ increases, the number of tasks in a task set also increases due to the way tasks are generated for our experiments. As the number of tasks increases for the experiments with medium and large tasks having larger number of processors, it is more likely that the number of tasks having relatively larger individual task utilization in a task set also increases. The ability of GS$_{search}$ to deal with relatively larger number of high utilization tasks results GS$_{search}$ to schedule large number of medium/mixed task sets in comparison to that of SM-US. Consequently, the value of dominance for GS$_{search}$ increases as $m$ increases.

However, the value of dominance decreases for experiments with light tasks as we increase the number of processors (notice the decreasing trend of dominance in second column of Table I). This is because when tasks are light (i.e., $u_i \leq 0.5$), most of the individual task's utilization of a task set are relatively small. As a result most of the tasks of such task sets are given the slack-monotonic priority for both SM-US and GS$_{search}$ scheduling. In other words, the difference between the assigned priorities to the tasks in SM-US and GS$_{search}$ scheduling decreases as we increase the number of processors for experiments with light tasks. Consequently, the value of dominance decreases. Nevertheless, the value of dominance DOM$_{\mathsf{Psearch}}$ for the light tasks is still greater than 23.5% for our experiments with at most 32 processors. In summary, GS$_{search}$ scheduling of medium/mixed tasks *scales* very well with the increase in number of processors, and the dominance of GS$_{search}$ over SM-US for light tasks is also significant.

In general, if the utilization of the tasks are within $[0,1]$, then more than 92% of the 1 million randomly-generated GS$_{search}$ schedulable task sets are not schedulable using SM-US scheduling. This percentage increases up to 99.99% as we increase the number of processors in our experiments from 4 to 32. The conclusion from the experiments is thus that GS$_{search}$ provides an order-of-magnitude performance improvement in terms of number of scheduled task sets, and *scales* very well as the number of processors increases.

## *B. Uniprocessor Slack-Monotonic Scheduling*

The schedulability analysis of "special" task system on multiprocessors (Section V) enables the derivation of a higher utilization bound for *uniprocessor* slack-monotonic scheduling compared to that of the state-of-the-art result in [2]. Our proposed utilization bound for uniprocessor slack-monotonic scheduling is $F_1(u^n_{min})$. We now show how we derive this bound $F_1(u^n_{min})$ and also show that it is higher than that of the state-of-the-art result (i.e., 50%) proposed in [2].

Consider a task system $\Gamma$ that is special on uniprocessor (i.e. $m=1$). According to Property 1 of special task system $\Gamma$, we have $u^n_{max} \leq 1$ because $m/(2m-1)=1$ for $m=1$. Therefore, special task system $\Gamma$ is in fact an arbitrary task system for uniprocessor. Note that we have $0 < u^n_{min} \leq u^n_{max} \leq 1$ where $u^n_{min}$ and $u^n_{max}$ are the minimum and maximum utilization of any task in $\Gamma$, respectively.

For $m=1$, the function $F_1(x)$ is increasing within $[0,1]$ since $F_1'(x) = 1 - \frac{1}{(2-x)^2} > 0$ within $(0,1)$. Consequently, *min*$\{F_1(u^n_{min}),F_1(u^n_{max})\} = F_1(u^n_{min})$ since $u^n_{min} \leq u^n_{max}$. It is obvious from Property 2 of special task system $\Gamma$ that we have

$$U^n \leq \mathit{min}\{F_1(u^n_{min}),F_1(u^n_{max})\} = F_1(u^n_{min}) \qquad (28)$$

Using Theorem 6, the special task set $\Gamma$ is schedulable using GS$_{\mathsf{SM}}$ (uniprocessor slack-monotonic scheduling) on $m=1$ processor. Thus, based on Eq (28) our proposed utilization bound for uniprocessor slack-monotonic scheduling is $F_1(u^n_{min})$. The current state-of-the-art utilization bound for slack-monotonic uniprocessor scheduling is 50% which is proposed in [2].

We now show that $F_1(u^n_{min}) > 50\%$. Since the function $F_1(x)$ is increasing within $[0,1]$, we have $F_1(u^n_{min}) > F_1(0)$ since $u^n_{min} > 0$. Note that $F_1(0) = \frac{1(1-0)}{2-0} + 0 = 1/2 = 50\%$. Therefore, $F_1(u^n_{min}) > 50\%$. Our proposed utilization bound $F_1(u^n_{min})$ for the uniprocessor slack-monotonic scheduling is higher than that of the state-of-the-art result in [2]. $\Box$

## VIII. Conclusion

In this paper, we have proposed two static-priority assignment policies and proved their superior schedulability compared to the state-of-the-art global multiprocessor scheduling algorithms. We have also derived a higher utilization bound for uniprocessor slack-monotonic scheduling than that of the state-of-the-art result. Our experimental results showed that around more than 92% of the one million GS$_{search}$ schedulable task sets are not schedulable using SM-US. In addition, the GS$_{search}$ scheduling scales very well with the increase in number of processors.

## References

[1] B. Andersson. Global Static-Priority Preemptive Multiprocessor Scheduling with Utilization Bound 38%. In *Proc.of OPODIS*, pages 73–88, 2008.

[2] B. Andersson. The Utilization Bound of Uniprocessor Preemptive Slack-Monotonic Scheduling is 50%. In *Proc. of ACM Symp. On Applied Computing*, pages 281–283, 2008.

[3] B. Andersson, S. Baruah, and J. Jonsson. Static-Priority Scheduling on Multiprocessors. In *Proc. of RTSS*, pages 193–202, 2001.

[4] B. Andersson and J. Jonsson. The utilization bounds of partitioned and pfair static-priority scheduling on multiprocessors are 50%. In *Proc. of ECRTS*, pages 33–40, 2003.

[5] T. P. Baker. An Analysis of Fixed-Priority Schedulability on a Multiprocessor. *Real-Time Systems*, 32(1-2):49–71, 2006.

[6] S. Baruah and J. Goossens. Rate-monotonic scheduling on uniform multiprocessors. *IEEE Transactions on Computers*, 52(7):966–970, 2003.

[7] M. Bertogna, M. Cirinei, and G. Lipari. New Schedulability Tests for Real-Time Task Sets Scheduled by Deadline Monotonic on Multiprocessors. In *Proc. of OPODIS*, pages 306–321, 2005.

[8] M. Bertogna, M. Cirinei, and G. Lipari. Schedulability Analysis of Global Scheduling Algorithms on Multiprocessor Platforms. *IEEE Transactions on Parallel and Distributed Systems*, 20(4):553–566, 2009.

[9] E. Bini, T. H. C. Nguyen, P. Richard, and S. Baruah. A Response-Time Bound in Fixed-Priority Scheduling with Arbitrary Deadlines. *IEEE Trans. Comput.*, 58(2):279–286, 2009.

[10] R. I. Davis and A. Burns. Priority Assignment for Global Fixed Priority Pre-Emptive Scheduling in Multiprocessor Real-Time Systems. In *Proc. of RTSS*, pages 398–409, 2009.

[11] S. K. Dhall and C. L. Liu. On a Real-Time Scheduling Problem. *Operations Research*, 26(1):127–140, 1978.

[12] N. Fisher, S. Baruah, and T. P. Baker. The Partitioned Scheduling of Sporadic Tasks According to Static-Priorities. In *Proc. of ECRTS*, pages 118–127, 2006.

[13] J. Goossens, S. Funk, and S. Baruah. Priority-driven scheduling of periodic task systems on multiprocessors. *Real-Time Syst.*, 25(2-3):187–205, 2003.

[14] R. Ha and J. Liu. Validating timing constraints in multiprocessor and distributed real-time systems. In *Proc. of ICDCS*, pages 162 –171, 1994.

[15] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the ACM*, 20(1):46–61, 1973.

[16] D.-I. Oh and T. P. Baker. Utilization Bounds for N-Processor Rate Monotone Scheduling with Static Processor Assignment. *Real-Time Systems*, 15(2):183–192, 1998.

[17] C. A. Phillips, C. Stein, E. Torng, and J. Wein. Optimal time-critical scheduling via resource augmentation (extended abstract). In *Proc. of the ACM symposium on Theory of computing*, pages 140–149, 1997.

## Appendix

**Lemma 2.** *Consider $a$, $b$, $x$, $c$ and $d$ such that $0 \leq a \leq b \leq x \leq c \leq d \leq \frac{m}{2m-1}$ for any integer $m > 0$. The following two inequalities hold:*

$$min\{F_m(b),F_m(c)\} \leq F_m(x) \tag{3}$$

$$min\{F_m(a),F_m(d)\} \leq min\{F_m(b),F_m(c)\} \tag{4}$$

*Proof:* To show that Eq. (3) holds we will show that, within $[b, c]$ where $b \leq x \leq c$, the function $F_m(x) = \frac{m(1-x)}{2-x} + x$ achieves its absolute minimum at one of the end-points of $[b, c]$ for any given $m$. Thus, the minimum between $F_m(b)$ and $F_m(c)$ is the absolute minimum of $F_m(x)$ , and consequently Eq. (3) holds.

The first derivative of $F_m(x) = \frac{m(1-x)}{2-x} + x$ with respect of $x$ is $F'_m(x) = 1 - \frac{m}{(2-x)^2}$. By setting $F'_m(x) = 0$, we have $x = (2 \pm \sqrt{m})$. For any value of $m > 0$, the point $x = (2 + \sqrt{m})$ is outside of $(b, c)$ since $c \leq \frac{m}{2m-1} \leq 1$ for $m > 0$. Moreover, the point $x = (2 - \sqrt{m})$ is outside of $(0, \frac{m}{2m-1})$ for both $m = 1$ and $m \geq 4$. Consequently, $x = (2 - \sqrt{m})$ is also outside of $(b, c)$ because $(b, c)$ is entirely contained within $(0, \frac{m}{2m-1})$ for $m = 1$ and $m \geq 4$. So, the only possible $x$ values where $x = (2 - \sqrt{m})$ and $F'_m(x) = 0$ are $x = (2 - \sqrt{2})$ and $x = (2 - \sqrt{3})$ for $m = 2$ and $m = 3$, respectively (*stationary points*).

Since there is no stationary point of $F_m(x)$ within $(b, c)$ for $m = 1$ or $m \geq 4$, the absolute minimum of $F_m(x)$ occurs at one of the end points of $[b, c]$ for $m = 1$ and $m \geq 4$.

Now for $m = 2$, if the point $x = (2 - \sqrt{2})$ is outside of $(b, c)$, then the absolute minimum of $F_2(x)$ occurs at one of the endpoints of $[b, c]$. Otherwise, if the point $x = (2 - \sqrt{2})$ is within $(b, c)$, then the absolute minimum occurs at one of the points $x = a$, $x = (2 - \sqrt{2})$, or $x = b$. The function $F_2(x)$ is increasing within $[b, 2 - \sqrt{2})$ since $F'_2(x)$ $=1 - \frac{2}{(2-x)^2} > 0$ within $(b, 2 - \sqrt{2})$ and $F_2(x)$ is decreasing within $(2 - \sqrt{2}, c]$ since $F'_2(x)$ $=1 - \frac{2}{(2-x)^2} < 0$ within $(2 - \sqrt{2}, c)$. Therefore, the function $F_2(x)$ has its absolute maximum at $x = (2 - \sqrt{2})$. Thus, the absolute *minimum* of $F_2(x)$ occurs at one of the end points of $[b, c]$.

Similarly for $m = 3$, if the point $x = (2 - \sqrt{3})$ is outside of $(b, c)$, then the absolute minimum of $F_3(x)$ occurs at one of the endpoints of $[b, c]$. Otherwise, if the point $x = (2 - \sqrt{3})$ is within $(b, c)$, then the absolute minimum occurs at one of the points $x = a$, $x = (2 - \sqrt{3})$, or $x = b$. The function $F_3(x)$ is increasing within $[b, 2 - \sqrt{3})$ since $F'_3(x)$ $=1 - \frac{3}{(2-x)^2} > 0$ within $(b, 2 - \sqrt{3})$ and $F_3(x)$ is decreasing within $(2 - \sqrt{3}, c]$ since $F'_3(x)$ $=1 - \frac{3}{(2-x)^2} < 0$ within $(2 - \sqrt{3}, c)$. Therefore, the function $F_3(x)$ has its absolute maximum at $x = (2 - \sqrt{3})$. Consequently, the absolute *minimum* of $F_3(x)$ occurs at one of the end points of $[b, c]$.

Since the function $F_m(x)$ has its minimum at one of the end points of $[b, c]$ for any $m$, we can conclude that if $x$ is within $[b, c]$ then $F_m(x)$ is not less than the minimum between $F_m(b)$ and $F_m(c)$. Therefore, Eq. (3) holds.

Since $a \leq b \leq d$ and $a \leq c \leq d$, it follows from Eq. (3) that $min\{F_m(a),F_m(d)\} \leq F_m(b)$ and $min\{F_m(a),F_m(d)\} \leq F_m(c)$. Consequently, $min\{F_m(a),F_m(d)\} \leq min\{F_m(b),F_m(c)\}$ which proves that Eq. (4) also holds. ■

**Lemma 10.** *Consider the task system $\Gamma^k$ that is special on $m$ processors. The following inequality holds for $m \geq 1$*

$$\mathit{min}\{F_m(u^k_{min}) \, , \, F_m(u^k_{max}) \ \} \ \leq \ \frac{m^2}{2m-1} \qquad (29)$$

*Proof:* We show that the inequality in Eq. (29) holds by considering four different cases: Case (i) $m=1$, Case (ii) $m=2$, Case (iii) $m=3$, and Case (iv) $m \geq 4$. Remember that, according to Property 1 of special task set $\Gamma^k$, we have $u^k_{max} \leq \frac{m}{2m-1}$.

**Case (i)** $m = 1$: The function $F_1(x)$ is increasing within $[0,1]$ since $F_1'(x) = 1 - \frac{1}{(2-x)^2} > 0$ within $(0,1)$. Thus, the maximum of $F_1(x)$ within $[0,1]$ occurs at $x=1$, and $F_1(1)=1$. Since $\frac{m}{2m-1}=1$ for $m=1$ and $u^k_{max} \leq \frac{m}{2m-1}$, both $u^k_{min}$ and $u^k_{max}$ are within $[0,1]$. Therefore, $\mathit{min}\{F_m(u^k_{min}),F_m(u^k_{max})\} \leq F_1(1)= 1$ for $m=1$. Because $\frac{m^2}{2m-1} = 1$ for $m=1$, we also have $\mathit{min}\{F_m(u^k_{min}),F_m(u^k_{max})\} \leq \frac{m^2}{2m-1}$.

**Case (ii)** $m=2$: Since $\frac{m}{2m-1} = \frac{2}{3}$ for $m=2$ and $u^k_{max} \leq \frac{m}{2m-1}$, both $u^k_{min}$ and $u^k_{max}$ of are within $[0,\frac{2}{3}]$. The function $F_2(x)$ is increasing within $[0, 2-\sqrt{2}]$ since $F_2'(x) = 1 - \frac{2}{(2-x)^2} > 0$ within $(0, 2-\sqrt{2})$ and the function $F_2(x)$ is decreasing within $[2-\sqrt{2}, \frac{2}{3}]$ since $F_2'(x) = 1 - \frac{2}{(2-x)^2} < 0$ within $(2-\sqrt{2}, \frac{2}{3})$. Therefore, the function $F_2(x)$ has its maximum at $x = (2-\sqrt{2})$ within $[0, \frac{2}{3}]$, and $F_2(2-\sqrt{2}) = 2(2-\sqrt{2})$. Consequently, $\mathit{min}\{F_m(u^k_{min}) \, , \, F_m(u^k_{max})\} \leq F_2(2-\sqrt{2})$. Since $F_2(2-\sqrt{2}) = 2(2-\sqrt{2}) \leq \frac{4}{3} = \frac{m^2}{2m-1}$ for $m=2$, we have $\mathit{min}\{F_m(u^k_{min}),F_m(u^k_{max}\})\} \leq \frac{m^2}{2m-1}$.

**Case (iii)** $m=3$: Since $\frac{m}{2m-1} = \frac{3}{5}$ for $m=3$ and $u^k_{max} \leq \frac{m}{2m-1}$, both $u^k_{min}$and $u^k_{max}$ of are within $[0,\frac{3}{5}]$. The function $F_3(x)$ is increasing within $[0, 2-\sqrt{3}]$ since $F_3'(x) = 1 - \frac{3}{(2-x)^2} > 0$ within $(0, 2-\sqrt{3})$ and $F_3(x)$ is decreasing within $[2-\sqrt{3}, \frac{3}{5}]$ since $F_3'(x) = 1 - \frac{3}{(2-x)^2} < 0$ within $(2-\sqrt{3}, \frac{3}{5})$. Therefore, the function $F_3(x)$ has its maximum at $x = (2-\sqrt{3})$ within $[0, \frac{3}{5}]$, and $F_3(2-\sqrt{3}) = (5-2\sqrt{3})$. Consequently, $\mathit{min}\{F_m(u^k_{min}) \, , \, F_m(u^k_{max})\} \leq F_3(2-\sqrt{3})$. Since $F_3(2-\sqrt{3}) = (5-2\sqrt{3}) \leq \frac{9}{5} = \frac{m^2}{2m-1}$ for $m=3$, we have $\mathit{min}\{F_m(u^k_{min}),F_m(u^k_{max}\})\} \leq \frac{m^2}{2m-1}$.

**Case (iv)** $m \geq 4$: The function $q(m) = \frac{m}{2m-1}$ is decreasing for $m \geq 4$ because $q'(m) = \frac{-1}{(2m-1)^2} < 0$ for $m \geq 4$. Therefore, $\frac{m}{2m-1} \leq \frac{4}{7}$ for $m \geq 4$, and both $u^k_{min}$ and $u^k_{max}$ are within $[0,\frac{4}{7}]$. The function $F_m(x)$ is decreasing within $[0, \frac{4}{7}]$ for $0 \leq x \leq \frac{4}{7}$ since $F_m'(x) =1 - \frac{m}{(2-x)^2} < 0$ within $(0,\frac{4}{7})$ for $m \geq 4$. Thus, the maximum of $F_m(x)$ occurs at $x=0$, and $F_m(0) = \frac{m}{2}$. Therefore, $\mathit{min}\{F_m(u^k_{min}) \, , \, F_m(u^k_{max})\} \leq F_m(0)$. Since $F_m(0)= \frac{m}{2} < \frac{m^2}{2m-1}$ for $m \geq 4$, we have $\mathit{min}\{F_m(u^k_{min}) \, , \, F_m(u^k_{max})\} \leq \frac{m^2}{2m-1}$.

It is proved for all the cases that if $\Gamma^k$ is special on $m$ processors, then the inequality in Eq. (29) holds. ■

**Lemma 11.** *The following inequality holds for $m \geq m' \geq 1$.*

$$B(m) \leq \frac{m}{2m-1} \leq \frac{m'}{2m'-1} \qquad (30)$$

*where $B(m)$ is the function defined in Eq.* (16).

*Proof:* We prove this Lemma considering three cases: Case (i) $m=1$, Case (ii) $m=2$ and Case (iii) $m \geq 3$.

**Case (i)** $m=1$**:** For this case, we have $m=m'=1$ since $m \geq m' \geq 1$. Therefore, $\frac{m}{2m-1} = \frac{m'}{2m'-1} = 1$ for $m=m'=1$. From Eq. (16), we have $B(m) = B(1) = 1$ for $m=1$. Therefore, Eq. (30) holds.

**Case (ii)** $m=2$**:** Using Eq. (16), we have $B(2) = (2-\sqrt{2})$ for $m=2$. And $\frac{m}{2m-1} = \frac{2}{3}$ for $m=2$. Because $(2-\sqrt{2}) < \frac{2}{3}$, we have $B(m) < \frac{m}{2m-1}$ for $m=2$. The function $q(x) = \frac{x}{2x-1}$ is decreasing for $x \geq 1$ because $q'(x) = \frac{-1}{(2x-1)^2} < 0$ for $x > 1$. Thus, we have $\frac{m}{2m-1} \leq \frac{m'}{2m'-1}$ for $m \geq m'$. Consequently, $B(m) < \frac{m}{2m-1} \leq \frac{m'}{2m'-1}$. Therefore, Eq. (30) holds.

**Case (iii)** $m \geq 3$: The following inequality in Eq (31) holds for any $m$ such that $m \geq 3$.

$$\begin{aligned}
& 0 \leq m^2 - 4m + 3 && (31)\\
\equiv\ & 4m^2 - 4m + 1 \leq 5m^2 - 8m + 4 \\
\equiv\ & 2m - 1 \leq \sqrt{5m^2 - 8m + 4} \\
\equiv\ & 3m - 2 - \sqrt{5m^2 - 8m + 4} \leq m - 1 \\
\equiv\ & \frac{3m - 2 - \sqrt{5m^2 - 8m + 4}}{2m-2} \leq \frac{1}{2} \ \equiv\ B(m) \leq \frac{1}{2} \\
\Rightarrow\ & \left[\text{since } \frac{1}{2} \leq \frac{m}{2m-1} \leq \frac{m'}{2m'-1} \text{ for } m \geq m' \geq 1\right] \\
& B(m) \leq \frac{m}{2m-1} \leq \frac{m'}{2m'-1}
\end{aligned}$$

Therefore, Eq. (30) holds for all the cases. ■

**Lemma 12.** *Let $m$ and $m'$ be two integers such that $m \geq m' \geq 1$. The following inequality in Eq.* (32) *holds*

$$B(m) \leq B(m') \qquad (32)$$

*where $B(m)$ is the function defined in Eq.* (16).

*Proof:* For $m \geq 2$, the first derivative of $B(m) = \frac{3m-2-\sqrt{5m^2-8m+4}}{2m-2}$ is $B'(m) = \frac{-2(\sqrt{5m^2-8m+4}-m)}{(\sqrt{5m^2-8m+4})(2m-2)^2}$. Note that we have $B'(m) < 0$ because $\sqrt{5m^2-8m+4} > m$ for $m \geq 2$. So, $B(m)$ is *decreasing* for $m \geq 2$. Thus, the maximum of $B(m)$ occurs at $m=2$ whenever $m \geq 2$, and $B(2) = (2-\sqrt{2})$. From Eq. (16), we have $B(1) = 1$. Since $1 > (2-\sqrt{2})$, we have $B(1) > B(m)$ for any $m \geq 2$.

If $m = m'$, then Eq. (32) trivially holds. So, to prove this Lemma, we consider where $m > m'$. Note that $m \geq 2$ whenever $m > m'$ because $m' \geq 1$.

Now, if $m' = 1$, then $B(m') > B(m)$. This is because $B(1) > B(m)$ for any $m \geq 2$. Otherwise, if $m' > 1$, then $m > 2$ since we consider $m > m'$. Because the function $B(m)$ is decreasing for $m \geq 2$, we have $B(m') > B(m)$ where $m > m'$. Therefore, if $m \geq m' \geq 1$, we have $B(m) \leq B(m')$. ■