Harmonic Semi-Partitioned Scheduling For Fixed-Priority Real-Time Tasks On Multi-Core Platform

Ming Fan Gang Quan
Electrical and Computer Engineering Department
Florida International University
Miami, FL, 33174
{mfan001, gaquan}@fiu.edu

***Abstract*—This paper presents a new semi-partitioned approach to schedule sporadic tasks on multi-core platform based on the *Rate Monotonic Scheduling* (RMS) policy. Our approach exploits the well known fact that harmonic tasks have better schedulablility than non-harmonic ones on a single processor. The challenge for our approach, however, is how to take advantage of this fact to assign and split appropriate tasks on different processors in the semi-partitioned approach. We formally prove that our scheduling approach can successfully schedule any task sets with system utilizations bounded by the Liu&Layland's bound. Our extensive experiment results demonstrate that the proposed algorithm can significantly improve the scheduling performance compared with the previous work.**

I. Introduction

As embedded applications become more and more complicated, embedded system designers rely more on multi-core platforms to obtain high computing performance [1], [2]. Meanwhile, due to the power/thermal constraints, the industry is changing its gear toward the multi-core architecture rather than continuing to pursue high performance under single processor architecture. One major issue in software developments for multi-core architecture is how to utilize the available computing resources most effectively. To this end, we study the problem of scheduling real-time tasks on multi-core architecture based on the *Rate Monotonic Scheduling* (RMS) policy, i.e. the most commonly used scheduling scheme for real-time systems [3].

There have been extensive researches published on real-time scheduling for homogeneous multi-core systems [4], [5], [6], [7]. These scheduling algorithms can be largely categorized into two classes [8]: the *partitioned* approach (e.g. [4]) and the *global (or non-partitioned)* approach (e.g. [5]). In the partitioned scheduling, each real-time task is assigned to a dedicated processor. All instances from the same task will be executed solely on that particular processor. In global scheduling, all jobs first enter a global queue, and thus each task can be potentially executed on any processor. Both approaches have their own pros and cons, and none of them dominates the other in terms of schedulability [8].

978-3-9810801-8-6/DATE12/

Recently, a new multi-core scheduling approach, i.e. so called *semi-partitioned* approach [9], [10], [11], [7], [6], [12], [13] has been proposed. In semi-partitioned scheduling, most tasks are assigned to one particular processor while a few of tasks are allowed to be split into several subtasks and assigned to different processors. From a different perspective, these tasks can migrate among different processors. By splitting tasks, the overall system utilization can be significantly improved. For example, the best known utilization bound for either global or partitioned fixed-priority schedule is no more than 50% [4], [14], [5]. For the semi-partitioned scheduling, Lakshmanan *et al.* [7] have shown an utilization bound of 65%, and Guan *et al.* [12], [15] improved this bound to the traditional Liu&Layland's bound [16], i.e. 69.3%.

In this paper, we present a new semi-partitioned strategy and related feasibility analysis for sporadic tasks on the multi-core platform based on RMS. Compared with the existing work on semi-partitioning of real-time tasks, we have made a number of novel contributions:

- First, we take the harmonic relation among tasks into consideration and develop two new semi-partitioned algorithms. The first algorithm, namely *HSP-light*, is intended for task sets with utilization factor of each task no more than $1/2$. The second one, namely *HSP*, is developed for more general task sets, i.e. the utilization factor of each task is no more than 1.
- Secondly, we present new feasibility analysis results for the semi-partitioned scheduling algorithms developed in this paper. We formally prove that the proposed algorithms can guarantee the feasibility for task sets with utilizations no larger than the Liu&Layland's bound. Moreover, different from the approach in [12], task sets with utilizations higher than the Liu&Layland's bound may also be schedulable with our approaches.
- Thirdly, we conducted extensive experiments to study the performance of our approach, and our experimental results demonstrate that our proposed algorithms can significantly outperform the previous work.

The rest of the paper is organized as follows. Section II describes system models and other background information necessary for this paper. Section III and IV present two semi-

partitioned algorithms we developed. Experiments and results are discussed in Section V, and we present the conclusions in Section VI.

II. Preliminary

In this section, we first introduce the system models used in this paper, and then we introduce some background information and concepts necessary for our research. We then use an example to motivate our research.

A. System models

The real-time system considered in this paper consists of N sporadic tasks, denoted as $\Gamma = \{\tau_1, \tau_2, ..., \tau_N\}$, and executed on M identical processors, i.e. $\mathcal{P} = \{P_1, P_2, ..., P_M\}$. Each task $\tau_i \in \Gamma$, is characterized by a tuple (C_i, T_i), where C_i is the *worst-case execution time* of τ_i, and T_i is the *minimum inter-arrival time (period)* between any two consecutive jobs of τ_i.

For the rest of this paper, we make two assumptions: 1) the deadline of each task is equal to its period; 2) Γ is sorted with decreasing priority order according to RMS, i.e. task τ_i has a higher priority than τ_j if $i < j$. For the sake of simplicity, let Γ_{P_m} denote the task set on processor P_m, and $P(\tau_i)$ denote the host processor of τ_i.

The *utilization factor* of a task τ_i is represented as $u_i = \frac{C_i}{T_i}$. The *total utilization of a task set* Γ is represented as $U(\Gamma) = \sum_{\tau_i \in \Gamma} u_i$. The *system utilization* of task set Γ on a multi-core platform with M processors is represented as $U_M(\Gamma) = \frac{U(\Gamma)}{M}$. Based on its utilization factor, a task can be *light* or *heavy*, which are formally defined below:

Definition 1: Task τ_i is called a *light task* if $u_i \leq \frac{1}{2}$, or a *heavy task* otherwise.

Note that, even though we used the same terminology as that in [12], our definitions of light and heavy tasks are totally different from those in the previous work.

Liu and Layland [16] showed that a task set Γ can be feasibly scheduled by RMS on a single processor if

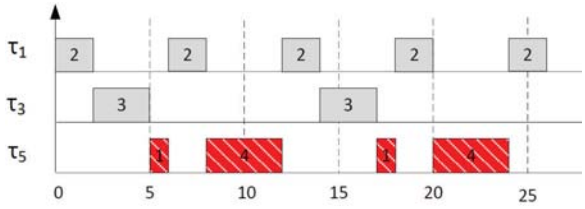
$$U(\Gamma) \leq \Theta(N) = N(2^{1/N} - 1). \quad (1)$$

$\Theta(N)$ is also traditionally referred to as the *Liu&Layland bound*.

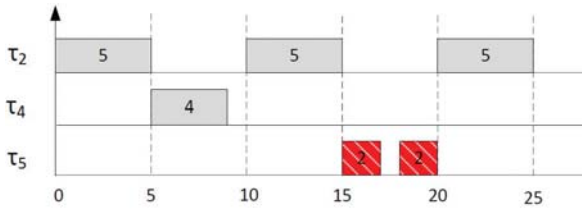
In this paper, we adopt the same terms, i.e. *non-split task* and *split task*, as that in [12] to present our proposed algorithms. Specifically, *non-split tasks* are the ones assigned to one processor and executed only on that particular processor during run time. *Split tasks* are the ones split into several *subtasks* and assigned to different processors. The last subtask of a split task τ_i is called the *tail task*, denoted as τ_i^t, and other subtasks of τ_i are called *body tasks*, denoted as $\tau_i^{b_j}$, where $j \in [1, B]$ and $B \geq 1$. We also assume that an appropriate timer is available to monitor the execution of body tasks. All body tasks from the same task can only run sequentially following their logical orders to ensure the correctness of the program. Therefore, the body/tail tasks from the same task can be viewed as tasks with the same periods but different deadlines. Moreover, we use C_i^B and u_i^B to represent the total execution time and utilization of all body tasks of τ_i, respectively.

TABLE I
Another task set with five real-time tasks

τ_i	C_i	T_i	u_i
1	2	6	0.33
2	5	10	0.50
3	3	12	0.25
4	4	20	0.20
5	15	25	0.60



(a) Processor 1



(b) Processor 2

Fig. 1. Allocation fails when simply grouping harmonic tasks and assigning them to the same processor.

B. Motivation example

As a well known fact, a harmonic task set, i.e. the tasks with periods being integer multiples of each other, can utilize the system more efficiently than other non-harmonic task sets on a single processor [17], [18]. Thus, an intuitive approach would therefore be the one that groups harmonic tasks together and assigns them to the same processor. Unfortunately, such a naive approach may not work in semi-partitioned scheduling.

Consider a two-processor platform with a task set shown in Table I. Since τ_1 and τ_3 are harmonic, we can group τ_1 and τ_3 to one processor, i.e. Processor 1. Similarly, we can group τ_2 and τ_4 to the other processor, i.e. Processor 2. Since no processor can accommodate τ_5 entirely, we have to split τ_5 between these two processors. There are two problems with this assignment. First, as shown in Figure 1(a), the maximum capacity in terms of execution time that can be accommodated in Processor 1 is 10. Since the subtasks from τ_5 cannot be executed concurrently on two processors, at most 4 time units from Processor 2 can be utilized by τ_5 as shown in Figure 1(b). As a result, τ_5 cannot complete before its deadline even if all available time units could be used for its execution. Second, in order to use all 4 time units on Processor 2, we need complicated process migration controls and synchronization mechanisms, which increase not only the the switching overhead, but also the control complexity among different processors. Note that, if we assign τ_1 and τ_5 to one processor, and the other tasks to another processor, it is not difficult to verify that the schedule is feasible.

As indicated by this example, to take the advantage of

harmonic relationship among tasks to improve the feasibility in semi-partitioned approach, a critical problem is how to judiciously choose the task to split and to synchronize among different processors. To solve this problem, we present two novel semi-partitioned algorithms, i.e. *HSP-light* and *HSP*, in the following.

III. The HSP-light algorithm

The *HSP-light* algorithm is a harmonic semi-partitioned algorithm developed for light tasks. When employing the harmonic relationship to improve the scheduling performance, it is not necessary that all tasks are strictly harmonic. To this end, we first introduce a metric, namely the *harmonic index*, to quantify the degree of harmonic for a task set. We then discuss our new algorithm that employs this metric. Finally, we give the feasibility analysis.

A. Quantifying the harmonicity

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 2: Given a task set $\Gamma = \{\tau_1, \tau_2, ..., \tau_N\}$, let $\Gamma' = \{\tau'_1, \tau'_2, ..., \tau'_N\}$ 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 more than one 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.

Definition 3: 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 $T'_i \leq T''_i$ for all $1 \leq i \leq N$.

Moreover, for any sub harmonic task set of Γ, let $\Delta U'$ represent as

$$\Delta U' = \begin{cases} U(\Gamma') - U(\Gamma), & \text{if } U(\Gamma') \leq 1, \\ +\infty, & \text{otherwise.} \end{cases} \quad (2)$$

From equation (2), $\Delta U'$ defines the "distance" of a task set to the corresponding sub harmonic task set in terms of its total utilization factor. If the utilization of that sub harmonic task set is greater than 1, then the "distance" is set to be infinity.

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

Definition 4: Given a task set Γ, let $\mathcal{G}(\Gamma)$ represent all the primary harmonic task sets of Γ. Then the *harmonic index* of Γ, denoted as $\mathcal{H}(\Gamma)$, is defined as

$$\mathcal{H}(\Gamma) = \min_{\Gamma' \in \mathcal{G}(\Gamma)} \Delta U' \quad (3)$$

It is worthy of mentioning that, when given a real-time task set, the *Sr* or *DCT* algorithm [19], [17] can be employed to find primary harmonic task sets with a complexity as low as $N\log(N)$. For a real-time task set and its primary harmonic task sets, it is not difficult to prove the following theorem [17].

Theorem 1: [17] Let Γ' be a primary harmonic task set of Γ. Then Γ is feasible on single processor under RMS if $U(\Gamma') \leq 1$.

B. Algorithm details

HSP-light algorithm assigns tasks to processors from lower priority to higher priority ones. A task is assigned to a processor such that processor can accommodate it entirely and also the result task set has the lowest harmonic index. When a task cannot be put in any processor as a whole, it is split with its subtask assigned to the processor with the highest available capacity to accommodate the task. Algorithm 1 shows the salient aspects of the HSP-light algorithm. For the HSP-light algorithm, we shall first ignore the *Heavy-Task-Preassignment* function (line 4) and other heavy task related statements (from line 4 to line 7).

Algorithm 1 HSP-light Algorithm

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

1: *//$\mathcal{P}^{Pre}$ = Heavy-Task-Preassignment(Γ,$\mathcal{P}$);*
2: **while** $\Gamma \neq \emptyset$ **do**
3: $\tau_i :=$ the task with the lowest priority in Γ;
4: *// $\tau_j :=$ the task with the lowest priority in $\Gamma_{\mathcal{P}^{Pre}}$;*
5: *// **if** τ_i has higher priority than τ_j **then***
6: *// Move $P(\tau_j)$ from $\mathcal{P}^{Pre}$ to $\mathcal{P}$;*
7: *// **end if***
8: $P_m :=$ the processor with minimum $\mathcal{H}(\Gamma_{P_m} + \tau_i)$ in $\mathcal{P}$;
9: **if** $\Gamma_{P_m} + \tau_i$ is feasible **then**
10: Assign τ_i to processor P_m;
11: Continue;
12: **end if**
13: $P_m :=$ the processor with maximum capacity for τ_i in $\mathcal{P}$;
14: **if** $P_m = \emptyset$, **then** Break, **end if**
15: **if** $\Gamma_{P_m} + \tau_i$ is feasible **then**
16: Assign τ_i to processor P_m;
17: **else**
18: Split τ_i into τ_{i1} and τ_{i2} such that $\Gamma_{P_m} + \tau_{i1}$ can fully utilize P_m;
19: Assign τ_{i1} to processor P_m;
20: Replace τ_i by τ_{i2}, and move τ_i back to Γ;
21: **end if**
22: **end while**
23: **if** $\Gamma = \emptyset$ **then** return "success"; **else** return "fail", **end if**

From Algorithm 1, it is easy to derive the following property.

Lemma 1: If a task set Γ is successfully partitioned by HSP-light on M processors, then there is at most one *body task* on each processor; and on all processors, there are at most $(M-1)$ tasks to be split.

Lemma 1 constrains the maximal number of tasks that needs to be split and migrated among different processors, and thus the extra cost associated with the migrations. More importantly, if a task set can be successfully allocated by HSP-light, all tasks can satisfy their deadlines. The conclusion is formally formulated in the following theorem.

Theorem 2: If a task set Γ is successfully partitioned by HSP-light on M processors and scheduled according to RMS, then all tasks can meet their deadlines.

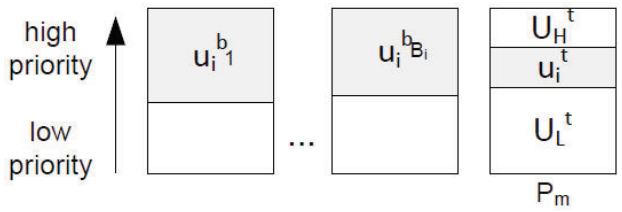


Fig. 2. Illustration of U_H^t and U_L^t

The theorem can be proved by noting that whenever a task is assigned to a processor, the feasibility of the task set on that processor is guaranteed based on the exact timing analysis method [20], i.e. line 9, line 15 and line 18.

C. Feasibility test analysis of HSP-light

In this subsection, we develop a more effective feasibility test method for HSP-light. We formally prove that any light task set with utilization no larger than the well-known Liu&Layland's bound, can be successfully partitioned by HSP-light.

Before discussing the feasibility of HSP-light, we first introduce the *critical task* concept:

Definition 5: Let $\Gamma = \{\tau_1, ..., \tau_i, ..., \tau_N\}$ be a task set that is schedulable by RMS on a single processor. τ_i is called the *critical task* if when increasing the execution time of the highest priority task τ_1, τ_i is the first task to miss its deadline.

There are only three type of tasks in the semi-partitioned system, i.e. non-split task, body task and tail task. Since any body task always has the highest priority on its host processor, from Definition 5, we know body tasks can not be the critical tasks. Thus the critical task on each processor can only be a non-split task or a tail task. In what follows, we want to study the feasibility characteristic for processors containing non-split or tail tasks as critical tasks. In the rest of this paper, we assume that any split task τ_i is split into B_i body tasks and one tail task, denoted as $\tau_i^{b_j}$ ($j \in [1, B_i]$) and τ_i^t, respectively.

For two different type of critical tasks, i.e. non-split tasks and tail tasks, we introduce two important properties, which are formulated in the following lemmas.

Lemma 2: Let Γ_{P_m} be the task set allocated to processor P_m in HSP-light. If the critical task is a non-split task and P_m is fully utilized, then $U(\Gamma_{P_m}) > \Theta(N)$.

Lemma 2 can be easily proved by contradiction.

Lemma 3: Let Γ_{P_m} be the task set allocated to processor P_m in HSP-light. If the critical task is a tail task and P_m is fully utilized, then $U(\Gamma_{P_m}) > \Theta(N)$.

Proof sketch: Let U_L^t (U_H^t) denote the total utilization of tasks with priorities lower (higher) than τ_i on P_m, i.e. see Figure 2. Since $\tau_i^{b_1}$'s host processor has the largest capacity to accommodate τ_i, thus

$$U_L^t + u_i^{b_1} \geq \Theta(N). \tag{4}$$

On the other hand, since τ_i^t is the critical task on its host processor P_m, we have

$$\sum_{j<i} C_j \lceil \frac{T_i - C_i^B}{T_j} \rceil + C_i^t \geq T_i - C_i^B$$

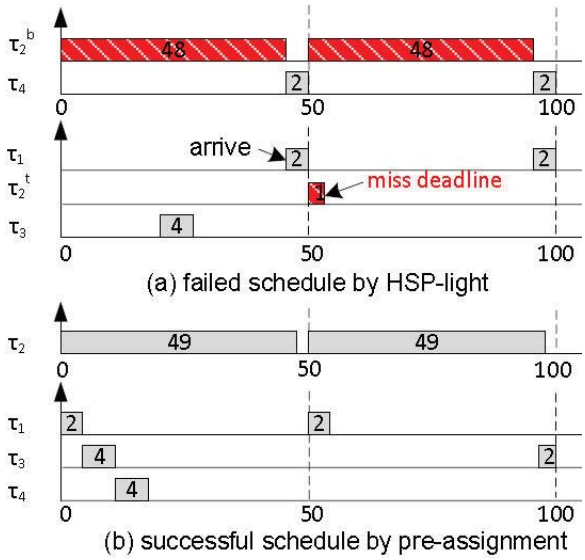


Fig. 3. Schedule task set with heavy task.

Divide $(T_i - C_i^B)$ on both side of the above, then apply $\frac{T_j}{T_i - C_i^B} < \frac{T_i}{T_i - C_i^B} \leq 2$, we can derive that

$$U_H^t + u_i^t > \frac{1}{2} \tag{5}$$

Finally, sum up equation (4) and (5), and replace $(U_L^t + U_H^t + u_i^t)$ by $U(\Gamma_{P_m})$, we can obtain that

$$U(\Gamma_{P_m}) > \Theta(N)$$

□

Based on Lemma 2 and Lemma 3, we can derive the following property.

Theorem 3: Given a light task set Γ consisting of N tasks to be scheduled on M processors, if

$$U_M(\Gamma) \leq \Theta(N), \tag{6}$$

then Γ is feasible by HSP-light under RMS.

Theorem 3 can be proved by contradiction based on Lemma 2 and Lemma 3. The detailed proof is omitted due to page limit.

Note that while Theorem 2 is valid for any general task set, Theorem 3 works only for light task sets. To deal with the general task sets, we develop a more advanced algorithm (*HSP*) in the next section.

IV. The HSP Algorithm

The reason that HSP-light cannot guarantee the feasibility of an arbitrary task set with utilization lower than the Liu&Layland's bound is that, if a split task is a heavy task and the tail task is very *light*, the overall system utilization can be very low. We use an example to explain this observation.

Consider a task set with four tasks, $\tau_1 = (2,50), \tau_2 = (49,50), \tau_3 = (4,90), \tau_4 = (4,100)$, to be scheduled on 2 processors. As shown in Figure 3(a), even though the system utilization is very small, i.e. $(2/50 + 49/50 + 4/90 + 4/100)/2 = 0.55 < 0.69$, HSP-light cannot schedule this task set successfully. Note that the tail task from τ_2 can be viewed as a

task with worst case execution time of 1 and deadline of 2. Adding any higher priority task with execution time more than 1 will make τ_2 infeasible. On the other hand, if we pre-assign the heavy task τ_2 to a processor, all tasks are feasible, see Figure 3(b). Therefore, in order to take the advantage of harmonic property to schedule general task sets, special operation, i.e. pre-assignment, needs to be performed for heavy tasks.

In HSP, the heavy task pre-assignment algorithm, Algorithm 2, follows the same strategy as introduced in [12]. Specifically, for any heavy task τ_i, let $\mathcal{P}_i^{Emp}$ denote the set of empty processors before τ_i's assignment and $|\mathcal{P}_i^{Emp}|$ denote the number of processors in this set. Then a heavy task τ_i needs to be pre-assigned to an empty processor if

$$\sum_{j>i} u_j \leq (|\mathcal{P}_i^{Emp}| - 1) \cdot \Theta(n). \tag{7}$$

Algorithm 2 Heavy-Task-Preassignment(Γ,$\mathcal{P}$)

Require:
1) Task set :$\Gamma = \{\tau_1, \tau_2, ...\tau_N\}$;
2) Multiprocessor : $\mathcal{P} = \{P_1, P_2, ..., P_M\}$;
1: $\mathcal{P}^{Pre} = \emptyset$;
2: **for** $i = 1$ to N **do**
3: **if** $u_i > 1/2$ and $\sum_{j>i} u_j \leq (|\mathcal{P}_i^{Emp}| - 1) \cdot \Theta(N)$ **then**
4: Assign τ_i to processor P_m, where $m = |\mathcal{P}|$;
5: Move P_m from $\mathcal{P}$ to $\mathcal{P}^{Pre}$;
6: **end if**
7: **end for**
8: Return $\mathcal{P}^{Pre}$;

Algorithm HSP is very similar to HSP-light, which can be obtained by replacing the function *Heavy-Task-Preassignment* in Algorithm 2 into line 4 of Algorithm 1, and removing the comment sign ("//") from line 4 to line 7. For algorithm HSP, we have the following lemma.

Lemma 4: Let Γ_{P_k} be the task set allocated to processor P_k in HSP. If the critical task is a tail task from a heavy task τ_i and P_k is fully utilized, then

$$\sum_{P_m \in \mathcal{P}^R} U(\Gamma_{P_m}) > |\mathcal{P}^R| \cdot \theta(N) \tag{8}$$

where $\mathcal{P}^R = \{P(\tau_j) | j \in [i, N]\}$.

Proof sketch: For all tasks assigned to processors in $\mathcal{P}^R$, let Γ_L (Γ_{HE}) denote the tasks with priorities lower (higher or equal) than τ_i. Since τ_i is heavy but not pre-assigned, we have

$$\sum_{\tau_j \in \Gamma_L} u_j > (|\mathcal{P}^R| - 1) \cdot \Theta(N) \tag{9}$$

Since τ_i^t is the critical task on its host processor P_q, we have

$$\sum_{j<i, \tau_j \in \Gamma_{P_q}} C_j \lceil \frac{T_i - C_i^B}{T_j} \rceil + C_i^t \geq T_i - C_i^B$$

By applying 1) $T_j \leq T_i$ for $j < i$, 2) and $T_i - C_i^B \leq \frac{1}{2} T_i$ (since $u_i^B \geq \frac{1}{2}$) into the above, we can derive

$$\sum_{\tau_j \in \Gamma_{HE}} u_j \geq 1 \tag{10}$$

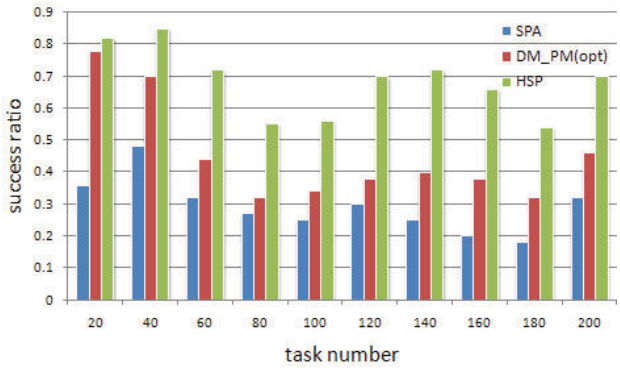


Fig. 4. Performance versus task number.

Finally, sum up equation (10) and (9), and apply $\Theta(N) \leq 1$, we can get

$$\sum_{P_m \in \mathcal{P}^R} U(\Gamma_{P_m}) > |\mathcal{P}^R| \cdot \theta(N)$$

□

From Lemma 2, Lemma 3, and Lemma 4, the following theorem can be proved. (The detailed proof is omitted due to page limit.)

Theorem 4: Given a task set Γ consisting of N tasks to be scheduled on M processors, if

$$U_M(\Gamma) \leq \Theta(N), \tag{11}$$

then Γ is feasible by HSP under RMS.

Thus, HSP can guarantee that as long as the system utilization of a task set is less than the Liu&Layland's bound, the task set is feasible by HSP.

V. Experiments

In this section, we investigate the performance of our proposed algorithms with experiments. We compare *HSP* with two most recent semi-partitioned algorithms, i.e. the *SPA* [12] and the *DM_PM(opt)* [6]. The *SPA* assigns the priority of each task by RMS, and splits a task to feed the processor to "full" (e.g. utilization equal to the Liu&Layland's bound). The *DM_PM(opt)* assigns the priorities by deadline monotonic scheduling (DMS) policy, and splits a task by exact timing analysis. We compare the performance of the two approaches with our *SPA* algorithm.

In our experiments, we randomly generated the test cases for an 8-core system. For each test point, 500 task sets were generated with utilizations evenly distributed within $[0.5, 1.00]$. The minimum inter-arrival time of each task was set to have a uniform distribution within $[50, 1000]$. We set up experiments for general task sets ($u_i \in [0.02, 1.00]$), which contained both light and heavy tasks. The scheduling results for different approaches were represented by *success ratios*, i.e. the number of feasible tasks over the number of total tasks generated.

Two groups of experiments were conducted. In the first experiment, we tested the performance versus task number. We varied the task numbers in our test cases from 20 to 200 with an increment of 20. The experimental results for three approaches were recorded and plotted in Figure 4. From Figure 4, we can

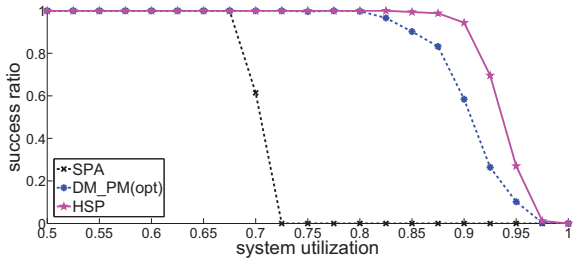


Fig. 5. Performance versus task set utilization.

observe that *HSP* in general can get success ratios significantly better than other two approaches. In addition, the performance improvement by *HSP* tends to increase as the task number increases. For example, when task number is equal to 40, *HSP* can achieve a success ratio of 83%, an improvement of 1.7 times of that by *SPA* (48%) and 1.1 times of that by *DM_PM(opt)* (70%). When the task number is 140, *HSP* can achieve a success ratio of 71%, an improvement of 2.5 times over *SPA* and 1.8 times over *DM_PM(opt)*. The improvement of *HSP* comes from two factors: first, *HSP* does not limit the total utilization of each processor by the Liu&Layland's bound as that in *SPA*; second, *HSP* takes the harmonic relationship among tasks into consideration for partitioning, and thus can obtain extra benefit to utilize system more efficiently. The more tasks are available, the more opportunities can be exploited by *HSP* to take advantage of the potential harmonic property and to improve the processor utilization.

In the second experiment, we tested the performance changes with different system utilizations for each approach. We varied the system utilization within $[0.5, 1.00]$ with an increment of 0.025. The results are shown in Figure 5. Figure 5 shows that the success ratio by *SPA* drops sharply around 0.7. This is because that while *SPA* can guarantee any task sets with utilizations no more than the Liu&Layland's bound, it cannot schedule any task set with system utilization exceeding the Liu&Layland's bound. While *DM_PM(opt)* may potentially schedule task sets with utilization higher than the Liu&Layland's bound, *HSP* can achieve a much higher performance than *DM_PM(opt)* as demonstrated in Figure 5. For example, *HSP* can achieve a success ratio two times that of *DM_PM(opt)* when the system utilization is around 0.9. Overall, the experimental results show clearly the effectiveness of our proposed algorithm.

VI. Conclusions

In this paper, we present a new semi-partitioned approach for scheduling sporadic tasks on multi-cores based on RMS. Our approach can take advantage of the harmonic relations among the tasks and improve the feasibility. Two algorithms, i.e. HSP-light and HSP, are developed to schedule light and general task sets, respectively. We formally analyze the feasibility as well as the utilization bound for both algorithms. The experimental results demonstrate that the proposed algorithm can significantly improve the scheduling performance compared with the previous work.

Acknowledgment

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

References

[1] S. Chaudhry, R. Cypher, M. Ekman, M. Karlsson, A. Landin, S. Yip, H. Zeffer, and M. Tremblay, "Rock: A high-performance sparc cmt processor," *Micro, IEEE*, vol. 29, no. 2, pp. 6 –16, March-April 2009.

[2] W. Wolf, "Multiprocessor system-on-chip technology," *Signal Processing Magazine, IEEE*, vol. 26, no. 6, pp. 50 –54, November 2009.

[3] J. Liu, *Real-Time Systems*. NJ: Prentice Hall, 2000.

[4] 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.

[5] 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

[6] 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.

[7] 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.

[8] 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.

[9] J. Anderson, V. Bud, and U. Devi, "An edf-based scheduling algorithm for multiprocessor soft real-time systems," in *Real-Time Systems, 2005. (ECRTS 2005). Proceedings. 17th Euromicro Conference on*, July 2005, pp. 199 – 208.

[10] S. Kato and N. Yamasaki, "Real-time scheduling with task splitting on multiprocessors," in *Embedded and Real-Time Computing Systems and Applications, 2007. RTCSA 2007. 13th IEEE International Conference on*, Aug. 2007, pp. 441 –450.

[11] 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.

[12] 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.

[13] A. Bastoni, B. Brandenburg, and J. Anderson, "Is semi-partitioned scheduling practical?" in *Real-Time Systems (ECRTS), 2011 23rd Euromicro Conference on*, july 2011, pp. 125 –135.

[14] 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.

[15] 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.

[16] 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

[17] C.-C. Han and H.-Y. Tyan, "A better polynomial-time schedulability test for real-time fixed-priority scheduling algorithms," in *Real-Time Systems Symposium, 1997. Proceedings., The 18th IEEE*, Dec 1997, pp. 36 –45.

[18] 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.

[19] C.-C. Han, K.-J. Lin, and C.-J. Hou, "Distance-constrained scheduling and its applications to real-time systems," *Computers, IEEE Transactions on*, vol. 45, no. 7, pp. 814 –826, Jul 1996.

[20] J. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: exact characterization and average case behavior," in *Real Time Systems Symposium, 1989., Proceedings.*, Dec 1989, pp. 166 –171.