Multi-Core Partitioned Scheduling For Fixed-Priority Periodic Real-Time Tasks With Enhanced RBound

Ming Fan[1], Qiushi Han[1], Gang Quan[1], Shangping Ren[2]
[1]Department of Electrical&Computer Engineering, Florida International University, Miami, FL USA
[2]Department of Computer Science, Illinois Institute of Technology, Chicago, IL USA
E-mail[1]: {mfan001, qhan001,gaquan}@fiu.edu
E-mail[2]: ren@iit.edu

***Abstract*—This paper presents a new partitioned scheduling approach to schedule fixed-priority periodic real-time tasks on multi-core platforms under *Rate Monotonic Scheduling (RMS)* policy. We first develop a novel method to scale task periods, as well as execution times, to improve the utilization bound (i.e. RBound [7]). We further apply the task set scaling method and the enhanced utilization bound to multi-core scheduling to improve the system utilization and at the same time guarantees the satisfaction of all timing constraints. We empirically evaluate the proposed techniques and the results show clear evidence of the proposed approach outperforming earlier work existed in the literature.**

***Index Terms*—Multi-core partitioned scheduling, real-time tasks, RMS, RBound**

I. Introduction

Multi-core architecture has been widely accepted as the most important technology in the future industrial market. By providing multiple processing cores on a single chip, multi-core systems, compared with the traditional single-core systems, can significantly increase the computing performance while relaxing the power requirement. Most of the major chip manufactures have already launched 16-core chips into the market, i.e. AMD *Opteron^{TM} 6300 Series* [1]. It is not surprising that in the coming future, hundreds or even thousands of cores will be integrated into a single chip [2]. The quickly emerging trend towards multi-core platform brings urgent needs for effective and efficient techniques for the design of different types of computing systems, e.g. real-time systems.

One key problem in the design of multi-core real-time systems is how to utilize the available computing resource most efficiently while satisfying the timing constraints of all real-time tasks. One direct and effective way to address this problem is to develop appropriate scheduling algorithms. However, finding an optimal scheduling algorithm for multi-core systems is an NP-hard [3] problem. Although there exist optimal scheduling algorithms on single-core systems (e.g. *Rate Monotonic Scheduling (RMS)* and *Earliest Deadline First (EDF)* [4]), none of them remains optimal any more [5] when moving to multi-core systems. The reason is that, different from single-core scheduling, multi-core scheduling needs to decide not only when (the *priority* problem) but also where (the *allocation* problem) to execute a real-time task. Therefore, developing a sub-optimal heuristic for scheduling strategy on multi-core systems is needed.

In this paper, we are interested in studying the problem of partitioned scheduling for periodic tasks under RMS policy. Compared with the existing work on fixed-priority partitioned scheduling, we have made a number of significant contributions,

- We develop a novel method (i.e. *TSS* method) to scale task sets, and based on which, we develop an enhanced utilization bound (i.e. *RBound^{en}*) that improves upon an existing utilization bound (i.e. *RBound* [7]);
- We develop a new partitioned scheduling algorithm (i.e. *PSER* algorithm) with consideration of the relationship between periods of tasks;
- We have conducted extensive experiments to evaluate the performance of our proposed techniques.

The rest of this paper is organized as follows. Section II discusses the related work. Section III describes the preliminaries and presents an example to motivate our work. Section IV introduces a new task set scaling method, and presents an enhanced utilization bound based on the scaling method. Section V introduces a novel partitioned scheduling algorithm for fixed-priority periodic tasks on multi-core systems. Experiments and results are discussed in Section VI, and we conclude this work in Section VII.

II. Related work

A. Parametric utilization bounds for single-core systems

A *parametric utilization bound* $f(\Gamma)$ for a task set Γ is a function of the parameters of Γ, and can be used to determine the schedulability for Γ under certain specific scheduling policy (e.g. RMS). By applying the parameters of Γ into $f(\Gamma)$, all tasks in Γ can be guaranteed to meet their deadlines if the task set utilization is no more than that parametric utilization bound, i.e. $U(\Gamma) \leq f(\Gamma)$.

For single-core systems, there are several parametric utilization bounds proposed under RMS policy [4], [6], [7].

- *LLBound* [4]: The *LLBound* is a function with respect to the number of tasks, and is formulated as

$$LLBound(\Gamma) = N(2^{1/N} - 1), \tag{1}$$

where N is the number of tasks in the task set Γ. When N goes to infinity, the *LLBound* achieves its worst-case as 69%.
- *KBound* [6]: The *KBound* has a similar form as the *LLBound*, and is formulated as

$$KBound(\Gamma) = K(2^{1/K} - 1), \tag{2}$$

where K, instead of being the number of all tasks as that used by *LLBound*, is the number of harmonic task sets[1] derived from the task set Γ.
- *RBound* [7]: The *RBound* takes not only the number of tasks but also the relationship among periods into consideration, i.e.

$$RBound(\Gamma) = (N-1)(r^{1/N-1} - 1) + 2/r - 1 \tag{3}$$

where N is the number of tasks in the task set, and r is the ratio between the maximum and minimum periods and need to satisfy $1 \leq r < 2$.

Among all three utilization bounds shown in the above, it has been proved that for RMS-based single-core scheduling, the *RBound* can potentially outperform the other two (i.e. the *LLBound* and the *KBound*) [7]. However, the *RBound* can only be applied when a given task set satisfies the period constraint (i.e. $1 \leq r < 2$). Hence, in order to use the *RBound* for checking the schedulability of an arbitrary task set which may not satisfy the period constraint, we need to first transform the task set so that it satisfies the period constraint.

There are a few methods proposed to transform a task set to satisfy the condition of $1 \leq r < 2$, such as [7], [8]. In particular, Lauzac *et al.* [7] proposed a task set scaling method by scaling all tasks with respect to the maximum period. Specifically, given a task set Γ, $\forall \tau_i \in \Gamma$, the period as well as the execution time of τ_i was scaled by

$$\begin{cases} C_i' = C_i \cdot 2^{\lfloor \log \frac{T_{max}}{T_i} \rfloor} \\ T_i' = T_i \cdot 2^{\lfloor \log \frac{T_{max}}{T_i} \rfloor} \end{cases} \tag{4}$$

where T_{max} represents the maximum period among all tasks. Their method scaled all task periods with respect to, but no larger than T_{max}. They formally proved that as long as the scaled task set was feasible then the original task set was also feasible.

Kandhalu *et al.* [8] presented another method by scaling the task set with respect to the minimum period. Specifically, given a task set Γ, $\forall \tau_i \in \Gamma$, the period and the execution time of τ_i was scaled by

$$\begin{cases} C_i' = C_i / \lfloor \frac{T_i}{T_{min}} \rfloor \\ T_i' = T_i / \lfloor \frac{T_i}{T_{min}} \rfloor \end{cases} \tag{5}$$

[1]A harmonic task set is a task set in which any two tasks are period dividable.

where T_{min} is the minimum period among all tasks. This method scaled all task periods with respect to, but no smaller than T_{min}. However, this approach cannot always guarantee the schedulability of the original task set once the scaled task set is schedulable. For example, consider a task set Γ consisting of four tasks with execution time and periods as $\{(3,24),(32,100),(40,135)\}$ and $(15,140)$. According to the scaling method introduced in [8], we can transform the task set to a new task set Γ' as $\{(3,24),(8,25),(8,27),(3,28)\}$. It is not difficult to verify that the new task set Γ' is schedulable while the original task set Γ is not schedulable. In our approach, we develop a new method to scale the task periods as well as the execution times, and establish a new utilization bound.

B. Scheduling approaches for multi-core systems

The multi-core scheduling approaches can be largely categorized into three paradigms [5], [9]: *global* scheduling, *partitioned* scheduling and *semi-partitioned* scheduling. In global scheduling (e.g. [12]), all jobs (or instances) from different tasks first enter a global queue, and thus each task can be potentially executed on any core. In partitioned scheduling (e.g. [15]), each task is assigned to a dedicated core. All jobs from the same task will be executed solely on that particular core. In semi-partitioned scheduling (e.g. [16], [19]), most tasks are assigned to one particular core, and a few of tasks (i.e. usually no more than $(M-1)$ tasks, where M is the number of cores) are allowed to be split into several subtasks and assigned to different cores.

In this paper, we focus on partitioned scheduling for fixed-priority periodic tasks. In what follows, we first introduce some preliminary concepts for our work.

III. Basic concepts

In this section, we first introduce the system models this research is based upon, then introduce a traditional RBound feasibility test method.

A. System models

We assume a multi-core platform consists of M identical cores, $M \geq 2$, denoted as $\mathcal{P} = \{P_1, P_2, ..., P_M\}$. Our task model consists of N periodic tasks, represented as $\Gamma = \{\tau_1, \tau_2, ..., \tau_N\}$. Each task τ_i is characterized by a two-parameter tuple (C_i, T_i). C_i is the *worst case execution time* of τ_i, and T_i is the *inter-arrival time* (period) between any two consecutive jobs of τ_i. We assume that Γ is sorted by non-decreasing period order, i.e. for $\forall \tau_i, \tau_j \in \Gamma$, $T_i \leq T_j$ if $i < j$.

The *task utilization* of τ_i, denoted as u_i, is defined as

$$u_i = \frac{C_i}{T_i} \tag{6}$$

The *task set utilization* of Γ, denoted as $U(\Gamma)$, is defined as

$$U(\Gamma) = \sum_{i=1}^{N} u_i \tag{7}$$

where N is the number of tasks in task set Γ. Moreover, let Γ_k denote the task set assigned to core P_k, then $U(\Gamma_k)$ represents the total utilization of all tasks assigned to P_k.

The *system utilization* of $\mathcal{P}$, denoted as $U_M(\Gamma)$, is defined as

$$U_M(\Gamma) = \frac{U(\Gamma)}{M} \tag{8}$$

Intuitively, $U_M(\Gamma)$ represents the average CPU utilization among all M cores.

B. RBound feasibility test

The *RBound* [7], as what we have introduced in Section II-A, can be used as a feasibility test method for scheduling fixed-priority periodic tasks on single-core systems. We formally present the RBound feasibility test approach with Theorem 1.

Theorem 1: [7] Given a task set Γ, let Γ' be the task set by scaling all tasks in Γ (i.e. $\forall \tau_i \in \Gamma$) through

$$\begin{cases} C_i' = C_i \cdot 2^{\lfloor \log \frac{T_{max}}{T_i} \rfloor} \\ T_i' = T_i \cdot 2^{\lfloor \log \frac{T_{max}}{T_i} \rfloor} \end{cases} \tag{9}$$

where $T_{max} = \max_{\forall \tau_i \in \Gamma} T_i$. Then Γ is schedulable on a single-core system under RMS if

$$U(\Gamma) \leq RBound(\Gamma') \tag{10}$$

The $RBound(*)$ is given by equation (3).

From Theorem 1, we can see that the *RBound* feasibility test first scales all tasks in Γ with respect to the maximum period, and then predicts the schedulability of Γ by comparing its utilization with the value of RBound under Γ'. In what follows, we present a new task set transformation method, based on which, we then develop a novel partitioned scheduling algorithm.

IV. Task set scaling with respect to the period of an arbitrary task

In order to apply the *RBound* to test the schedulability of a task set, one key point is to develop an effective and efficient method to transform the task set to a new one such that the ratio of the maximum and minimum period is between 1 and 2 [7]. In addition, we need to guarantee that once the new task set is schedulable, so is the original task set.

To transform a task set, one approach [7] is to fix T_{max} and scale up the rest task periods towards T_{max} so that the maximum/minimum period ratio is between 1 and 2. Another effort [8] is to keep the T_{min} unchanged and scale down task periods such that the maximum/minimum period ratio is between 1 and 2. Unfortunately, as explained before(see Section II-A), this approach cannot guarantee the schedulability of the original task set even though the new task set can be schedulable. In this section, we introduce a new method to scale task periods based on the period of *any task* in the task set, and most importantly, we guarantee that the original task set is schedulable if the new task set is schedulable.

A. Task set scaling (TSS)

Instead of using a restricted transformation, such as scaling the entire task set only with respect to a unique task (i.e. the task with the maximum period), we introduce a more general and flexible task set transformation method, denoted as the *TSS* method, which can scale a task set with respect to the period of an arbitrary task in a given task set.

Algorithm 1 $TSS(\Gamma, \tau_k)$

Require:

1) Γ: input task set, sorted with non-decreasing period order;

2) τ_k: the k^{th} task in Γ, based on which the task set is scaled.

1: $N = |\Gamma|$;
2: $T_k' = Z_k = T_k$, and $C_k' = C_k$;
3: *// step 1: transform LOWER-priority tasks into harmonic*;
4: **for** $i = k+1$ **to** $N \quad Z_i = Z_{i-1} \cdot \lfloor \frac{T_i}{Z_{i-1}} \rfloor$ **end for**;
5: *// step 2: scale all tasks with respect to* τ_k;
6: **for** $i = 1$ to $k-1$ **do**
7: $\quad R_i = 2^{\lfloor log_2 \frac{T_k}{T_i} \rfloor}$
8: $\quad T_i' = T_i \cdot R_i$
9: $\quad C_i' = C_i \cdot R_i$
10: **end for**
11: **for** $i = k+1$ to N **do**
12: $\quad R_i = Z_i / T_k$;
13: $\quad T_i' = Z_i / R_i$;
14: $\quad C_i' = C_i / R_i$;
15: **end for**
16: **return** Γ';

Algorithm 1 shows the details of our proposed *Task Set Scaling* (*TSS*) method. We assume that the input task set Γ is sorted with non-decreasing period order, i.e. for any two tasks τ_i and τ_j, it holds $T_i \leq T_j$ if $i < j$. *TSS* method transforms the entire task set Γ into another task set Γ' by scaling all tasks with respect to τ_k's period, i.e. T_k, where τ_k is an arbitrary task in Γ.

There are two major steps in Algorithm 1: 1) Tasks with priorities lower than τ_k are transformed into harmonic tasks with their periods being integer multiples of T_k (line 4); 2) Tasks with priorities higher than τ_k are scaled up (line 6-10), and tasks with priorities lower than or equal to τ_k are scaled such that the new period is equal to T_k (line 11-15). After all tasks in Γ are scaled appropriately, the corresponding task set Γ' is returned. In what follows, we discuss the relationship between Γ' and Γ in terms of schedulability.

B. Feasibility relationship between Γ and Γ'

In this subsection, we discuss the relationship between a transformed task set Γ' and its original task set Γ in terms of feasibility. we show that if Γ' is schedulable under RMS, then Γ must be schedulable under RMS. This is essential to the application of our utilization bound in schedulability test.

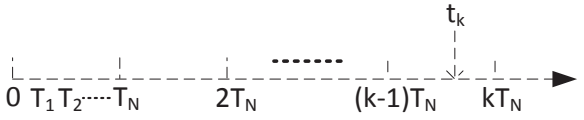


Fig. 1. Proof of Theorem 2: given a task set Γ with $T_1 \leq T_2 ... \leq T_N$ and $\tau_N = (C_N, T_N)$, transform Γ into Γ^* such that $\tau_N^* = (kC_N, kT_N)$ and $\tau_i^* = \tau_i, \forall i < N$.

Theorem 2: Given a task set $\Gamma = \{\tau_1, \tau_2, ..., \tau_{N-1}, \tau_N\}$ with $T_1 \leq T_2 ... \leq T_N$, let $\Gamma^* = \{\tau_1, \tau_2, ..., \tau_{N-1}, \tau_N^*\}$, such that $\tau_N^* = (C_N^*, T_N^*)$ satisfies that

$$C_N^* = k \cdot C_N, \qquad T_N^* = k \cdot T_N \tag{11}$$

where k is an arbitrary positive integer. If Γ is schedulable on a single-core system under *RMS*, then Γ^* must be schedulable on a single-core system under *RMS*.

Proof: Since the first $N-1$ tasks always have higher priorities than the N^{th} task in either Γ or Γ^*, their schedulability does not change. Thus, we only need to prove that τ_N^* is schedulable in Γ^*. Consider the k^{th} instance of task τ_N in Γ. Since Γ is schedulable, we know that the k^{th} instance of task τ_N must be able to meet its deadline. In other words, there must exist a time point t_k, where $t_k \in ((k-1) \cdot T_N, k \cdot T_N]$, such that (see Figure 1)

$$\sum_{i=1}^{N-1} C_i \cdot \lceil \frac{t_k}{T_i} \rceil + N \cdot C_N \leq t_k \tag{12}$$

According to equation (11), we have that $C_N^* = k \cdot C_N$. Thus, the above can be rewritten as

$$\sum_{i=1}^{N-1} C_i \cdot \lceil \frac{t_k}{T_i} \rceil + C_N^* \leq t_k \tag{13}$$

The above inequality means that at time point t_k, τ_N^* as well as all other higher priority tasks can completely finish their execution requirements. Note that $t_k \leq k \cdot T_N = T_N^*$. Thus, τ_N^* is schedulable in Γ^*. Therefore, we can see that if Γ is schedulable, then Γ^* must be schedulable. □

Next, for any given task set Γ, let Γ' be the task set obtained by applying *TSS* method given by Algorithm 1. We prove that the ratio between the maximum and minimum periods of all tasks in Γ' is less than 2. We formally conclude this property in Lemma 1.

Lemma 1: Given a task set Γ sorted with non-decreasing period order and a task τ_k representing the k^{th} task in Γ, let Γ' be the scaled task set obtained by applying *TSS* method (see Algorithm 1). Then we have

$$1 \leq \frac{T'_{max}}{T'_{min}} < 2 \tag{14}$$

where $T'_{max} = \max_{\forall \tau'_i \in \Gamma'} T'_i$ and $T'_{min} = \min_{\forall \tau'_i \in \Gamma'} T'_i$.

Proof: We prove this property by showing that T'_k (same as T_k) in the transformed task set Γ' is the maximum period and the ratio between T_k and any other period is less than 2. On one hand, for any task τ_i with priority higher than τ_k, i.e. $i < k$, according to Algorithm 1, we have

$$\frac{T'_k}{T'_i} = \frac{T_k}{T_i \cdot 2^{\lfloor \log \frac{T_k}{T_i} \rfloor}} \tag{15}$$

from which we can derive that

$$1 = \frac{T_k}{T_i \cdot 2^{\log \frac{T_k}{T_i}}} \leq \frac{T'_k}{T'_i} < \frac{T_k}{T_i \cdot 2^{(\log \frac{T_k}{T_i} - 1)}} = \frac{2T_k}{T_i \cdot 2^{\log \frac{T_k}{T_i}}} = 2 \tag{16}$$

On the other hand, for any task τ_i with priority lower than or equal to τ_k, i.e $i > k$, according to Algorithm 1, its transformed period can be represented as

$$T'_i = \frac{Z_i}{Z_i / T_k} = T_k \tag{17}$$

Based on the above, we can immediately get

$$\frac{T'_k}{T'_i} = \frac{T'_k}{T_k} = 1 \tag{18}$$

where $i > k$. Thus far, we show $T'_k(T_k)$ is the maximum period in Γ', i.e. $\forall i$, $\frac{T'_k}{T'_i} >= 1$, and the ratio of T'_k over any other period T'_i is less than 2. Therefore, Lemma 1 is proved. □

Now we are ready to show that after applying *TSS* method, the schedulability of the original task set Γ can be predicted by that of Γ'. We formulate this property in Theorem 3

Theorem 3: Given a task set Γ, let Γ' be the scaled task set obtained by applying the *TSS* method with respect to any task τ_k in Γ. If $U(\Gamma') \leq RBound(\Gamma')$, then Γ must be schedulable on a single-core system under *RMS*.

Proof: According to Lemma 1, we have that $\Gamma' = \{\tau'_1, ..., \tau'_{k-1}, \tau'_k, \tau'_{k+1}, ..., \tau'_N\}$ satisfies that $1 \leq r < 2$, where r is the ratio between the maximum and minimum periods in Γ'. Thus, if $U(\Gamma') \leq RBound(\Gamma')$, according to Theorem 1, Γ' is schedulable on a single-core system under RMS.

Next, for $\forall i > k$, according to line 11-15 in Algorithm 2, we have that

$$T'_i = Z_i / R_i = Z_i / (Z_i / T_k) = T_k = T'_k \tag{19}$$

Thus, $\tau'_k, \tau'_{k+1}, ..., \tau'_N$ have the same as well as the lowest priority in Γ'. Moreover, $\tau'_1, ..., \tau'_{k-1}$ are tasks with priorities higher than τ'_k before as well as after the transformation. Based on Lemma 2 in work [7], if Γ' is schedulable, we know that the following task set $\widehat{\Gamma}'$ must be schedulable.

$$\widehat{\Gamma}' = \{\tau_1, ..., \tau_{k-1}, \tau_k, \tau'_{k+1}, ..., \tau'_N\} \tag{20}$$

Then we construct task set Γ^* from $\widehat{\Gamma}'$ by replacing τ'_i with τ_i^*, where $i = k+1, ..., N$, such that $T_i^* = Z_i$ (based on line 4 in Algorithm 2) and $C_i^* = C'_i \cdot (T_i^* / T'_i)$.

$$\Gamma^* = \{\tau_1, ..., \tau_{k-1}, \tau_k, \tau*_{k+1}, ..., \tau*_N\} \tag{21}$$

For $i = k, ..., N-1$, we have that T_{i+1}^* is an integer multiple of T_i^*, thus according to Theorem 2, if $\widehat{\Gamma}'$ is schedulable, Γ^* must be schedulable.

Finally, since $T_{k+1}^* \leq ... \leq T_N^*$ and $T_{k+1} \leq ... \leq T_N$, thus by extending T_i^* to T_i, where $i = k+1, ..., N$, the schedulability of all tasks do not change. In other words, if Γ^* is schedulable, the original task set $\Gamma = \{\tau_1, ..., \tau_{k-1}, \tau_k, \tau_{k+1}, ..., \tau_N\}$ must be schedulable.

In sum, after applying the *TSS* method on a given task set Γ, if the scaled task set Γ' satisfies that $U(\Gamma') \leq RBound(\Gamma')$, then Γ is schedulable on a single-core system under *RMS*. □

C. Enhanced RBound

In this part, we propose an enhanced utilization bound based on our *TSS* method, and then introduce a new feasibility test method.

First, after the transformation by *TSS*, we can apply the *RBound* function given by equation (3) to evaluate the schedulability of the transformed task set, and therefore that of the original task set. By applying *TSS* with different initial tasks, we can possibly attain a higher utilization bound. Subsequently, we derive our *enhanced utilization bound* in the following equation,

$$RBound^{en}(\Gamma) = \max_{\forall \tau_i \in \Gamma} \left\{ RBound(\Gamma'_i) \ | \Gamma'_i = TSS(\Gamma, \tau_i) \right\} \quad (22)$$

where $RBound(*)$ is the utilization bound function given by equation (3) and *TSS(*, *)* is our task set scaling method shown in Algorithm 1.

Next, in light of Theorem 3, we know that the task set Γ is guaranteed to be schedulable if there exists a task $\tau_i \in \Gamma$ such that the condition $U(\Gamma') \leq RBound(\Gamma')$ is satisfied. The feasibility test method based on our $RBound^{en}$ is concluded with Theorem 4 in light of Theorem 1, Lemma 1 and Theorem 3.

Theorem 4: Given a task set Γ, if $\exists \tau_i$, $\tau_i \in \Gamma$, such that

$$U(\Gamma') \leq RBound(\Gamma') \quad (23)$$

where $\Gamma' = TSS(\Gamma, \tau_i)$, then Γ is schedulable on a single-core system under *RMS*.

Theorem 4 provides a new feasibility test method by applying our proposed *TSS* method to obtain an enhanced RBound to predict the schedulability for a given task set. It is not surprising to see that our proposed feasibility test (given by Theorem 4) can always outperform the previous RBound feasibility test[7].

Corollary 1: Given a task set Γ, if Γ can successfully pass the traditional RBound feasibility test given by Theorem 1, then Γ must be able to successfully pass the enhance RBound feasibility test given by Theorem 4.

Proof: If Γ can pass the traditional RBound feasibility test successfully, according to Theorem 1, we must have that

$$U(\Gamma) \leq RBound(\Gamma'_1)$$

where Γ'_1 is obtained by using equation (9). Note that $U(\Gamma) = U(\Gamma'_1)$. On other hand, let τ_N represent the task with the maximum period in Γ, by using τ_N to our *TSS* method, we can get a scaled task set, denoted as Γ'_2. According to *TSS* method given by Algorithm 1, we have that Γ'_2 is exactly the same as Γ'_1. Thus, we have

$$U(\Gamma'_2) = U(\Gamma) \leq RBound(\Gamma'_1) = RBound(\Gamma'_2)$$

According to Theorem 4, we get that Γ is schedulable. Therefore, if Γ can successfully pass the traditional RBound feasibility test given by Theorem 1, then Γ must be able to successfully pass the enhance RBound feasibility test given by Theorem 4. □

From Corollary 1, we can see that our proposed feasibility test method can always outperform the previous RBound feasibility test method. In fact, the traditional feasibility test condition (given by equation (10)) is only one of the conditions tested in our enhance feasibility test. In other words, we proposed feasibility test method completely covers the case of the traditional RBound feasibility test.

For example, consider the following three tasks, $\tau_1 = (7,10)$, $\tau_2 = (1,11)$ and $\tau_3 = (1,15)$. According to the traditional RBound feasibility test (see Theorem 1), we get that

$$U(\Gamma) = 0.858 > RBound(\Gamma'_1) = 0.783$$

Thus Γ is not schedulable under the traditional RBound test. However, by transforming Γ with respect to τ_2 under our *TSS* method, i.e. $\Gamma'_2 = TSS(\Gamma, \tau_2)$, we can get the $\Gamma'_2 = \{(7,10),(1,11),(1,11)\}$. Directly, we can derive that

$$U(\Gamma'_2) = 0.882 < RBound(\Gamma'_2) = 0.916$$

According to Theorem 4, our proposed feasibility test can guarantee that Γ is schedulable on a single-core system under RMS.

V. Partitioned scheduling with enhanced utilization bound

In this section, we first present a new multi-core scheduling algorithm, *Partitioned Scheduling with Enhanced RBound (PSER)*, then we prove its schedulability after a successful partition.

A. Algorithm detail

PSER is a multi-core partitioned scheduling algorithm, which adopts our proposed *TSS* to make the partitioning decisions for a task set.

We show the details of *PSER* in Algorithm 2. During each iteration, we assign a group of tasks to a core such that the core utilization is maximized and the tasks are deemed to be schedulable according to the *RBound*. The algorithm is terminated when either all the tasks are successfully assigned or a schedulable partition can not be found.

Note that in order to find the best combination of tasks in each iteration, the unassigned task set Γ is transformed with respect to each of tasks in Γ (i.e. line 6). Thus, by exploring all transformations with different initial conditions, we can optimize the grouping decisions and as a result, maximize the system utilization.

B. Schedulability analysis

After successfully partitioning all tasks by *PSER*, we apply the *RMS* on each core as the local scheduling policy. We prove that the schedulability of any task set after a successful partitioning by *PSER* can be guaranteed.

Theorem 5: If a task set Γ is successfully partitioned by *PSER* on M cores and scheduled under *RMS*, then all tasks can meet their deadlines.

Proof: Assume that a task set Γ is successfully partitioned by *PSER*, then we prove that each core can guarantee the schedulability of all tasks assigned to it. Consider an arbitrary core $P_m \in \mathcal{P}$, and let Γ_m be the corresponding task set assigned to P_m. Once *PSER* finishes successfully,

Algorithm 2 Partitioned Scheduling with Enhanced RBound (PSER)

Require:
1) Task set :$\Gamma = \{\tau_1, \tau_2, ...\tau_N\}$;
2) Multi-core : $\mathcal{P} = \{P_1, P_2, ..., P_M\}$;

1: sort Γ with non-decreasing period order;
2: **for** $m = 1$ to M **do**
3: **if** $\Gamma == \emptyset$ **then** break, **end if**;
4: $U_{opt} = 0$;
5: **for** $i = 1$ to $|\Gamma|$ **do**
6: $\Gamma' = TSS(\Gamma, \tau_i)$;
7: sort Γ' with non-increasing period order (for tasks with same periods, sort them with non-increasing utilization order);
8: $\Gamma'_{sub} = \emptyset$;
9: **for** $j = 1$ to $|\Gamma'|$ **do**
10: **if** $U(\Gamma'_{sub} \bigcup \{\tau'_j\}) \leq RBound(\Gamma'_{sub} \bigcup \{\tau'_j\})$ **then**
11: $\Gamma'_{sub} = \Gamma'_{sub} \bigcup \{\tau'_j\}$;
12: **end if**
13: **end for**
14: **if** $U(\Gamma_{sub}) > U_{opt}$ **then**
15: $\Gamma_{opt} = \Gamma_{sub}$;
16: $U_{opt} = U(\Gamma_{sub})$;
17: **end if**
18: **end for**
19: assign Γ_{opt} to core P_m, and remove Γ_{opt} from Γ;
20: **end for**
21: **if** $\Gamma = \emptyset$ **then** return "success"; **else** return "failure", **end if**;

according to line 9-17 in Algorithm 2, we know that there must exist $\tau_i \in \Gamma_m$, such that

$$U(\Gamma'_m) \leq RBound(\Gamma'_m) \tag{24}$$

where $\Gamma'_m = TSS(\Gamma, \tau_i)$. According to Theorem 4, Γ_m is schedulable on core P_m under *RMS* policy. Therefore, for an arbitrary core P_m, after the partitioning procedure *PSER* is successfully completed, all tasks assigned to P_m can meet their deadline. Thus far, this theorem is proved. □

In what follows, we will evaluate the performance of our proposed algorithms with experiments.

VI. Experimental evaluations

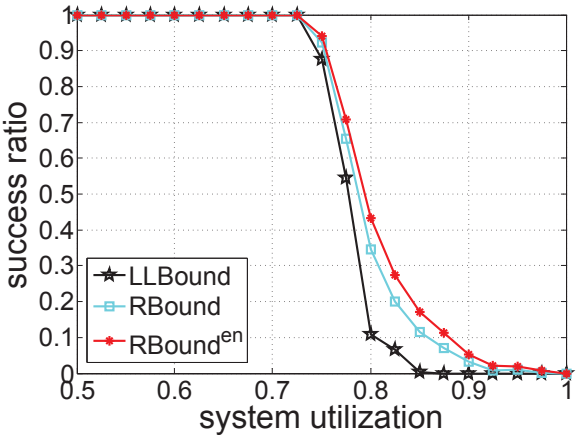
In this section, we evaluate our proposed techniques with experiments. We conducted two groups of experiments to study the performance of our enhanced utilization bound (i.e. $RBound^{en}$) and the multi-core scheduling algorithm (i.e. *PSER*), respectively.

A. Experiment 1 : efficiency of our enhanced utilization bound $RBound^{en}$

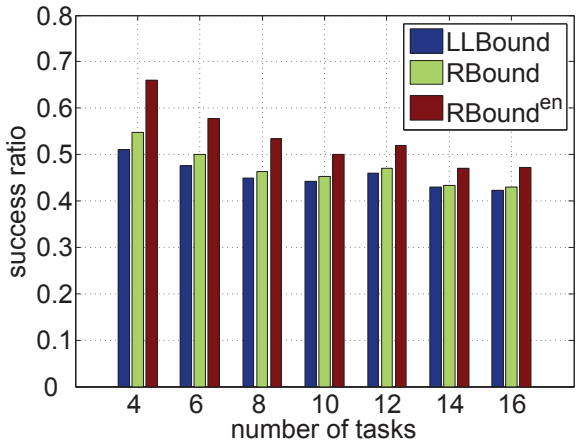
In this group of experiments, we evaluate the efficiency of our proposed utilization bound, i.e. $RBound^{en}$, on a single-core platform. Three different utilization bounds were implemented:

- *LLBound* [4]: See equation (1).
- *RBound* [7]: Calculate the utilization bound by equation (3) under the traditional task set transformation method as given by equation (9).
- $RBound^{en}$ (our method): Calculate the utilization bound by equation (22).

We tested the above three utilization bounds with respect to the system utilization and the number of tasks, respectively. In the first experiment, we varied the system utilization (see equation (8)) from 0.5 to 1 with an increment of 0.025. In the second experiment, we varied the number of tasks from 4 to 16 with an increment of 2, and the total utilization of all tasks at each test point is randomly generated with [0.5,1]. The task periods are randomly generated within $[10, 500]$. For each testing point, we generated 500 task sets, and the performance was evaluated by using the metric *success ratio*, which is the fraction of the number of feasible task sets over the number of total task sets. The experimental results were collected and plotted in Figure 2.



(a) Performance v.s. system utilization



(b) Performance v.s. number of tasks

Fig. 2. Efficiency of our enhanced utilization bound on a single core.

Figure 2 shows the performance of three different utilization bounds with respect to system utilization and number of tasks, respectively. From Figure 2(a) and 2(b), we can observe that our proposed $RBound^{en}$ outperforms the others, i.e. *LLBound* and *RBound*. For example, in Figure

2(a), when system utilization is 0.8, $RBound^{en}$ can achieve a success ratio around 0.49, an improvement of 29% over *RBound* (0.38), and an improvement of 2.7 times over *LLBound* (0.13). In Figure 2(b), when the number of tasks is 12, the success ratio of $RBound^{em}$ is 52%, while that ratio of *LLBound* and *RBound* are 47% and 46%, respectively.

Compared with *RBound*, the improvement of our proposed utilization bound (i.e. $RBound^{en}$) comes from the fact that, instead of choosing only one task period as the task set transformation standard, $RBound^{en}$ takes all periods into consideration, and find the optimal transformation among all task set scalings. Thus our proposed $RBound^{en}$ always outperforms the traditional *RBound*.

B. Experiment 2 : performance of our proposed multi-core partitioning algorithm PSER

In this group of experiments, we evaluate the performance of our proposed multi-core scheduling algorithm, i.e. *PSER* algorithm, by comparing with other three different algorithms.

- *LLBound-WF*: Partitions each task based on the Worst-Fit (WF) bin-packing method (which assigns each task to the core with the largest remaining capacity that can accommodate the task), and checks the capacity of each local core with the *LLBound* (see equation (1)).
- *LLBound-BF*: Partitions each task based on the Best-Fit (BF) bin-packing method (which assigns each task to the core with the smallest remaining capacity that can successfully accommodate that task), and checks the capacity of each local core with the *LLBound*.
- *RBOUNDMP* [7]: Exploits the *RBound* with traditional task set scaling method (see equation (9)), and allocates each task based on the Best-Fit strategy under the *RBound*.
- *PSER*: Our proposed partitioned scheduling algorithm (see Algorithm 2).

To study the performance of different multi-core scheduling approaches, we conducted two sub-sets of experiments, for light and general task sets, respectively. In light task sets, the utilization of each task was randomly generated within $[0, 0.5]$, while in general task sets, the utilization of each task was randomly generated within $[0, 1]$. For both experiments, we tested all four approaches with different number of cores, i.e. $M = 4$, 8, and 16. For each testing point, we generated 500 task sets, each of which has a utilization randomly generated within $[0.5, 1]$. The experimental results are collected and shown in Figure 3 and Figure 4.

Figure 3 shows the experimental results for task sets containing only light tasks ($u_i \in [0, 0.5]$). From Figure 3(a), 3(b) and 3(c), we can observe that the proposed *PSER* approach outperforms all other three approaches. For example, in Figure 3(a), when the system utilization is 0.8, *PSER* can achieve a success ratio up to 0.97, while that of *RBOUNDMP*, *LLBound-BF* and *LLBound-WF* are only 0.78, 0.05 and 0, respectively.

Figure 4 shows the experimental results for general task sets containing both heavy ($u_i \in [0.5, 1]$) and light ($u_i \in [0, 0.5]$) tasks. From Figure 4, we can also observe that *PSER* outperforms the other three approaches. For example, in Figure 4(c), when the system utilization is 0.85, *PSER* can achieve a success ratio of 0.69, which is 1.49 times of that by *RBOUNDMP* (0.47), 6.9 times of that by *LLBound-BF*(0.10) and 11.5 times of that by *LLBound-WF* (0.06).

In summary, our experimental results clearly show that the proposed *PSER* approach, which exploits our task set scaling (*TSS*) method and enhanced utilization bound ($RBound^{en}$), can effectively improve the schedulability of multi-core partitioned scheduling compared with the previous related work.

VII. Conclusions

Multi-core scheduling problem is the most fundamental problem in real-time embedded system design. Partitioned scheduling, as one of the major types in multi-core scheduling design, becomes more important as the multi-core platform emerging as the dominant technology in both research and industry fields.

In this paper, we first introduced a novel task set scaling method (i.e. *TSS*) which scales a task set with respect to any given task period in the task set. Based on our task set scaling method, we further presented an enhanced utilization bound for checking the schedulability of fixed-priority periodic tasks on single-core systems. In addition, we proposed a novel partitioned scheduling algorithm (i.e. *PSER*) to optimize the system utilization by assigning tasks group by group based on our task set scaling method and the enhanced utilization bound. We formally proved that our scheduling algorithm could guarantee the feasibility of any task set successfully passed the partitioning procedure. Our extensive experimental results demonstrated that the proposed algorithm could effectively improve the scheduling performance compared with the previous work.

Acknowledgement

This work is supported in part by NSF under projects CNS-0969013, CNS-0917021, and CNS-1018108.

References

[1] AMD. [Online]. Available: "http://www.amd.com/US/PRODUCTS/SERVER/PROCESSORS/6000-SERIES-PLATFORM/6300/Pages/6300-series-processors.aspx"

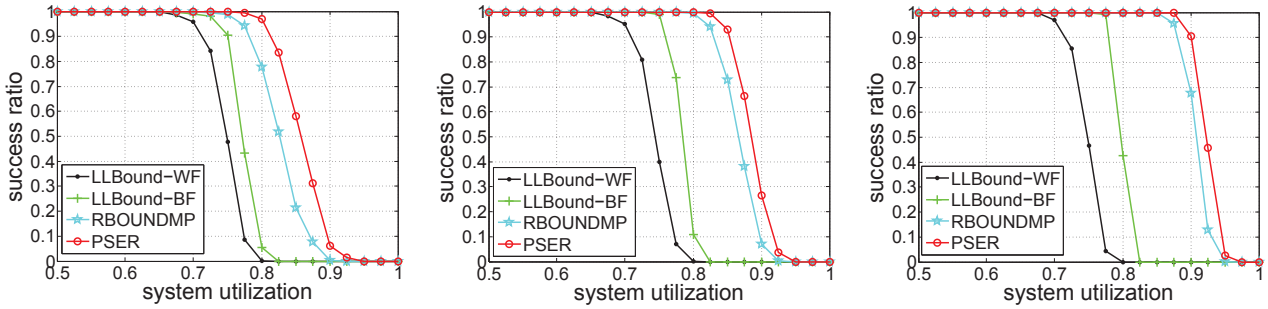
[2] D. Yeh, L.-S. Peh, S. Borkar, J. Darringer, A. Agarwal, and W. Hwu, "Thousand-core chips [roundtable]," *Design Test of Computers, IEEE*, vol. 25, no. 3, pp. 272 –278, may-june 2008.

[3] K. Shin and P. Ramanathan, "Real-time computing: a new discipline of computer science and engineering," *Proceedings of the IEEE*, vol. 82, no. 1, pp. 6 –24, Jan 1994.

[4] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, pp. 46–61, January 1973. [Online]. Available: http://doi.acm.org/10.1145/321738.321743

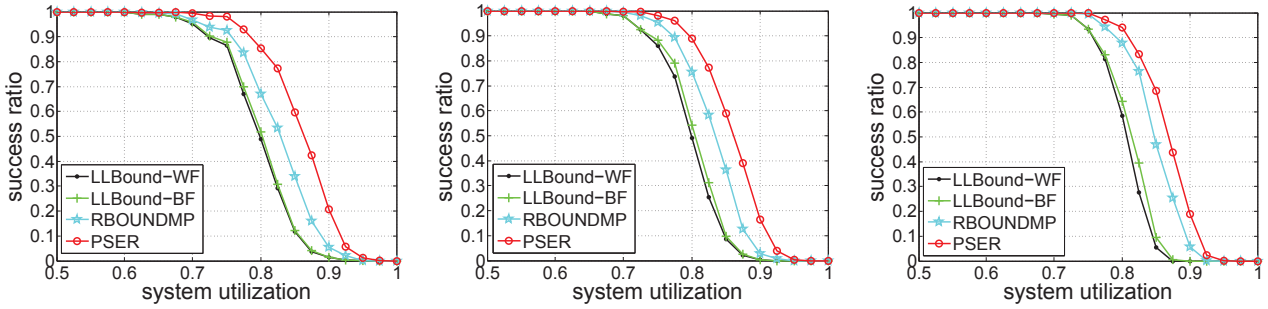
[5] S. K. Dhall and C. L. Liu, "On a real-time scheduling problem," *Operations Research*, vol. 26, no. 1, pp. pp. 127–140, 1978. [Online]. Available: http://www.jstor.org/stable/169896

[6] T.-W. Kuo and A. Mok, "Load adjustment in adaptive real-time systems," in *Real-Time Systems Symposium, 1991. Proceedings., Twelfth*, Dec 1991, pp. 160 –170.



(a) Number of cores: M = 4 (b) Number of cores: M = 8 (c) Number of cores: M = 16

Fig. 3. Experimental results for light task sets ($u_i \in [0, 0.5]$)



(a) Number of cores: M = 4 (b) Number of cores: M = 8 (c) Number of cores: M = 16

Fig. 4. Experimental results for general task sets ($u_i \in [0, 1]$)

[7] S. Lauzac, R. Melhem, and D. Mossé, "An Efficient RMS Admission Control and Its Application to Multiprocessor Scheduling," in *IPPS/SPDP Parallel Processing Symposium*, Mar. 1998.

[8] A. Kandhalu, K. Lakshmanan, J. Kim, and R. Rajkumar, "pCOMPATS: Period-Compatible Task Allocation and Splitting on Multicore Processors," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, Apr. 2012.

[9] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, and S. Baruah, "A categorization of real-time multiprocessor scheduling problems and algorithms," in *Handbook on Scheduling Algorithms, Methods, and Models*. Chapman Hall/CRC, Boca, 2004.

[10] B. Andersson, S. Baruah, and J. Jonsson, "Static-priority scheduling on multiprocessors," in *Real-Time Systems Symposium, 2001. (RTSS 2001). Proceedings. 22nd IEEE*, Dec 2001, pp. 193 – 202.

[11] B. Andersson and J. Jonsson, "The utilization bounds of partitioned and pfair static-priority scheduling on multiprocessors are 50%," in *Real-Time Systems, 2003. Proceedings. 15th Euromicro Conference on*, July 2003, pp. 33 – 40.

[12] B. Andersson, "Global static-priority preemptive multiprocessor scheduling with utilization bound 38%," in *Principles of Distributed Systems*, ser. Lecture Notes in Computer Science, T. Baker, A. Bui, and S. Tixeuil, Eds. Springer Berlin / Heidelberg, 2008, vol. 5401, pp. 73–88, 10.1007/978-3-540-92221-6_7. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-92221-6_7

[13] T. Baker, "An analysis of edf schedulability on a multiprocessor," *Parallel and Distributed Systems, IEEE Transactions on*, 2005.

[14] B. Andersson, K. Bletsas, and S. Baruah, "Scheduling arbitrary-deadline sporadic task systems on multiprocessors," in *Real-Time Systems Symposium, 2008*, Dec 2008, pp. 385 –394.

[15] K. Lakshmanan, R. Rajkumar, and J. Lehoczky, "Partitioned fixed-priority preemptive scheduling for multi-core processors," in *Real-Time Systems, 2009. ECRTS '09. 21st Euromicro Conference on*, July 2009, pp. 239 –248.

[16] S. Kato and N. Yamasaki, "Semi-partitioned fixed-priority scheduling on multiprocessors," in *Real-Time and Embedded Technology and Applications Symposium, 2009. RTAS 2009. 15th IEEE*, April 2009, pp. 23 –32.

[17] N. Guan, M. Stigge, W. Yi, and G. Yu, "Fixed-priority multiprocessor scheduling with liu and layland's utilization bound," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2010 16th IEEE*, April 2010, pp. 165 –174.

[18] A. Bastoni, B. B. Brandenburg, and J. H. Anderson, "Is Semi-Partitioned Scheduling Practical?" in *Euromicro Conference on Real-Time Systems (ECRTS)*, Jul. 2011.

[19] M. Fan and G. Quan, "Harmonic semi-partitioned scheduling for fixed-priority real-time tasks on multi-core platform," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, Mar. 2012, pp. 503 –508.

[20] N. Guan, M. Stigge, Y. Wang, and G. Yu, "Parametric utilization bounds for fixed-priority multiprocessor scheduling," in *Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, May, pp. 261–272.

[21] N. Guan, M. Stigge, W. Yi, and G. Yu, "Fixed-priority multiprocessor scheduling: Beyond liu and layland's utilization bound," in *Real-Time Systems Symposium, 2010, Work In Progress*, Dec 2010.

[22] C.-C. Han and H.-Y. Tyan, "A Better Polynomial-Time Schedulability Test for Real-Time Fixed-Priority Scheduling Algorithms," in *Proc. IEEE Real-Time Systems Symposium (RTSS)*, Dec. 1997.