The Journal of Systems and Software xxx (2014) xxx–xxx

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Enhanced fixed-priority real-time scheduling on multi-core platforms by exploiting task period relationship☆

Ming Fan[a], Qiushi Han[a,*], Shuo Liu[a], Shaolei Ren[a,c], Gang Quan[a], Shangping Ren[b]

[a] Department of Electrical and Computer Engineering, Florida International University, 10555 West Flagler Street, Miami, FL 33174, United States
[b] Department of Computer Science, Illinois Institute of Technology, 10 West 31st Street, Chicago, IL 60616, United States
[c] School of Computing and Information Sciences, Florida International University, 11200 SW 8th Street, ECS 350, Miami, FL 33199, United States

ARTICLE INFO

Article history:
Received 1 May 2014
Received in revised form 2 September 2014
Accepted 4 September 2014
Available online xxx

Keywords:
Partitioned scheduling
RMS
Harmonic

ABSTRACT

One common approach for multi-core partitioned scheduling problem is to transform this problem into a traditional bin-packing problem, with the utilization of a task being the "size" of the object and the utilization bound of a processing core being the "capacity" of the bin. However, this approach ignores the fact that some implicit relations among tasks may significantly affect the feasibility of the tasks allocated to each local core. In this paper, we study the problem of partitioned scheduling of periodic real-time tasks on multi-core platforms under the *Rate Monotonic Scheduling* (*RMS*) policy. We present two effective and efficient partitioned scheduling algorithms, i.e. *PSER* and *HAPS*, by exploiting the fact that the utilization bound of a task set increases as task periods are closer to harmonic on a single-core platform. We formally prove the schedulability of our partitioned scheduling algorithms. Our extensive experimental results demonstrate that the proposed algorithms can significantly improve the scheduling performance compared with the existing work.

1. 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 can significantly increase the computing performance while relaxing the power requirement over traditional single-core systems. Most of the major chip manufactures have already launched multi-core chips into the market, i.e. AMD *Opteron*™ *6300 Series* (AMD, 2013). It is not surprising that in the coming future, hundreds or even thousands of cores will be integrated into a single chip (Yeh et al., 2008). 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 major problem in the design of multi-core real-time system is how to utilize the available computing resources most efficiently while satisfying the timing constraints of all real-time tasks. To address this problem, one effective way is to develop an appropriate scheduling algorithm, which plays one of the most significant roles in real-time operating systems.

It is well known that the scheduling problem for multi-core systems is an NP-hard problem (Shin and Ramanathan, 1994). Although there exists optimal scheduling algorithms on single-core systems, i.e. *Rate Monotonic Scheduling (RMS)* and *Earliest Deadline First (EDF)* (Liu and Layland, 1973), none of them are optimal any more (Dhall and Liu, 1978) for multi-core systems. The reason is that, different from single-core scheduling, the multi-core scheduling needs to decide not only when but also where to execute a real-time task. Therefore, developing a sub-optimal heuristic for scheduling strategy on multi-core systems is reasonable and practical.

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

- We develop two new partitioned scheduling algorithms for fixed-priority periodic real-time tasks by taking the relationship among task periods into consideration. The first algorithm, namely *Partitioned Scheduling with Enhanced RBound (PSER)*, improves the traditional R-Bound by applying a more flexible task set scaling method (i.e. *TSS* method) and then partitions tasks under

☆ This work is supported in part by NSF under projects CNS-0969013, CNS-1423137, CAREER-0746643, CNS-1018731, CNS-0917021 and CNS-1018108.
* Corresponding author. Tel.: +1 3059151789.
E-mail addresses: mfan001@fiu.edu (M. Fan), qhan001@fiu.edu (Q. Han), sliu005@fiu.edu (S. Liu), sren@cs.fiu.edu (S. Ren), ganquan@fiu.edu (G. Quan), ren@iit.edu (S. Ren).

http://dx.doi.org/10.1016/j.jss.2014.09.010
0164-1212/

the enhanced utilization bound (i.e. $RBound^{en}$). The second one, namely *Harmonic Aware Partition Scheduling (HAPS)*, captures the "degree" of harmonic for a set of tasks with a novel harmonic metric (i.e. *harmonic index*) and then makes the partitioning decision such that tasks with closer harmonic relationship can be allocated to the same core.
- We analytically prove that both of our proposed partitioned scheduling algorithms, i.e. PSER and HAPS, can guarantee the schedulability of any task set that can successfully pass the partitioning procedure.
- We conduct extensive experiments to evaluate the performance of our proposed techniques. Through the experimental results, we can see that our proposed algorithms can significantly improve the scheduling performance compared with the existing works.

The rest of the paper is organized as follows. Section 2 introduces the work closely related to this paper. Section 3 describes the system models and Section 4 presents our motivational examples for this work. Section 5 and 6 present two partitioned scheduling algorithms we developed. Experiments and results are discussed in Section 7, and we conclude this work in Section 8.

2. Related work

In partitioned multi-core scheduling problem, the schedulability for tasks allocated on each processor can be determined based on feasibility conditions on single processors. To search for the optimal task partition for multiple processors is essentially a design space exploration problem, with complexity increasing rapidly with the size of the problem (e.g. the numbers of tasks or processors). How to quickly and accurately evaluate the schedulability of a design alternative (i.e. task partition) is key to the success of the partitioned multi-core scheduling problem. As a result, while there exists exact timing analysis method for feasibility checking for tasks on a single core platform (Liu and Layland, 1973; Kuo and Mok, 1991; Lauzac et al., 1998), they are not commonly used for partitioned multi-core scheduling problem due to their large computational complexity. Instead, many other timing-efficient feasibility checking methods, such as the utilization-bound based feasibility checking methods, are commonly used in the search for task partitions for multi-core scheduling problem.

2.1. Different utilization bounds for single-core systems

A *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 of 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 (denoted as $U(\Gamma)$) is no more than that parametric utilization bound, i.e. $U(\Gamma) \le f(\Gamma)$. Note that $U(\Gamma)$ can be calculated by summing up the task utilizations of all tasks in the task set Γ, where a task utilization is the ratio of its execution time over its period.

For single-core systems, there are several utilization bounds proposed under RMS policy (Liu and Layland, 1973; Kuo and Mok, 1991; Lauzac et al., 1998).

- *LLBound* (Liu and Layland, 1973): 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* (Kuo and Mok, 1991): 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 tasks in original task sets such that no two tasks are completely harmonic.
- *RBound* (Lauzac et al., 1998): 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 $1 \le r < 2$.
- *CBound* (Han and Tyan, 1997): The *CBound* is the utilization bound for a *harmonic task set*, in which the periods of any two tasks being integer multiple of each other, i.e.

$$CBound(\Gamma) = 1, \tag{4}$$

where Γ is a harmonic task set.

Among all four utilization bounds shown in the above, it has been proved that for RMS-based single-core scheduling, the *RBound* and *CBound* are higher than the other two (i.e. the *LLBound* and the *KBound*) (Lauzac et al., 1998; Han and Tyan, 1997). However, these two utilization bounds (*RBound* or *CBound*) have critical limitations. The *RBound* can only be applied when a given task set satisfies the period constraint (i.e. $1 \le r < 2$), while the *CBound* can only be used directly to harmonic task sets. Hence, in order to use the *RBound* or *CBound* for checking the schedulability of an arbitrary task set, we need to first transform the task set appropriately such that it satisfies the required condition.

For *RBound*, there are a few methods proposed to transform a task set to satisfy the condition of $1 \le r < 2$, such as Lauzac et al. (1998) and Kandhalu et al. (2012). In particular, Lauzac et al. (1998) 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

$$\begin{cases} C_i' = C_i \cdot 2^{\lfloor \log(T_{\max}/T_i) \rfloor} \\ T_i' = T_i \cdot 2^{\lfloor \log(T_{\max}/T_i) \rfloor} \end{cases} \tag{5}$$

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 is feasible then the original task set is also feasible.

Kandhalu et al. (2012) 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

$$\begin{cases} C_i' = C_i / \lfloor (T_i/T_{\min}) \rfloor \\ T_i' = T_i / \lfloor (T_i/T_{\min}) \rfloor \end{cases} \tag{6}$$

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 even when 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 Kandhalu et al. (2012), 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.

Please cite this article in press as: Fan, M., et al., Enhanced fixed-priority real-time scheduling on multi-core platforms by exploiting task period relationship. J. Syst. Software (2014), http://dx.doi.org/10.1016/j.jss.2014.09.010

For *CBound*, there are also a few methods proposed to transform a task set to satisfy the harmonic condition. Han et al. (1996), Han and Tyan (1997) proposed two methods, i.e. *Sr* and *DCT*, to transform a task set into a harmonic one. Since both methods result in the same harmonic task set, we only introduce the *DCT* method (which has a complexity equal to N^2) as below:

- Sort Γ by T with non-increasing order.
- For each $\tau_i \in \Gamma$, transform Γ to Γ_i' by

$$T_j' = \begin{cases} \dfrac{T_{j+1}'}{(\lfloor T_{j+1}'/T_j \rfloor)}, & \text{if } j < i \\ T_j, & \text{if } j = i \\ T_{j-1}' \cdot \left\lfloor \dfrac{T_j}{T_{j-1}'} \right\rfloor, & \text{if } j > i \end{cases} \tag{7}$$

- Find the optimal primary harmonic task Γ' that minimizes the total task set utilization among all Γ_i where $i = 1, 2, \ldots, N$. In other word, $U(\Gamma') = \min_{i=1}^{N} U(\Gamma_i')$.

The *RBound* and *CBound* indicate that on a single-core processor, the system utilization as well as the task set schedulability, can be greatly improved if the relationship between task periods can be appropriately exploited.

Existing work (i.e. Lauzac et al., 1998; Kandhalu et al., 2012; Han et al., 1996; Han and Tyan, 1997) has shown that, with appropriate task transformation, using *RBound* and *CBound* can significantly improve the schedulability checking accuracy.

2.2. Partitioned scheduling

Partitioned scheduling is originally derived based on the traditional *bin-packing* technique (Shin and Ramanathan, 1994). By mapping the utilization of a task to the "size" of the object and the utilization bound of a processor to the "capacity" of the bin, people can directly apply the common bin-packing strategies, i.e. *First-Fit* (FF), *Best-Fit* (BF) and *Worst-Fit* (WF), to deal with the partitioned multi-core problem. Coffman et al. (1997) presented several partitioned scheduling approaches derived from the traditional bin-packing strategies. For example, the FF approach assigns a task immediately to the first processor that can provide enough capacity for it, while the BF (WF) approach always assigns a task to the processor with the largest (smallest) total utilization that still can accommodate that task.

A few papers have been published on studying the problem of partitioned multi-core scheduling of fixed-priority periodic real-time tasks (Dhall and Liu, 1978; Burchard et al., 1995; Andersson et al., 2001; Darera and Jenkins, 2006; Andersson and Tovar, 2007; Fan and Quan, 2011; Fan et al., 2014). Burchard et al. (1995) evaluated the partitioned multi-core scheduling under RMS policy by exploiting the traditional bin-packing heuristics, such as FF, BF and WF, with a decreasing order of task utilizations. Andersson et al. (2001) developed a multi-core scheduling algorithm for fixed-priority periodic tasks, and proved that their proposed algorithm can guarantee the schedulability of any task set with system utilization no more than 1/3. They also showed that the utilization bound of fixed-priority multi-core scheduling (for both partitioned and global scheduling) was no more than 50% (Andersson et al., 2001; Andersson and Jonsson, 2003). Lopez et al. (2001, 2004) developed more accurate but complex utilization bounds for multi-core scheduling under RMS by combining the number of processors, the number of tasks and the maximum task utilization into consideration. Later, Darera and Jenkins (2006) developed a specific utilization bound for partitioned multi-core scheduling under the case of a greedy RMS-based algorithm. Andersson and Tovar (2007) introduced a new performance metric, named speed competitive ratio, to measure the performance of partitioned multi-core scheduling under RMS, and based on that new metric, they developed an algorithm with guaranteed schedulability under deterministic processor speedup. In what follows, we first introduce some basic concepts necessary to our work.

3. Preliminary

In this section, we first present our system models and some basic concepts, then we introduce two feasibility test methods, i.e. the RBound and CBound feasibility tests.

3.1. System models

We first introduce the multi-core platform and the task model used in this paper. The multi-core platform consists of M identical cores, $M \geq 2$, denoted as $\mathcal{P} = \{P_1, P_2, \ldots, P_M\}$. Our task model consists of N periodic tasks, represented as $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_N\}$. 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 $i < j$.

Next, we define three concepts, i.e. *task utilization*, *task set utilization* and *system utilization*, that are necessarily required to our work.

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

$$u_i = \frac{C_i}{T_i} \tag{8}$$

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

$$U(\Gamma) = \sum_{i=1}^{N} u_i \tag{9}$$

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 $U(\Gamma_k)$ represent the total utilization of all tasks assigned to P_k.

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

$$U_M(\Gamma) = \frac{U(\Gamma)}{M} \tag{10}$$

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

3.2. Two feasibility test methods

3.2.1. RBound feasibility test

The *RBound* (Lauzac et al., 1998), as what we have introduced in Section 2.1, 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 ((Lauzac et al., 1998)). *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*

$$\begin{aligned} C_i' &= C_i \cdot 2^{\lfloor \log(T_{\max}/T_i) \rfloor} \\ T_i' &= T_i \cdot 2^{\lfloor \log(T_{\max}/T_i) \rfloor} \end{aligned} \tag{11}$$

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

$$U(\Gamma) \leq RBound(\Gamma') \tag{12}$$

The *RBound*(*) is given by Eq. (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

Table 1
A task set with six real-time periodic tasks.

τ_i	C_i	T_i	u_i
1	1	4	0.25
2	2	8	0.25
3	3	10	0.30
4	8	16	0.50
5	8	20	0.40
6	12	40	0.30

predicts the schedulability of Γ by comparing its utilization with the value of RBound under Γ'.

3.2.2. CBound feasibility test

The *CBound* (Han and Tyan, 1997) is another efficient utilization bound to test the feasibility of periodic tasks by taking harmonic characteristic into consideration. The *CBound* feasibility test method is formally concluded in Theorem 7.

Theorem 2. *Given a task set Γ, let Γ' be a harmonic task set transformed from Γ by DCT method. Then Γ is schedulable on a single-core system under RMS if*

$$U(\Gamma') \leq 1 \tag{13}$$

Theorem 2 shows that by transforming a task set Γ into a harmonic task set Γ', we can easily predict the feasibility of Γ by check whether the utilization of Γ' is less than or equal to "1". Note that the *CBound* feasibility test is different from the *RBound* feasibility test in terms of the way of task set transformation, i.e. *CBound* test only scales the periods while *RBound* test scales both periods and execution times.

4. Motivational examples

Before presenting our approach in detail, we first use two examples to motivate our research. In the first example, we illustrate that exploiting the harmonic relationship can significantly improve the schedulability in multi-core scheduling. In the second example, we demonstrate that we can explore this property for tasks not strictly harmonic.

Consider a multi-core platform with two processors, i.e. $M = 2$, and a task set consisting of six tasks with parameters shown in Table 1. When scheduling those six tasks on two processors, it is not difficult to verify that none of the existing bin-packing heuristics (e.g. "first-fit", "best-fit" and "worst-fit") can successfully schedule the tasks listed in Table 1.

Note that, current bin-packing based approaches allocate real-time tasks solely based on their utilization factors and simply ignore other factors such as the task period, which can significantly affect the schedulability of a real-time task. For example, it is a well known fact (Kuo and Mok, 1991; Han and Tyan, 1997) that a harmonic task set, i.e. the tasks with periods being integer multiples of each other, can have a much higher schedulability than other non-harmonic task sets. If we take this factor into consideration and assign τ_1, τ_2 and τ_4 to one processor, and τ_3, τ_5 and τ_6 to another processor, as shown in Fig. 1(a), the task set in Table 1 can be perfectly scheduled on two processors.

Since tasks with ideal harmonic relationship have much higher feasibility on a single-core, one intuitive idea for partitioned multi-core scheduling would therefore be the one to group tasks with ideal harmonic relationship together and assign them to one processor. The question is what if tasks are not exactly harmonic. We use another example to illustrate this scenario.

We consider another example to schedule a task set consisting of four tasks as shown in Table 2 on two processors. Different from the first example, from Table 2, we can see that none of any

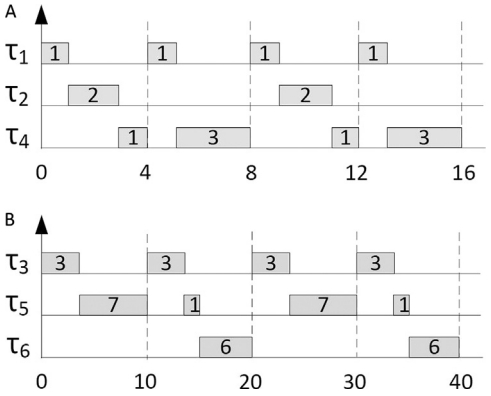


Fig. 1. Assign tasks in Table 1 based on ideal harmonic relationship, and all tasks can be scheduled successfully on two processors.

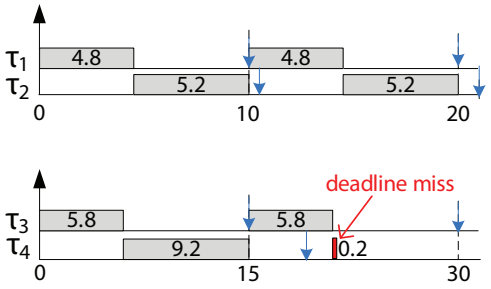


Fig. 2. Assign tasks in Table 2 based on pCOMPACTS (Kandhalu et al., 2012), while τ_4 missing its deadline.

two tasks are strictly harmonic. Thus, we cannot directly taking the harmonic advantage by assigning tasks according to the ideal harmonic relationship, i.e. the divisible period relationship. Once again, it is not difficult to verify that the traditional "first-fit" and "best-fit" approaches fail in satisfying the timing constraints for all four tasks. Even some most recent partitioning approach with harmonic awareness, i.e. *pCOMPACTS* approach (Kandhalu et al., 2012), cannot successfully guarantee the timing constraints in this example. *pCOMPACTS* measures the harmonic relationship by the distance between periods under the condition of $T_{max}/T_{min} < 2$, and thus first assigns τ_1 and τ_2 to the same processor and then leaves τ_3 and τ_4 running on another processor. This results in a failure of schedule for τ_4, see Fig. 2.

However, by assigning τ_1 and τ_4 to one processor and τ_2 and τ_3 to another processor, as shown in Fig. 3, we can build a feasible solution for all four tasks. As indicated by this example, although τ_1 and τ_4 have the largest period distance, i.e. $T_4 - T_1 = 19 - 10 = 9$, among all tasks, they are more harmonic. In fact, by assigning τ_1 and τ_4 to one processor, that local processor can achieve a much higher system utilization up to 0.97 (0.48 + 0.49), which is very close to the maximum ideal harmonic performance, i.e. 1.

From the above two motivational examples, we can observe that: (1) taking advantage of the harmonic relationship can utilize the system processing resource more efficiently; (2) exploiting the task period relationship can also improve the system utilization, even though how to quantify the harmonic relationship between

Table 2
A task set with four real-time periodic tasks.

τ_i	C_i	T_i	u_i
1	4.8	10	0.48
2	5.2	11	0.47
3	5.8	15	0.39
4	9.4	19	0.49

Please cite this article in press as: Fan, M., et al., Enhanced fixed-priority real-time scheduling on multi-core platforms by exploiting task period relationship. J. Syst. Software (2014), http://dx.doi.org/10.1016/j.jss.2014.09.010

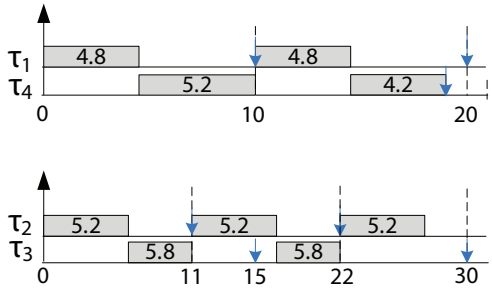


Fig. 3. Assign tasks in Table 2 based on closely harmonic relationship, and all tasks can be scheduled successfully on two processors.

tasks is a challenge. In what follows, we present our first partitioned scheduling algorithm based on an enhanced RBound.

5. Task partition with an enhanced RBound

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 (Lauzac et al., 1998). 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 (Lauzac et al., 1998) 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 (Kandhalu et al., 2012) 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 2.1), 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.

5.1. 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*(Γ, τ_k)

Require
(1) Γ: input task set, sorted with non-decreasing period order;
(2) τ_k: the kth 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 $Z_i = Z_{i-1} \cdot \lfloor \frac{T_i}{Z_{i-1}} \rfloor$ **end for**;
5: *// step 2: scale all tasks with respect to* τ_k;
6: **for** $i = 1$ to $k-1$ **do**
7: $R_i = 2^{\lfloor \log_2 \frac{T_k}{T_i} \rfloor}$
8: $T'_i = T_i \cdot R_i$
9: $C'_i = C_i \cdot R_i$
10: **end for**
11: **for** $i = k+1$ to N **do**
12: $R_i = Z_i / T_k$;
13: $T'_i = Z_i / R_i$;
14: $C'_i = C_i / R_i$;
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 $T_i \le T_j$ 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 (lines 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 (lines 11–15). Therefore, the timing complexity of TSS is $O(N)$, where N is the number of tasks in Γ. After all tasks in Γ are scaled appropriately, the corresponding task set Γ' is returned. In what follows, we discuss the property of TSS transformation, specifically the relationship between the transformed task set (Γ') and its original task set (Γ) in terms of schedulability.

5.2. Property of TSS transformation

In this subsection, we discuss the property of our proposed TSS transformation method. We show that if a transformed task set Γ' is schedulable under RMS, then its original task set Γ must be schedulable under RMS. This is essential to the application of our utilization bound in schedulability test.

Theorem 3. *Given a task set* $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_{N-1}, \tau_N\}$ *with* $T_1 \le T_2 \ldots \le T_N$, *let* $\Gamma^* = \{\tau_1, \tau_2, \ldots, \tau^*_q, \ldots, \tau^*_{N-1}, \tau^*_N\}$, *such that* $\forall j \in [q, N]$, $\tau^*_j = (C^*_j, T^*_j)$ *satisfies that*

$$C^*_j = k \cdot C_j, \quad T^*_j = k \cdot T_j, \tag{14}$$

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 $q-1$ tasks always have higher priorities than the q^{th} task in either Γ or Γ^*, their schedulability does not change. Thus, we only need to prove that $\forall j \in [q, N]$, τ^*_j is schedulable in Γ^*. Since Γ is schedulable, we know that the first instance of task τ_j must be able to meet its deadline. In other words, there must exist a time point t_x, where $t_x \in (0, T_j]$, such that

$$\sum_{i=1}^{q-1} C_i \cdot \left\lceil \frac{t_x}{T_i} \right\rceil + \sum_{i=q}^{j-1} C_i \cdot \left\lceil \frac{t_x}{T_i} \right\rceil + C_j \le t_x \tag{15}$$

Multiply both sides of the equation by k, the above equation can be rewritten as

$$\sum_{i=1}^{q-1} k \cdot C_i \cdot \left\lceil \frac{t_x}{T_i} \right\rceil + \sum_{i=q}^{j-1} C^*_i \cdot \left\lceil \frac{k \cdot t_x}{T^*_i} \right\rceil + C^*_j \le k \cdot t_x. \tag{16}$$

Since $\sum_{i=1}^{q-1} k \cdot C_i \cdot \left\lceil \frac{t_x}{T_i} \right\rceil \ge \sum_{i=1}^{q-1} C_i \cdot \left\lceil \frac{k \cdot t_x}{T_i} \right\rceil$ and let $t'_x = k \cdot t_x$, we have

$$\sum_{i=1}^{q-1} C_i \cdot \left\lceil \frac{t'_x}{T_i} \right\rceil + \sum_{i=q}^{j-1} C^*_i \cdot \left\lceil \frac{t'_x}{T^*_i} \right\rceil + C^*_j \le t'_x. \tag{17}$$

The above inequality means that at time point t'_k, τ^*_j as well as all other higher priority tasks can completely finish their execution requirements. Note that $t'_k \le k \cdot T_j = T^*_j$. Thus, τ^*_j 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*

$$1 \le \frac{T'_{\max}}{T'_{\min}} < 2 \tag{18}$$

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

$$\frac{T'_k}{T'_i} = \frac{T_k}{T_i \cdot 2^{\lfloor \log(T_k/T_i) \rfloor}} \tag{19}$$

from which we can derive that

$$1 = \frac{T_k}{T_i \cdot 2^{\log(T_k/T_i)}} \le \frac{T'_k}{T'_i} < \frac{T_k}{T_i \cdot 2^{(\log(T_k/T_i)-1)}} = \frac{2T_k}{T_i \cdot 2^{\log(T_k/T_i)}} = 2 \tag{20}$$

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

$$T'_i = \frac{Z_i}{Z_i/T_k} = T_k \tag{21}$$

Based on the above, we can immediately get

$$\frac{T'_k}{T'_i} = \frac{T'_k}{T_k} = 1 \tag{22}$$

where $i > k$. Thus far, we show $T'_k(T_k)$ is the maximum period in Γ', i.e. $\forall i$, $\frac{T'_k}{T'_i} \ge = 1$, 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 4.

Theorem 4. *Given a task set Γ, let Γ' be the scaled task set obtained by applying the TSS method with respect to an arbitrary task τ_k in Γ. If $U(\Gamma') \le RBound(\Gamma')$, then Γ must be schedulable on a single-core system under RMS.*

Proof. According to Lemma 1, we have that $\Gamma' = \{\tau'_1, \ldots, \tau'_{k-1}, \tau'_k, \tau'_{k+1}, \ldots, \tau'_N\}$ satisfies that $1 \le r < 2$, where r is the ratio between the maximum and minimum periods in Γ'. Thus, if $U(\Gamma') \le RBound(\Gamma')$, 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 1, we have that

$$T'_i = \frac{Z_i}{R_i} = \frac{Z_i}{(Z_i/T_k)} = T_k = T'_k \tag{23}$$

Thus, $\tau'_k, \tau'_{k+1}, \ldots, \tau'_N$ have the same as well as the lowest priority in Γ'. Moreover, $\tau'_1, \ldots, \tau'_{k-1}$ are tasks with priorities higher than τ'_k before as well as after the transformation. Based on Lemma 2 in work (Lauzac et al., 1998), if Γ' is schedulable, we know that the following task set $\widehat{\Gamma}'$ must be schedulable.

$$\widehat{\Gamma}' = \{\tau_1, \ldots, \tau_{k-1}, \tau_k, \tau'_{k+1}, \ldots, \tau'_N\} \tag{24}$$

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

$$\Gamma^* = \{\tau_1, \ldots, \tau_{k-1}, \tau_k, \tau^*_{k+1}, \ldots, \tau^*_N\} \tag{25}$$

For $i = k, \ldots, N-1$, we have that T^*_{i+1} is an integer multiple of T^*_i.i.e. there exists a perfect harmonic relationship between periods of T^*_k to T^*_N. In other words, Γ^* can be obtained through a series of transformation of $\widehat{\Gamma}'$ as descried in Theorem 3, therefore if $\widehat{\Gamma}'$ is schedulable, Γ^* must be schedulable.

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

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

5.3. 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 Eq. (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,

$$RBound^{en}(\Gamma) = \max_{\forall \tau_i \in \Gamma} \{RBound(\Gamma'_i) | \Gamma'_i = \mathrm{TSS}(\Gamma, \tau_i)\} \tag{26}$$

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

Next, in light of Theorem 4, 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 $U(\Gamma') \le RBound(\Gamma')$ is satisfied. The feasibility test method based on our $RBound^{en}$ is concluded with Theorem 5 in light of Theorem 1, Lemma 1 and Theorem 4.

Theorem 5. *Given a task set Γ, if $\exists \tau_i, \tau_i \in \Gamma$, such that*

$$U(\Gamma') \le RBound(\Gamma') \tag{27}$$

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

Theorem 5 provides a new feasibility test method by applying our proposed *TSS* method to obtain an enhanced RBound for a given task set. It is not surprising to see that our proposed feasibility test (given by Theorem 5) can always outperform the previous RBound feasibility test(Lauzac et al., 1998).

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 5.*

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

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

where Γ'_1 is obtained by using Eq. (5). 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

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

Please cite this article in press as: Fan, M., et al., Enhanced fixed-priority real-time scheduling on multi-core platforms by exploiting task period relationship. J. Syst. Software (2014), http://dx.doi.org/10.1016/j.jss.2014.09.010

According to Theorem 5, 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 enhanced RBound feasibility test given by Theorem 5. □

Corollary 1 shows that our proposed feasibility test method can always outperform the previous RBound feasibility test method. This is because the proposed feasibility test method completely covers the test cases in the traditional RBound feasibility test.

As an 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 $\Gamma'_2 = \{(7, 10), (1, 11), (1, 11)\}$. Hence we can derive that

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

According to Theorem 5, our proposed feasibility test can guarantee that Γ is schedulable on a single-core system under RMS.

5.4. The partitioning algorithm

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

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

Algorithm 2. Partitioned Scheduling with Enhanced RBound (PSER)

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

We show the details of *PSER* in Algorithm 2. During each iteration (with a total number of M iterations), 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 cannot 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 the tasks in Γ (i.e. line 6) with a complexity of $O(N)$ as discussed in Section 5.1. Following each transformation, two subroutines are performed, namely sorting the transformed task set (line 7) and tentatively grouping as many tasks as possible while satisfying the RBound constraint (line 9-12), whose timing complexities are $O(N \log N)$ (e.g. quick sort) and $O(N)$, respectively. By exploring all transformations with different initial conditions, we can optimize the grouping decisions and as a result, maximize the system utilization. Since we have at most N tasks in an unassigned task set Γ, the total complexity of the PSER algorithm is then $O(M \times N \times (N + N \log N + N))$. Dropping the low-order terms, this algorithm has a polynomial complexity of $O(MN^2 \log N)$.

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 6. *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, according to lines 9–17 in Algorithm 2, we know that there must exist $\tau_i \in \Gamma_m$, such that

$$U(\Gamma'_m) \leq RBound(\Gamma'_m) \tag{28}$$

where $\Gamma'_m = TSS(\Gamma, \tau_i)$. According to Theorem 5, 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. □

From Theorem 6, we can see that any task set successfully partitioned by *PSER* can be guaranteed to be schedulable under RMS on a multi-core system. In what follows, we will introduce another strategy for partitioned scheduling by exploring the harmonic advantage with *CBound*.

6. Harmonic advantage exploration with CBound

Instead of scaling each task with respect to both period and execution time, i.e. like TSS (shown in Algorithm 1), in this section, we introduce another approach that scales only the task periods. We first introduce a new metric, called "harmonic index" to quantify the harmonic characteristic among periodic tasks. Then based on that harmonic index, we present our second partitioned scheduling algorithm, i.e. *HAPS*, by taking the harmonic relationship into consideration to optimize the system utilization. Finally, we analyze the schedulability of our proposed algorithm *HAPS*.

6.1. Quantifying harmonic property

Since not all tasks in a given task set are harmonic, it is desirable that we can quantify the harmonicity of a task set. We first introduce the following two concepts.

Definition 1. Given a task set $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_N\}$ where $\tau_i = (C_i, T_i)$, let $\Gamma' = \{\tau'_1, \tau'_2, \ldots, \tau'_N\}$ where $\tau'_i = (C_i, T'_i)$, $T'_i \leq T_i$, and $T'_i | T'_j$ if $i < j$. (Note $a|b$ means "a divides b" or "b is an integer multiple of a".) Then Γ' is called a *sub-harmonic task set* of Γ.

Given a task set, there may be infinite numbers of different sub-harmonic task sets. There is one type of sub-harmonic task sets that is of most interest to us, which we call the *primary harmonic task set* and is formally defined as follows.

Please cite this article in press as: Fan, M., et al., Enhanced fixed-priority real-time scheduling on multi-core platforms by exploiting task period relationship. J. Syst. Software (2014), http://dx.doi.org/10.1016/j.jss.2014.09.010

Definition 2. Let Γ' be a sub-harmonic task set of Γ. Then Γ' is called a *primary harmonic task set* of Γ if there exists no other sub harmonic task set Γ'' such that $T'_i \le T''_i$ for all $1 \le i \le n$.

We are now ready to define a metric, i.e. the *harmonic index*, to measure the harmonicity of a real-time task set.

Definition 3. Given a task set Γ, let $\mathcal{G}(\Gamma)$ represent all the sub-harmonic task sets of Γ. Then the *harmonic index* of Γ, denoted as $\mathcal{H}(\Gamma)$, is defined as

$$\mathcal{H}(\Gamma) = \min_{\Gamma' \in \mathcal{G}(\Gamma)} \Delta U', \tag{29}$$

where $\Delta U' = U(\Gamma') - U(\Gamma)$

From Eq. (29), $\Delta U'$ defines the "distance" of a task set to the corresponding harmonic task sets in terms of its total utilization factor.

In this paper, we adopt the *DCT* algorithm (Han and Tyan, 1997) to find a primary harmonic task set for any given periodic task set. In the rest of this section, we will present our second partitioned scheduling algorithm *HAPS* by exploiting the harmonic metrics (i.e. harmonic index $\mathcal{H}$) to make our partitioning decision.

6.2. Harmonic Aware Partitioned Scheduling

In this subsection, we introduce our second partitioned scheduling algorithm, namely *Harmonic Aware Partitioned Scheduling* (*HAPS*). *HAPS* significantly distinguishes from *PSER*, as well as the bin-packing based scheduling approaches (i.e. First-Fit, Worst-Fit and Best-Fit). Instead of assigning tasks one by one, *HAPS* assigns tasks group by group in order to allocate as many tasks with closer harmonic relationship as possible to the same core.

The basic idea of *HAPS* can be briefly described as below:

- Among all unassigned tasks, for each task τ_i, construct a sub harmonic task set Γ' with respect of T_i.
- Pick up N_i tasks, denoted as Γ_{sub}, from higher harmonic relationship to lower harmonic relationship by maximizing $U(\Gamma_{sub})$ while keeping $U(\Gamma'_{sub}) \le 1$.
- Find the task group Γ_{opt} among unassigned tasks such that $U(\Gamma_{sub})$ is maximized.
- Allocate Γ_{opt} to an empty core.

Algorithm 3. Harmonic Aware Partitioned Scheduling (HAPS)

Require
(1) Task set: $\Gamma = \{\tau_1, \tau_2, \ldots \tau_N\}$;
(2) Multi-core system: $\mathcal{P} = \{P_1, P_2, \ldots, P_M\}$;
1: Sort Γ with no-decreasing order of task period;
2: **while** $\Gamma \neq \emptyset$ and $\mathcal{P} \neq \emptyset$
3: $\Gamma_{opt} = \emptyset$;
4: **for** $i = 1$ to $|\Gamma|$ **do**
5: $T'_i = T_i$
6: *for* $j = i+1$ to $|\Gamma|$ *do* $T'_j = T'_{j-1} \cdot \lfloor T_j / T'_{j-1} \rfloor$;
7: *for* $j = i-1$ downto 1 *do* $T'_j = \frac{T'_{j+1}}{\lfloor T'_{j+1}/T_j \rfloor}$;
8: Γ_{sub} = pick up N_i tasks from Γ such that
(1) $U(\Gamma'_{sub}) \le 1$, and $U(\Gamma'_{sub})$ is maximized;
(2) $H(\Gamma_{sub})$ is minimized;
9: **if** $U(\Gamma_{sub}) > U(\Gamma_{opt})$ **then**
10: $\Gamma_{opt} = \Gamma_{sub}$;
11: **end if**
12: **end for**
13: Pick up $P_m \in \mathcal{P}$, and assign Γ_{opt} to P_m;
14: $\Gamma = \Gamma \setminus \Gamma_{opt}$;
15: $\mathcal{P} = \mathcal{P} \setminus P_m$;
16: **end while**
17: **if** $\Gamma = \emptyset$ **then** return "success"; **else** return "fail", **endif**;

The *HAPS* is described in more details in Algorithm 3. Similar to *PSER*, we denote Γ as the task set containing all unassigned tasks and denote $\mathcal{P}$ as the core set containing all empty processors. We first sort Γ with non-decreasing order of task period (line 1). Then, when both Γ and $\mathcal{P}$ are not empty, we pick up a group of tasks with optimal combination, in terms of harmonic index and total utilization, and allocate them together to one empty core (from line 2 to line 16).

In each iteration of the "while" loop, we first initialize the objective subset of tasks as empty (line 3). The "for" loop (from line 4 to line 12) contains three steps: (1) transforming the task set Γ into a harmonic task set by using the T_i as the harmonic standard; (2) picking up a sub task set, denoted as Γ_{sub}, consisting of N_i tasks with higher harmonic relationship, meanwhile the corresponding task set utilization $U(\Gamma'_{sub})$ is maximized under the constraint of $U(\Gamma'_{sub}) \le 1$; (3) among all $|\Gamma|$ harmonic transformations, choosing the sub task set that has the maximum utilization in order to optimize the total system utilization. Note that the first step has a complexity of $O(N)$ as only one traverse of the entire task set is required. In addition, to accomplish step 2 (line 8), the transformed task set is first sorted (e.g. quick sort) in a non-decreasing order w.r.t. the harmonic index and group as many tasks as possible following the sorted order while ensuring the total utilization is less than or equal to 1. It is not hard to see that step (2) requires a total running time of $O(N \log N + N)$. Note that the second term N can be ignored since $N \log N$ is the dominant factor.

After finding the optimal group of tasks by the "for" loop, we assign that sub task set together to an empty core(line 13). Accordingly, we update the unassigned task set by removing the sub task set from Γ (line 14), and update the available processors by removing the occupied one from $\mathcal{P}$ (line 15). The algorithm succeeds if all tasks could be allocated, otherwise, it fails (line 17). Therefore, there is at most $\min(N, M)$ iterations for the "while" loop (line 2) as the algorithm terminates when either N tasks have been assigned (only one task is assigned in each iteration in the worst case) with excessive cores or total M core are occupied. Without loss of generality, we assume that $N > M$. Additionally, the "for" loop (line 4) is executed at most N times, therefore the HAPS algorithm also has a polynomial run-time complexity, i.e. $O(MN^2 \log N)$. In what follows, we conduct further feasibility analysis for this algorithm.

6.3. Schedulability analysis for HAPS

In this section, we discuss the schedulability of our proposed *HAPS* algorithm. We adopt the *RMS* policy on each core as the priority assignment criteria for all local tasks assigned on that local core. We prove that, after successfully partitioning all tasks by *HAPS*, the schedulability of all tasks can be guaranteed under *RMS*.

First, recall that in Section 6.1, we define the concept of primary harmonic task set, in which for any two tasks, the period of one can divide or be divided by the other. Then we introduce a feasibility test approach for real-time task set on single-core by checking its corresponding primary harmonic task set.

Theorem 7 ((Han and Tyan, 1997)). *Let Γ' be a primary harmonic task set of Γ. Then Γ is feasible on a single-core system under RMS if $U(\Gamma') \le 1$.*

From Theorem 7, we see that given a task set Γ and its corresponding primary harmonic task set Γ', if the utilization of Γ' is no greater than 1, then scheduling Γ on a single-core platform under *RMS*, all tasks can meet their deadlines.

Now we are ready to draw the conclusion of the feasibility of our proposed *HAPS* algorithm. We formally conclude this property in Theorem 8.

Theorem 8. *If a task set Γ is successfully partitioned by HAPS on M processors and scheduled under RMS, then all tasks can meet their deadlines.*

Please cite this article in press as: Fan, M., et al., Enhanced fixed-priority real-time scheduling on multi-core platforms by exploiting task period relationship. J. Syst. Software (2014), http://dx.doi.org/10.1016/j.jss.2014.09.010

Proof. Consider an arbitrary core P_m, let Γ_m be the task set assigned to P_m. Based on Algorithm 3, we know that all tasks in Γ_m are partitioned together at one time. Moreover, according to line 8 in Algorithm 3, there exists a primary harmonic task set of Γ_m, denote as Γ'_m, such that

$$U(\Gamma'_m) \leq 1$$

According to Theorem 7, we can see that Γ_m is feasible on core P_m under *RMS*. Consequently, the task set on each local core can be successfully scheduled. Therefore, all tasks in Γ can meet their deadlines. □

HAPS algorithm assigns all tasks group by group instead of one by one as the traditional bin-packing based partitioning algorithms (i.e. First-Fit, Best-Fit and Worst-Fit). Thus, *HAPS* can take the advantage of period relations and optimize task allocations. In the following section, we will conduct different experiments to evaluate the performance of our proposed scheduling algorithms in terms of task set schedulability and system utilization.

7. Experimental evaluations

In this section, we present a detailed discussion of our experimental evaluations for the proposed partitioned scheduling algorithms. We first introduce the experimental setup used in our evaluation. Then we present two groups of experiments to investigate the performance of our proposed techniques.

7.1. Experimental setup

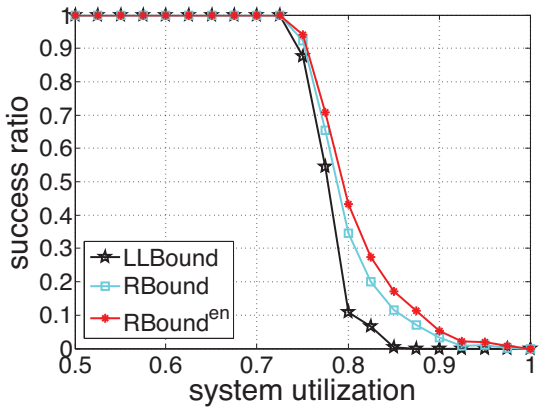
We conducted two sets of experiments to study the performance of our proposed enhanced utilization bound (in Section 5.3) and partitioned scheduling algorithms (in Sections 5.4 and 6.2), respectively. The scheduling performance for different approaches were compared by using the *success ratios*, i.e. the number of feasible tasks over the number of total tasks generated under a specific test point. In what follows, we respectively present the results of two group experiments.

7.2. Experiment 1: efficiency of our enhanced utilization bound

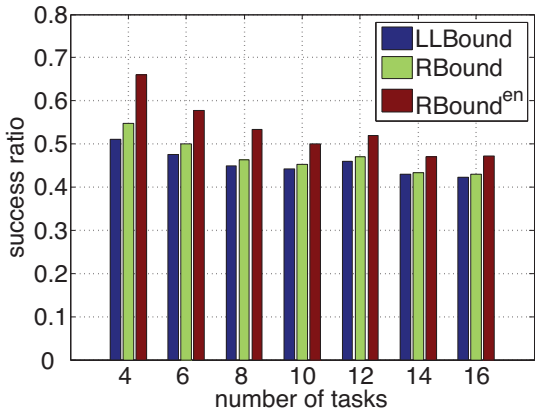
In this experiment, we evaluated the efficiency of our enhanced R-Bound (see Section 5.3) on a single-core platform. Three different utilization bounds were implemented:

- *LLBound* (Liu and Layland, 1973): Apply the *Liu & Layland's* utilization bound as shown in Eq. (1).
- *RBound* (Lauzac et al., 1998): Calculate the utilization bound by Eq. (3) under the traditional task set transformation method as given by Eq. (5).
- *RBound*^{en} (our proposed method): Calculate the utilization bound by Eq. (26).

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 Eq. (10)) 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 Fig. 4.



(a) Performance v.s. system utilization



(b) Performance v.s. number of tasks

Fig. 4. Efficiency of our enhanced utilization bound on a single core.

Fig. 4 shows the performance of three different utilization bounds with respect to system utilization and number of tasks, respectively. From Fig. 4(a) and (b), we can observe that our proposed *RBound*^{en} outperforms the others, i.e. *LLBound* and *RBound*. For example, in Fig. 4(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 Fig. 4(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*.

7.3. Experiment 2: performance of our partitioned scheduling algorithms

In this experiment, we studied the performance differences by different scheduling algorithms under different system utilizations. Five algorithms were implemented in this experiment.

- *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 Eq. (1)).
- *BF*: Partitions each task based on the Best-Fit (BF) bin-packing method (which assigns each task to the core with the smallest

Please cite this article in press as: Fan, M., et al., Enhanced fixed-priority real-time scheduling on multi-core platforms by exploiting task period relationship. J. Syst. Software (2014), http://dx.doi.org/10.1016/j.jss.2014.09.010

remaining capacity that can successfully accommodate that task), and checks the capacity of each local core with the *LLBound*.

- *RBOUNDMP*: Exploits the *RBound* with traditional task set scaling method (see Eq. (5)), and allocates each task based on the Best-Fit strategy under the *RBound*.
- *PSER*: *PSER* (our first proposed algorithm) scales the entire task set (including both periods and execution times of all tasks) with respect of each task's period, and then finds the maximal utilization bound among all scaled task sets, and further partitions each task based on the corresponding scaled task set meanwhile maximizes the total utilization of each local core.
- *HAPS*: *HAPS* (our second proposed algorithm) transforms the original task set into a harmonic counterpart by scaling and only scaling the periods of all tasks, and then based on our proposed harmonic index, assigns tasks with closer harmonic relationship into the same core to maximize the system utilization.

To study the performance differences among the above scheduling approaches with respect of system utilizations, we conducted two sub-sets of experiments, for light and general task sets, respectively. In light task sets, the utilization of each task was evenly distributed within [0, 0.5], while in general task sets, the utilization of each task was evenly distributed within [0, 1]. For each experiment, we varied the system utilization from 0.5 to 1.0 with an increment of 0.025. For both sub-sets of experiments, we tested on different number of processors, i.e. $M = 4$, 8, and 16. The experimental results for all approaches are collected and shown in Figs. 5 and 6.

Fig. 5 shows the experimental results for task sets containing only light tasks (i.e. $u_i \in [0, 0.5]$). From Fig. 5, we can observe that *PSER* and *HAPS* can achieve success ratios significantly better than other three approaches. Compared with *PSER* and *HAPS*, all other three approaches, i.e. *WF*, *BF* and *RBOUNDMP*, can guarantee the feasibility of any task set with utilization below *Liu&Layland's bound*, the same as *PSER* and *HAPS*. The success ratio by *WF* and *BF* drop sharply when system utilization is around 0.7. This is because that while *WF* and *BF* can guarantee task sets with utilizations no more than the *Liu&Layland's bound*, they reject any task set that cannot pass the feasibility checking condition determined by the *Liu&Layland's* approach. While *RBOUNDMP* may potentially schedule task sets with utilization higher than the *Liu&Layland's bound*, *PSER* and *HAPS* can achieve higher performance. For example, in Fig. 5(a), when the system utilization is around 0.85, *PSER* and *HAPS* can respectively achieve a success ratio up to 0.55 and 0.95, while that of *RBOUNDMP* is around 0.3. We can also see that the performance improvement by *PSER* and *HAPS* tends to increase as the number of processors increases. Under the system utilization of 0.9, *PSER* (*HAPS*) can achieve a success ratio of 0.05 (0.7) with 4 processors, 0.25 (0.95) with 8 processors, and increased up to 0.8 (1) with 16 processors.

Fig. 6 shows our 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 Fig. 6, we can also observe that our proposed algorithms, i.e. *PSER* and *HAPS*, perform better than other three approaches. In Fig. 6(b), when the system utilization is 0.85, *PSER* (*HAPS*) can achieve a success ratio 5 times (7 times) of that by *WF* and *BF*, and 1.25 times of that by *RBOUNDMP*.

It is important to observe that for both light and general task sets, our second proposed algorithm (*HAPS*) always outperforms our first proposed algorithm (*PSER*). For example, from Figs. 5(c) and 6(c), we can see that as the number of processors is fixed to 16, *HAPS* achieves better performance than *PSER* when system utilization is greater than 0.85 and 0.75 for light and general task sets, respectively. The reason is that *PSER* takes the harmonic advantage by only considering the relationship among periods of tasks (i.e. see task set scaling(Algorithm 1)), while *HAPS* takes both period and utilization of each task into consideration (i.e. see primary harmonic task set Definition (2) and harmonic index Definition (3)). Although by taking the harmonic relationship among periods of tasks can potentially increase the system utilization, we cannot consider the period factor isolated in order to optimize the system utilization, specifically for multi-core scheduling. Thus, to better improve the system performance of schedulability by taking the harmonic advantage, we need to appropriately consider not only the relationship among periods but also the utilizations of all tasks.

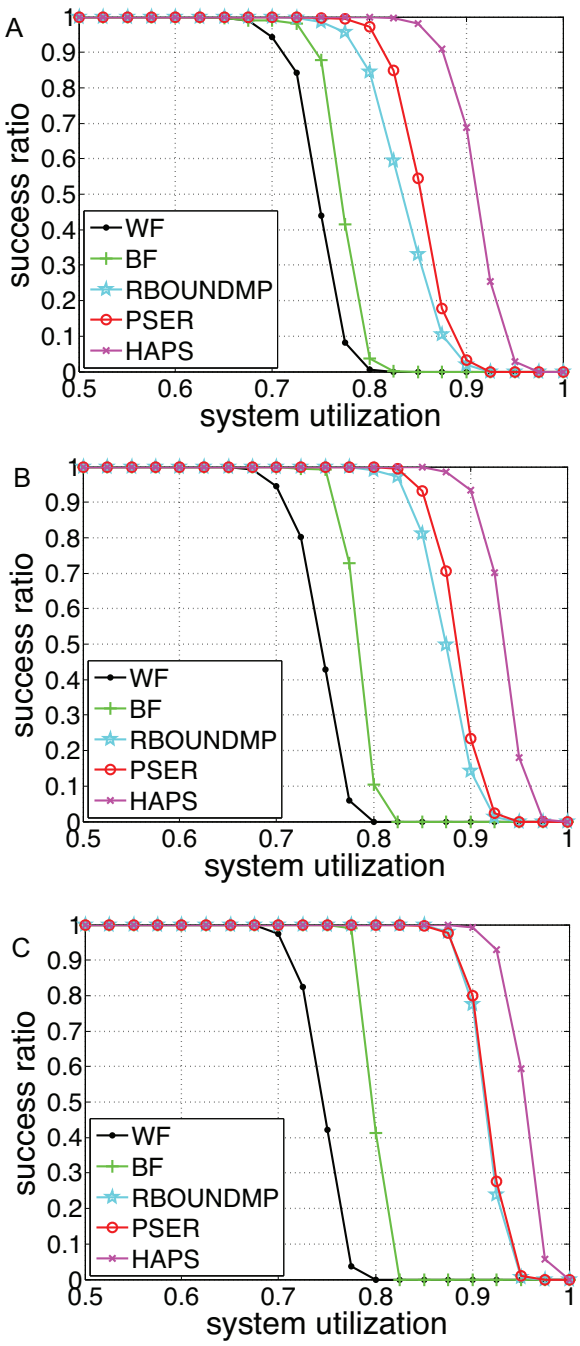


Fig. 5. Experimental results for light task sets ($u_i \in [0, 0.5]$) by different system utilization. (a) Number of processors: $M = 4$. (b) Number of processors: $M = 8$. (c) Number of processors: $M = 16$.

In summary, our experimental results clearly show that by exploiting the harmonic relationship among tasks appropriately, *PSER* and *HAPS* can significantly improve the schedulability of partitioned scheduling compared with the existing algorithms.

Please cite this article in press as: Fan, M., et al., Enhanced fixed-priority real-time scheduling on multi-core platforms by exploiting task period relationship. J. Syst. Software (2014), http://dx.doi.org/10.1016/j.jss.2014.09.010

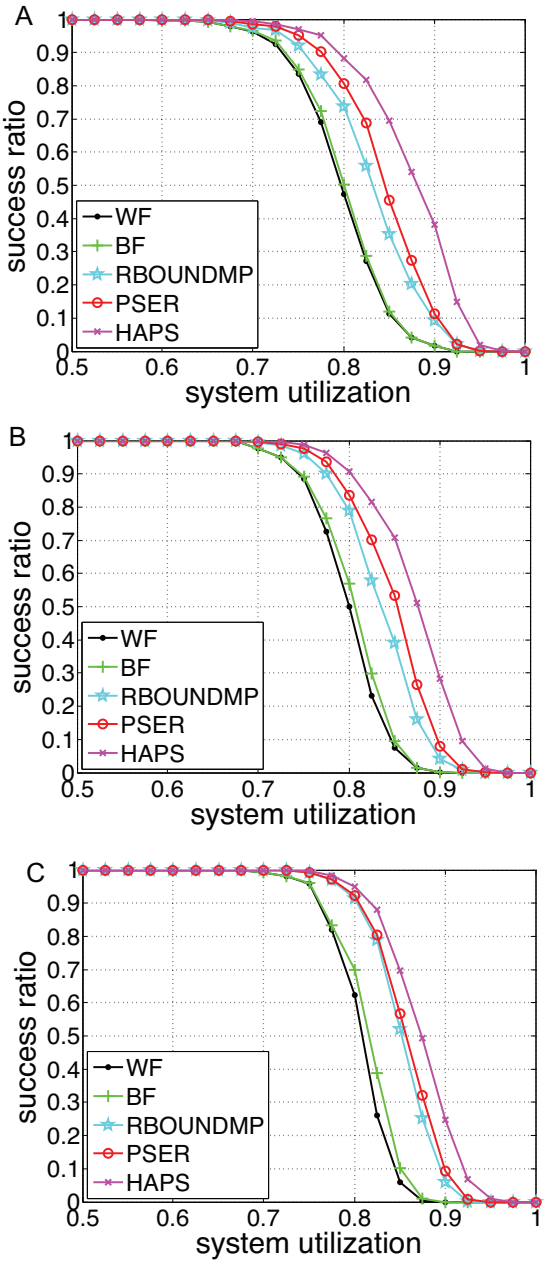


Fig. 6. Experimental results for general task sets ($u_i \in [0, 1]$) by different system utilization. (a) Number of processors: $M = 4$. (b) Number of processors: $M = 8$. (c) Number of processors: $M = 16$.

8. 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 have presented two new partitioned approaches (i.e. *PSER* and *HAPS*) for scheduling real-time sporadic tasks on multi-core platform under *RMS*. The *PSER* algorithm first transformed a given task set with respect to each task's period, and then assigned tasks based on their scaled periods under the traditional RBound. The *HAPS* algorithm took the harmonic advantage by transforming the entire task set into a harmonic set, and based on made the partitioning decision according to an efficient utilization bound, i.e. CBound. We formally proved that our scheduling algorithms could guarantee the feasibility of any task set successfully passed the partitioned procedures. Our extensive experimental results demonstrated that the proposed algorithm can significantly improve the scheduling performance compared with previous work.

References

AMD [link]. URL http://www.amd.com/US/PRODUCTS/SERVER/PROCESSORS/6000-SERIES-PLATFORM/6300/Pages/6300-series-processors.aspx

Andersson, B., Jonsson, J., 2003. The utilization bounds of partitioned and pfair staticpriority scheduling on multiprocessors are 50%. In: 15th Euromicro Conference on Real-Time Systems, 2003, pp. 33–40, http://dx.doi.org/10.1109/EMRTS.2003.1212725.

Andersson, B., Tovar, E., 2007. Competitive analysis of static-priority partitioned scheduling on uniform multiprocessors. In: 13th IEEE International Conference on Embedded and Real-time Computing Systems and Applications. RTCSA 2007, August, pp. 111–119, http://dx.doi.org/10.1109/RTCSA.2007.31.

Andersson, B., Baruah, S., Jonsson, J., 2001. Static-priority scheduling on multiprocessors. In: 22nd IEEE Real-Time Systems Symposium, 2001 (RTSS 2001), pp. 193–202, http://dx.doi.org/10.1109/REAL.2001.990610.

Burchard, A., Liebeherr, J., Oh, Y., Son, S., 1995. New strategies for assigning real-time tasks to multiprocessor systems. IEEE Trans. Comput. 44 (December (12)), 1429–1442, http://dx.doi.org/10.1109/12.477248.

Coffman Jr., E.G., Garey, M.R., Johnson, D.S., 1997. Approximation Algorithms for Bin Packing: A Survey. PWS Publishing Co., Boston, MA, USA, pp. 46–93 http://portal.acm.org/citation.cfm?id=241938.241940

Darera, V., Jenkins, L., 2006. Utilization bounds for RM scheduling on uniform multiprocessors. In: 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, pp. 315–321, http://dx.doi.org/10.1109/RTCSA.2006.63.

Dhall, S.K., Liu, C.L., 1978. On a real-time scheduling problem. Oper. Res. 26 (1), 127–140 http://www.jstor.org/stable/169896

Fan, M., Quan, G., 2011. Harmonic-fit partitioned scheduling for fixed-priority real-time tasks on the multiprocessor platform. In: IFIP 9th International Conference on Embedded and Ubiquitous Computing (EUC), October, pp. 27–32, http://dx.doi.org/10.1109/EUC.2011.41.

Fan, M., Han, Q., Quan, G., Ren, S., 2014. Multi-core partitioned scheduling for fixed-priority periodic real-time tasks with enhanced RBound. In: 2014 15th International Symposium on Quality Electronic Design (ISQED), pp. 284–291, http://dx.doi.org/10.1109/ISQED.2014.6783338.

Han, C.-C., Tyan, H.-Y., 1997. A better polynomial-time schedulability test for real-time fixed-priority scheduling algorithms. In: Proc. IEEE Real-Time Systems Symposium (RTSS), http://dx.doi.org/10.1109/REAL.1997.641267.

Han, C., Lin, K., Hou, C., 1996. Distance-constrained scheduling and its applications to real-time systems. IEEE Trans. Comput. 45 (7), 814–826, http://dx.doi.org/10.1109/12.508320.

Kandhalu, A., Lakshmanan, K., Kim, J., Rajkumar, R., 2012. pCOMPATS: period-compatible task allocation and splitting on multi-core processors. In: IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), http://dx.doi.org/10.1109/RTAS.2012.18.

Kuo, T.-W., Mok, A., 1991. Load adjustment in adaptive real-time systems. In: Proc. Real-Time Systems Symposium, 1991, Twelfth, pp. 160–170, http://dx.doi.org/10.1109/REAL.1991.160369.

Lauzac, S., Melhem, R., Mossè, D., 1998. An efficient RMS admission control and its application to multiprocessor scheduling. In: IPPS/SPDP Parallel Processing Symposium, http://dx.doi.org/10.1109/IPPS.1998.669964.

Liu, C.L., Layland, J.W., 1973. Scheduling algorithms for multiprogramming in a hard-real-time environment. J. ACM 20, 46–61, http://dx.doi.org/10.1145/321738.321743.

Lopez, J., Diaz, J., Garcia, D., 2001. Minimum and maximum utilization bounds for multiprocessor RM scheduling. In: 13th Euromicro Conference on Real-Time Systems, pp. 67–75, http://dx.doi.org/10.1109/EMRTS.2001.934003.

Lopez, J., Diaz, J., Garcia, D., 2004. Minimum and maximum utilization bounds for multiprocessor rate monotonic scheduling. IEEE Trans. Parallel and Distributed Systems 15 (July (7)), 642–653, http://dx.doi.org/10.1109/TPDS.2004.25.

Shin, K., Ramanathan, P., 1994. Real-time computing: a new discipline of computer science and engineering. Proc. IEEE 82 (1), 6–24, http://dx.doi.org/10.1109/5.259423.

Yeh, D., Peh, L.-S., Borkar, S., Darringer, J., Agarwal, A., Hwu, W., 2008. Thousand-core chips [roundtable]. IEEE Des. Test Comput. 25 (3), 272–278, http://dx.doi.org/10.1109/MDT.2008.85.

Dr. Ming Fan received his Ph.D. from the Department of Electrical and Computer Engineering at Florida International University, Florida, USA, in 2014. He received both B.S. and M.S. from the Department of Software Engineering at Beihang University, Beijing, China, in 2006 and 2009, respectively. His research interests include real-time systems, power-/thermal-aware computing and fault-tolerant system design.

Qiushi Han is a Ph.D. candidate in the Department of Electrical and Computer Engineering at the Florida International University, Florida, USA. He received his B.S. from the Department of Software Engineering, Beijing Jiaotong University. His

Please cite this article in press as: Fan, M., et al., Enhanced fixed-priority real-time scheduling on multi-core platforms by exploiting task period relationship. J. Syst. Software (2014), http://dx.doi.org/10.1016/j.jss.2014.09.010

research interests include real-time systems, power-/thermal- aware computing and reliable/fault-tolerant system designs.

Shuo Liu is a Ph.D. candidate in the Department of Electrical and Computer Engineering at Florida International University, Florida, USA. He received his M.S. from the Department of Electrical Engineering at Utah State University, Utah, USA, in 2009, and his B.S from the Department of Electrical Engineering at Beihang University, Beijing, China, in 2006. His research interests include real-time systems, parallel and distributed systems and cloud computing.

Dr. Shaolei Ren received the B.E. degree in Electronic Engineering from Tsinghua University in July 2006, the M.Phil. degree in Eletronic and Computer Engineering from Hong Kong University of Science and Technology in August 2008, and the Ph.D. degree in Electrical Engineering from University of California, Los Angeles, in June 2012. Since August 2012, he has been with Florida International University, where he currently holds a joint appointment of Assistant Professor in the School of Computing and Information Sciences and the Department of Electrical and Computer Engineering. His research centers around cloud computing and data center resource management, with an emphasis on sustainability.

Dr. Gang Quan received his Ph.D. from the Department of Computer Science & Engineering, University of Notre Dame, USA, his M.S. from the Chinese Academy of Sciences, Beijing, China, and his B.S. from the Department of Electronic Engineering, Tsinghua University, Beijing, China. He is currently an associate professor in the Electrical and Computer Engineering Department, Florida International University. Before he joined the department, he was an assistant professor at the Department of Computer Science and Engineering, University of South Carolina. His research interests and expertise include real-time systems, embedded system design, power-/thermal-aware computing, advanced computer architecture and reconfigurable computing. Dr. Quan is the recipient of a National Science Foundation Faculty Career Award. He also won the Best Paper Award from the 38th Design Automation Conference. His paper was also selected as one of the Most Influential Papers of 10 Years Design, Automation, and Test in Europe Conference (DATE) in 2007. Dr. Quan is a senior member of IEEE.

Dr. Shangping Ren is an Associate Professor at the Department of Computer Science, Illinois Institute of Technology. She obtained her Ph.D. in CS from UIUC in 1997. Before she joined IIT in 2003, she worked as a software engineer in software industry for six years. Her main research focus is in the area of software development of time critical distributed systems, including software architecture, system reliability under resource constraints, scheduling algorithms for meeting reliability and deadline constraints that are critical in time critical systems, such as military command control systems, automotive systems, and some types of health care systems.

Please cite this article in press as: Fan, M., et al., Enhanced fixed-priority real-time scheduling on multi-core platforms by exploiting task period relationship. J. Syst. Software (2014), http://dx.doi.org/10.1016/j.jss.2014.09.010