Fixed-Priority Multiprocessor Scheduling with Liu $\&$ Layland's Utilization Bound

Nan Guan[†‡], Martin Stigge[†], Wang Yi[†‡] and Ge Yu[‡]

[†]*Department of Information Technology,*
Uppsala University, Sweden

[‡]*Department of Computer Science and Technology,*
Northeastern University, China

***Abstract*—Liu and Layland discovered the famous utilization bound $N(2^{\frac{1}{N}}-1)$ for fixed-priority scheduling on single-processor systems in the 1970's. Since then, it has been a long standing open problem to find fixed-priority scheduling algorithms with the same bound for multiprocessor systems. In this paper, we present a partitioning-based fixed-priority multiprocessor scheduling algorithm with Liu and Layland's utilization bound.**

***Keywords*-real-time systems; utilization bound; multiprocessor; fixed priority scheduling**

I. Introduction

Utilization bound is a well-known concept first introduced by Liu and Layland in their seminal paper [18]. Utilization bound can be used as a simple and practical way to test the schedulability of real-time task sets, as well as a good metric to evaluate the "quality" of a scheduling algorithm. It was shown that the utilization bound of *Rate Monotonic Scheduling* (RMS) on single processors is $N(2^{\frac{1}{N}}-1)$. For simplicity of presentation we let $\Theta(N)=N(2^{\frac{1}{N}}-1)$.

Multiprocessor scheduling are usually categorized into two paradigms [10]: *global scheduling*, in which each task can execute on any available processor in the run time, and *partitioned scheduling* in which each tasks is assigned to a processor beforehand, and during the run time each task can only execute on this particular processor. Although global scheduling on average utilizes computing resource better, the best known utilization bound of global fixed-priority scheduling is only 38% [3], which is much lower than the best known result of partitioned fixed-priority scheduling 50% [7]. 50% is also known as the maximum utilization bound for both global and partitioned fixed-priority scheduling [4], [19]. Although there exist scheduling algorithms, like the pfair family [2], [9], offering utilization bounds of 100%, these scheduling algorithms are not priority-based and incur much higher context-switch overhead [11].

Recently a number of works have been done on the *semi-partitioned scheduling*, which can exceed the maximum utilization bound 50% of the partitioned scheduling. In semi-partitioned scheduling, most tasks are statically assigned to one fixed processor as in partitioned scheduling, while a few number of tasks are split into several subtasks, which are assigned to different processors. A recent work [17] has shown that the worst-case utilization bound of semi-partitioned fixed-priority scheduling can achieve 65%, which is still lower than 69.3% (the worst-case value of $\Theta(N)$ when N is increasing to the infinity). This gap is even larger with a smaller N.

In this paper, we propose a new fixed-priority scheduling algorithm for multiprocessor systems based on semi-partitioned scheduling, whose utilization bound is $\Theta(N)$. The algorithm uses RMS on each processor, and has the same task splitting overhead as in previous work.

We first propose a semi-partitioned fixed-priority scheduling algorithm, whose utilization bound is $\Theta(N)$ for a class of task sets in which the utilization of each task is no larger than $\Theta(N)/(1+\Theta(N))$. This algorithm assigns tasks in decreasing period order, and always selects the processor with the least workload assigned so far among all processors, to assign the next task. Then we remove the constraint on the utilization of each task, by introducing an extra task pre-assigning mechanism; the algorithm can achieve the utilization bound of $\Theta(N)$ for any task set.

The rest of the paper is structured as follows: Section II reviews the prior work on semi-partitioned scheduling; Section III introduces the notations and the basic concept of semi-partitioned scheduling. The first and second proposed algorithm, as well as their worst-case utilization bound property, is presented in Section IV and V respectively. Finally, the conclusion is made in Section VI.

II. Prior Work

Semi-partitioned scheduling has been studied with both EDF scheduling [1], [8], [5], [6], [12], [13], [16] and fixed-priority scheduling [14], [15], [17].

The first semi-partitioned scheduling algorithm is EDF-fm [1] for soft real-time systems based on EDF scheduling. Andersson et al. proposed EKG [8] for hard real-time systems, in which split tasks are forced to executed in certain

This work was partially sponsored by CoDeR-MP, UPMARC, and NSF of China under Grant No. 60973017 and 60773220.

time slots. Later EKG was extended to sporadic and arbitrary deadline task systems [5] [6] with the similar idea. Kato et al. proposed EDDHP and EDDP [12] [13] in which split tasks are scheduled based on priority rather than time slots. The worst-case utilization bound of EDDP is 65%. Later Kato et al. proposed EDF-WM, which can significantly reduce the context switch overhead against previous work.

There are relatively fewer works on the fixed-priority scheduling side. Kato et al. proposed RMDP [14] and DMPM [15], both with the worst-case utilization bound of 50%. which is the same as the partitioned scheduling without task splitting. Recently, Lakshmanan et al. [17] proposed the algorithm PDMS_HPTS_DS, which can achieve the worst-case utilization bound of 65%, and can achieve the bound 69.3% for a special type of task sets that consist of "light" tasks. They also conducted case studies on an Intel Core 2 Duo processor to characterize the practical overhead of task-splitting, and showed that the cache overheads due to task-splitting can be expected to be negligible on multi-core platforms.

III. Basic Concepts

We first introduce the processor platform and task model. The multiprocessor platform consists of M identical processors $\{P_1, P_2, ...P_M\}$. A task set $\tau = \{\tau_1, \tau_2, ..., \tau_N\}$ consists of N independent tasks. Each task τ_i is a 2-tuple $\langle C_i, T_i \rangle$, where C_i is the worst-case execution time, T_i is the minimum inter-release separation (also called period). T_i is also τ_i's relative deadline.

Tasks in τ are sorted in non-decreasing period order, i.e., $j > i \Rightarrow T_j \geq T_i$. Since our proposed algorithms use rate-monotonic scheduling (RMS) as the scheduling algorithm on each processor, we can use the task indices to represent the task priorities, i.e., τ_i has higher priority than τ_j if and only if $i < j$. The *utilization* of each task τ_i is defined as $U_i = C_i/T_i$.

We recall the classical result of Liu and Layland:

Theorem 1 ([18])**.** *On a single-processor system, each task set τ with*

$$\sum_{\tau_i \in \tau} U_i \leq N(2^{\frac{1}{N}} - 1)$$

is schedulable using rate-monotonic scheduling (RMS).

The utilization bound of our proposed semi-partitioned scheduling algorithm is built upon this result. In the remainder of this paper, we use $\Theta(N)$ to denote the above utilization bound for N tasks:

$$\Theta(N) = N(2^{\frac{1}{N}} - 1) \quad (1)$$

We further define the *utilization of a task set τ* in multiprocessor scheduling on M processors as

$$U(\tau) = \sum_{\tau_i \in \tau} U_i / M \quad (2)$$

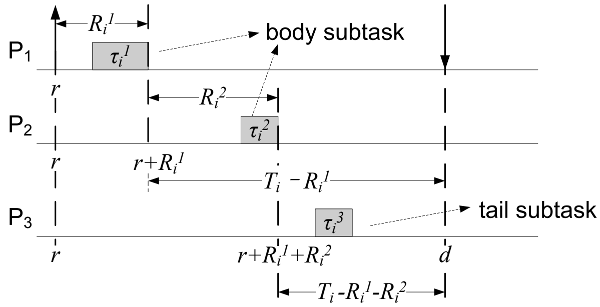


Figure 1. Subtasks

For simplicity of presenting our algorithms, we assume each task $\tau_i \in \tau$ has utilization $U_i \leq \Theta(N)$. Note that this assumption does not invalidate our results on task sets containing tasks with utilization higher than $\Theta(N)$: If in a task set with $U(\tau) \leq \Theta(N)$ there are tasks with a higher (individual) utilization than $\Theta(N)$, we can just let them run each exclusively on an own processor. The remaining task set on the remaining processors still has a utilization of at most $\Theta(N)$. If we are able to show its schedulability, then together this results in the desired bound $\Theta(N)$ for the full task set.

A semi-partitioned scheduling algorithm consists of two parts: the *partitioning algorithm*, which determines how to split and assign each task (or rather each of its parts) to a fixed processor, and the *scheduling algorithm*, which determines how to schedule the tasks assigned to each processor.

With the partitioning algorithm, most tasks are assigned to a processor and only execute on this processor at run time. We call these tasks *non-split tasks*. Other tasks are called *split tasks*, which are split into several *subtasks*. Each subtask of split task τ_i is assigned to (thereby executes on) a different processor, and the sum of the execution time of all subtasks equals C_i. For example, in Figure 1 the task τ_i is split into three subtasks τ_i^1, τ_i^2 and τ_i^3, executing on processor P_1, P_2 and P_3, respectively.

The subtasks of a task need to be synchronized to execute correctly. For example, in Figure 1, τ_i^2 can not start execution until τ_i^1 is finished. This equals deferring the actual ready time of τ_i^2 by up to R_i^1 (relative to τ_i's original release time), where R_i^1 is the worst-case response time of τ_i^1. One can regard this as shortening the actual relative deadline of τ_i^2 by up to R_i^1. Similarly, the actual ready time of τ_i^3 is deferred by up to $R_i^1 + R_i^2$, and τ_i^3's actual relative deadline is shortened by up to $R_i^1 + R_i^2$. We use τ_i^k to denote the k^{th} subtask of a split task τ_i, and define τ_i^k's *synthetic deadline* as

$$\triangle_i^k = T_i - \sum_{l \in [1, k-1]} R_i^l \quad (3)$$

Thus, we represent each subtask τ_i^k by a 3-tuple

$\langle c_i^k, T_i, \triangle_i^k \rangle$, in which c_i^k is the execution time of τ_i^k, T_i is the original period, $\triangle_i^k$ is the synthetic deadline. For consistency, each non-split task τ_i can be represented by a single subtask τ_i^1 with $c_i^1 = C_i$ and $\triangle_i^1 = T_i$.

The *normal utilization* of a subtask τ_i^k is $U_i^k = c_i^k / T_i$, and we define another new metric, the *synthetic utilization* V_i^k, to describe τ_i^k's workload with its synthetic deadline:

$$V_i^k = c_i^k / \triangle_i^k \tag{4}$$

We call the last subtask of τ_i its *tail subtask*, denoted by τ_i^t and the other subtasks its *body subtasks*, as shown in Figure 1. We use $\tau_i^{b_j}$ to denote the j^{th} body subtask. We use $\tau_i \mapsto P_q$ to denote that τ_i is assigned to processor P_q, and say that P_q is the *host processor* of τ_i.

A task set τ is *schedulable* under a semi-partitioned scheduling algorithm $\mathcal{A}$, if after assigning tasks to processors by $\mathcal{A}$'s partitioning algorithm, each task $\tau_i \in \tau$ can meet its deadline under $\mathcal{A}$'s scheduling algorithm.

IV. The First Algorithm SPA1

A significant difference between SPA1 and the algorithms in previous work is that SPA1 employs a "worst-fit" partitioning, while all previous algorithms employ a "first-fit" partitioning [17], [14], [15].

The basic procedure of "first-fit" partitioning is as follows: one selects a processor, and assign tasks to this processor as much as possible to fill its capacity, then pick the next processor and repeat the procedure. In contrast, the "worst-fit" partitioning always selects the processor with the minimal total utilization of tasks that have been assigned to it, so the occupied capacities of all processors are increased roughly "in turn".

The reason for us to prefer worst-fit partitioning is intuitively explained as follows. A subtask τ_i^k's actual deadline ($\triangle_i^k$) is shorter than τ_i's original deadline T_i, and the sum of the synthetic utilizations of all τ_i's subtasks is larger than τ_i's original utilization U_i, which is the key difficulty for semi-partitioned scheduling to achieve the same utilization bound as on single-processors. With worst-fit partitioning, the occupied capacity of all processors are increased "in turn", and task splitting only occurs when the capacity of a processor is completely "filled". Then, if one partitions all tasks in increasing priority order, the split tasks in worst-fit partitioning will generally have relatively high priority levels on each processor. This is good for the schedulability of the task set, since the tasks with high priorities usually have better chance to be schedulable, so they can tolerate the shortened deadlines better. Consider an extreme scenario: if one can make sure that all split tasks' subtasks have the highest priority on their host processors, then there is no need to consider the shortened deadlines of these subtasks, since, being of the highest priority level on each processor, they are schedulable anyway. Thus, as long as the split tasks with shorten deadlines do not cause any problem, Liu and Layland's utilization bound can be easily achieved. The philosophy behind our proposed algorithms is *making the split subtasks get as high priority as possible on each processor*.

In contrast, with the first-fit partitioning, a split subtask may get quite low priority on its host processors[1]. For instance, with the algorithm in [17] that achieves the utilization bound of 65%, in the worst case the second subtask of a split task will always get the lowest priority on its host processor.

As will be seen later in this section, SPA1 does not completely solve the problem. More precisely, SPA1 is restricted to a class of *light* task sets, in which the utilization of each task is no larger than $\Theta(N)/(1+\Theta(N))$. Intuitively, this is because if a task's utilization is very large, its tail subtask might still get a relatively low priority on its host processor, even using worst-fit partitioning. (We will solve this problem with SPA2 in Section V.)

In the following, we will introduce SPA1 as well as its utilization bound property. The remaining part of this section is structured as follows: we first present the partitioning algorithm of SPA1, and show that any task set τ satisfying $U(\tau) \leq \Theta(N)$ can be successfully partitioned by SPA1. Then we introduce how the tasks assigned to each processor are scheduled. Next, we prove that if a *light* task set is successfully partitioned by SPA1, then all tasks can meet their deadlines under the scheduling algorithm of SPA1. Together, this implies that any *light* task set with $U(\tau) \leq \Theta(N)$ is schedulable by SPA1, and finally indicates the utilization bound of SPA1 is $\Theta(N)$ for *light* task sets.

```
if $U(\tau) > \Theta(N)$ then abort
$UQ := [\tau_N^1, \tau_{N-1}^1, ..., \tau_1^1]$
$\Psi[1...M]$ := all zeros
while $UQ \neq \emptyset$ do
    $P_q$ := the processor with the minimal $\Psi$
    $\tau_i^k$ := pop_front($UQ$)
    if ($U_i^k + \Psi[q] \leq \Theta(N)$) then
        $\tau_i^k \mapsto P_q$
        $\Psi[q] := \Psi[q] + U_i^k$
    else
        split $\tau_i^k$ into two parts $\tau_i^k$ and $\tau_i^{k+1}$ such that
        $U_i^k + \Psi[q] = \Theta(N)$
        $\tau_i^k \mapsto P_q$
        $\Psi[q] := \Theta(N)$
        push_front($\tau_i^{k+1}$, $UQ$)
    end if
end while
```

Algorithm 1: The partitioning algorithm of SPA1.

[1] Under the algorithms in [15], a split subtask's priority is artificially advanced to the highest level on its host processor, which breaks down the RMS priority order and thereby leads to a lower utilization bound.

A. SPA1*: Partitioning and Scheduling*

The partitioning algorithm of SPA1 is very simple, which can be briefly described as follows:

- We assign tasks in increasing priority order, and always select the processor on which the total utilization of tasks have been assigned so far is minimal among all the processors.
- When a task (subtask) can not be assigned entirely to the current selected processor, we split it into two parts and assign the first part such that the total utilization of the current selected processor is $\Theta(N)$, and assign the second part to the next selected processor.

The precise description of the partitioning algorithm is in Algorithm 1. UQ is the list accommodating unassigned tasks, sorted in increasing priority order. UQ is initialized by $\{\tau_N^1, \tau_{N-1}^1, ..., \tau_1^1\}$, in which each element $\tau_i^1 = \langle c_i^1 = C_i, T_i, \triangle_k^1 = T_i\rangle$ is the initial subtask form of task τ_i. Each element $\Psi[q]$ in the array $\Psi[1...M]$ denotes the sum of the utilization of tasks that have been assigned to processor P_q.

The work flow of SPA1 is as follows. In each loop iteration, we pick the task at the front of UQ, denoted by τ_i^k, which has the lowest priority among all unassigned tasks. We try to assign τ_i^k to the processor P_q, which has the minimal $\Psi[q]$ among all elements in $\Psi[1...M]$. If

$$U_i^k + \Psi[q] \leq \Theta(N)$$

then we can assign the entire τ_i^k to P_q, since there is enough capacity available on P_q. Otherwise, we split τ_i^k into two subtasks τ_i^k and τ_i^{k+1}, such that

$$U_i^k + \Psi[q] = \Theta(N)$$

(Note that with $U_i^k = c_i^k / T_i$ we denote the utilization of subtask τ_i^k.) We further set $\Psi[q] := \Theta(N)$, which means this processor P_q is *full* and we will not assign any more tasks to P_q.

Then we insert τ_i^{k+1} back to the front of UQ, to assign it in the next loop iteration. We continue this procedure until all tasks have been assigned.

It is easy to see that all task sets below the desired utilization bound can be successfully partitioned by SPA1:

Lemma 1. *Any task set with*

$$U(\tau) \leq \Theta(N) \tag{5}$$

can be successfully partitioned to M processors with SPA1.

Note that there is no schedulability guarantee in the partitioning algorithm. It will be proved in next subsection.

After the tasks are assigned (and possibly split) to the processors by the partitioning algorithm of SPA1, they will be scheduled using RMS on each processor locally, i.e., with their original priorities. The subtasks of a split task respect their precedence relations, i.e., a split subtask τ_i^k is ready for execution when its preceding subtask τ_i^{k-1} on some other processor has finished.

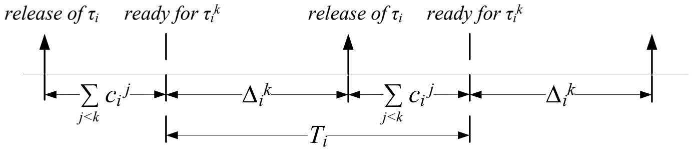


Figure 2. Each subtask τ_i^k can be viewed as an independent task with period of T_i and deadline of $\triangle_i^k$.

B. *Schedulability*

We first show an important property of SPA1:

Lemma 2. *After partitioning according to* SPA1*, each body subtask has the highest priority on its host processor.*

Proof: In the partitioning algorithm of SPA1, task splitting only occurs when a processor is full. Thus, after a body task was assigned to a processor, there will be no more tasks assigned to it. Further, the tasks are partitioned in increasing priority order, so all tasks assigned to the processor before have lower priority. ■

By Lemma 2, we further know that the response time of each body subtask equals its execution time, so the synthetic deadline $\triangle_i^t$ of each tail subtask τ_i^t is calculated as follows:

$$\triangle_i^t = T_i - \sum_{j \in [1,B]} c_i^{bj} = T_i - (C_i - c_i^t) \tag{6}$$

So we can view the scheduling in SPA1 on each processor without considering the synchronization between the subtasks of a split task, and just *regard every split subtask τ_i^k as an independent task with period T_i and a shorter relative deadline $\triangle_i^k$ calculated by Equation (6)*, as shown in Figure 2.

In the following we prove the schedulability of non-split tasks, body subtasks and tail subtasks, respectively.

1) Non-split Tasks:

Lemma 3. *If task set τ with $U(\tau) \leq \Theta(N)$ is partitioned by* SPA1*, then any* ***non-split task*** *of τ can meet its deadline.*

Proof: The tasks on each processor are scheduled by RMS, and the sum of the utilization of all tasks on a processor is no larger than $\Theta(N)$. Further, the deadlines of the non-split tasks are unchanged and therefore still equal their periods. Thus, each non-split task is schedulable. ■

Note that although the synthetic deadlines of other subtasks are shorter than their original periods, this does not affect the schedulability of the non-split tasks, since only the periods of these subtasks are relevant to the schedulability of the non-split tasks.

2) Body Subtasks:

Lemma 4. *If task set τ with $U(\tau) \leq \Theta(N)$ is partitioned by* SPA1*, then any* ***body subtask*** *of τ can meet its deadline.*

Proof: The body subtasks have the highest priorities on their host processors and will therefore always meet

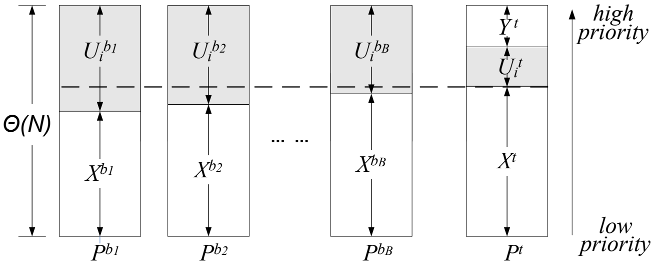
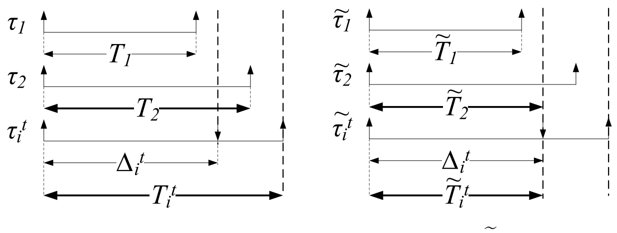


Figure 3. Illustration of X^{b_j}, X^t and Y^t



(a) Γ (b) $\widetilde{\Gamma}$

Figure 4. Illustration of $\widetilde{\Gamma}$

their deadlines. (This holds even though the deadlines were shortened because of the task splitting). ■

3) Tail Subtasks:

Now we prove the schedulability for an arbitrary tail subtask τ_i^t, during which we only focus on τ_i^t, but do not consider whether other tail subtasks are schedulable or not. Since the same reasoning can be applied to every tail subtask, the proofs guarantee that all tail subtasks are schedulable.

Suppose task τ_i is split into B body subtasks and one tail subtask. Recall that we use $\tau_i^{b_j}, j \in [1, B]$ to denote the j^{th} body subtask of τ_i, and τ_i^t to denote τ_i's tail subtask. $U_i^{b_j} = c_i^{b_j}/T_i$ and $U_i^t = c_i^t/T_i$ denotes $\tau_i^{b_j}$'s and τ_i^t's original utilization respectively.

Additionally, we use the following notations (cf. Figure 3):

- For each body subtask $\tau_i^{b_j}$, let X^{b_j} denote the sum of the utilizations of all the tasks τ_k assigned to P^{b_j} with lower priority than $\tau_i^{b_j}$.
- For the tail subtask τ_i^t, let X^t denote the sum of the utilizations of all the tasks assigned to P^t with lower priority than τ_i^t.
- For the tail subtask τ_i^t, let Y^t denote the sum of the utilizations of all the tasks assigned to P^t with higher priority than τ_i^t.

We can use these now for the schedulability of the tail subtasks:

Lemma 5. *Suppose a tail subtask τ_i^t is assigned to processor P_t. If τ_i^t satisfies*

$$Y^t \cdot T_i/\triangle_i^t + V_i^t \leq \Theta(N), \tag{7}$$

then τ_i^t can meet its deadline.

Proof: The proof idea is as follows: We consider the set Γ consisting of τ_i^t and all tasks with higher priority than τ_i^t *on the same processor*, i.e., the tasks contributing to Y^t. For this set, we construct a new task set $\widetilde{\Gamma}$, in which the tasks' periods that are larger than $\triangle_i^t$ are all reduced to $\triangle_i^t$. The main idea is to first show that the counterpart of τ_i^t is schedulable with this new set $\widetilde{\Gamma}$ by RMS because of the utilization bound, and then to prove this implies the schedulability of τ_i^t in the original set Γ.

In particular, let P_t be the processor to which τ_i^t is assigned. We define Γ as follows:

$$\Gamma = \{\tau_h^k \mid \tau_h^k \mapsto P_t \wedge h \leq i\} \tag{8}$$

We now give the construction of $\widetilde{\Gamma}$: For each task $\tau_h^k \in \Gamma$, we have a counterpart $\widetilde{\tau_h^k}$ in $\widetilde{\Gamma}$. The only difference is that we possibly reduce the periods:

$$\widetilde{c_h^k} = c_h^k, \quad \widetilde{T_h} = \begin{cases} T_h, & \text{if } T_h \leq \triangle_i^t \\ \triangle_i^t, & \text{if } T_h > \triangle_i^t \end{cases}$$

We also keep the same priority order of tasks in $\widetilde{\Gamma}$ as their counterparts in Γ, which is still a rate-monotonic ordering.

Figure 4 illustrates the construction. In Figure 4(a), Γ contains three tasks. τ_1 has a period that is smaller than $\triangle_i^t$, and τ_2 has a larger one. Further, τ_i^t is contained in Γ. According to the construction, $\widetilde{\Gamma}$ in Figure 4(b) has also three tasks $\widetilde{\tau_1}$, $\widetilde{\tau_2}$ and $\widetilde{\tau_i^t}$, where only the periods of $\widetilde{\tau_2}$ and $\widetilde{\tau_i^t}$ are reduced to $\triangle_i^t$.

Now we show the schedulability of $\widetilde{\tau_i^t}$ in $\widetilde{\Gamma}$. We do this by showing the sufficient upper bound of $\Theta(N)$ on the total utilization of $\widetilde{\Gamma}$.

$$U(\widetilde{\Gamma}) = \sum_{\tau_h^k \in \Gamma} c_h^k/\widetilde{T_h} = \sum_{\tau_h^k \in \Gamma \setminus \{\tau_i^t\}} c_h^k/\widetilde{T_h} + V_i^k \tag{9}$$

We now do a case distinction for tasks $\widetilde{\tau_h^k} \in \widetilde{\Gamma}$, according to whether their periods were reduced or not.

- If $T_h \leq \triangle_i^t$, we have $\widetilde{T_h} = T_h$. Since $T_i > \triangle_i^t$, we have:
$$c_h^k/\widetilde{T_h} = c_h^k/T_h = U_h^k < U_h^k \cdot T_i/\triangle_i^t$$
- If $T_h > \triangle_i^t$, we have $\widetilde{T_h} = \triangle_i^t$. Because of the priority ordered by periods, we have $T_h \leq T_i$. Thus:
$$c_h^k/\widetilde{T_h} = c_h^k/\triangle_i^t \leq c_h^k/T_h \cdot T_i/\triangle_i^t = U_h^k \cdot T_i/\triangle_i^t$$

Both cases lead to $c_h^k/\widetilde{T_h} \leq U_h^k \cdot T_i/\triangle_i^t$, so we can apply this to (9) from above:

$$U(\widetilde{\Gamma}) \leq \sum_{\tau_h^k \in \Gamma \setminus \{\tau_i^t\}} U_h^k \cdot T_i/\triangle_i^t + V_i^k \tag{10}$$

Since $Y^t = \sum_{\tau_h^k \in \Gamma \setminus \{\tau_i^t\}} U_h^k$, we have:

$$U(\widetilde{\Gamma}) \leq Y^t \cdot T_i / \triangle_i^t + V_i^t$$

Finally, by the assumption from Condition (7) we know that the right-hand side is at most $\Theta(N)$, and thus $U(\widetilde{\Gamma}) \leq \Theta(N)$. Therefore, $\widetilde{\tau_i^k}$ is schedulable. Note that in $\widetilde{\Gamma}$ there could exist other tail subtasks whose deadlines are shorter than their periods. However, this does not invalidate that the condition $U(\widetilde{\Gamma}) \leq \Theta(N)$ is sufficient to guarantee the schedulability of $\widetilde{\tau_i^t}$ under RMS.

Now we need to see that this implies the schedulability of τ_i^t. Recall that the only difference between Γ and $\widetilde{\Gamma}$ is that the period of a task in Γ is possibly larger than its counterpart in $\widetilde{\Gamma}$. So the interference τ_i^t suffered from the higher-priority tasks in Γ, is no larger than the interference $\widetilde{\tau_i^t}$ suffered in $\widetilde{\Gamma}$, and since the deadlines of $\widetilde{\tau_i^t}$ and τ_i^t are the same, we know the schedulability of $\widetilde{\tau_i^t}$ implies the schedulability of τ_i^t. ■

It remains to show that Condition (7) holds, which was the assumption for this lemma and thus a sufficient condition for tail subtasks to be schedulable. As in the introduction of this section, this condition does not hold in general for SPA1, but only for certain *light* task sets:

Definition 1. *A task τ_i is a* light *task if*

$$U_i \leq \frac{\Theta(N)}{1 + \Theta(N)}.$$

Otherwise, τ_i is a heavy *task.*

A task set τ is a light *task sets if all tasks in τ are light tasks.*

Lemma 6. *Suppose a tail subtask τ_i^t is assigned to processor P_t. If τ_i is a light task, we have*

$$Y^t \cdot T_i / \triangle_i^t + V_i^t \leq \Theta(N).$$

Proof: We will first derive a general upper bound on Y^t based on the properties of X^{b_j}, X^t and the subtasks' utilizations. Based on this, we derive the bound we want to show, using the assumption that τ_i is a light task.

For deriving the upper bound on Y^t, we note that as soon as a task is split into a body subtask and a rest, the processor hosting this new body subtask is full, i.e., its utilization is $\Theta(N)$. Further, each body subtask has by construction the highest priority on its host processor, so we have:

$$\forall j \in [1, B] : U_i^{b_j} + X^{b_j} = \Theta(N)$$

We sum over all B of these equations, and get:

$$\sum_{j \in [1,B]} U_i^{b_j} + \sum_{j \in [1,B]} X^{b_j} = B \cdot \Theta(N) \tag{11}$$

Now we consider the processor containing τ_i^t, denoted by P_t. Its total utilization is $X^t + U_i^t + Y^t$ and is at most $\Theta(N)$, i.e.,

$$X^t + U_i^t + Y^t \leq \Theta(N).$$

We combine this with (11) and get:

$$Y^t \leq \frac{\sum_{j \in [1,B]} U_i^{b_j}}{B} + \frac{\sum_{j \in [1,B]} X^{b_j}}{B} - U_i^t - X^t \tag{12}$$

In order to simplify this, we recall that during the partitioning phase, we always select the processor with the smallest total utilization of tasks that have been assigned to it so far. (Recall line 5 in Algorithm 1). This implies $X^{b_j} \leq X^t$ for all subtasks $\tau_i^{b_j}$. Thus, the sum over all X^{b_j} is bounded by $B \cdot X^t$ and we can cancel out both terms in (12):

$$Y^t \leq \frac{\sum_{j \in [1,B]} U_i^{b_j}}{B} - U_i^t$$

Another simplification is possible using that $B \geq 1$ and that τ_i's utilization U_i is the sum of the utilizations of all of its subtasks, i.e., $\sum_{j \in [1,B]} U_i^{b_j} = U_i - U_i^t$:

$$Y^t \leq U_i - 2 \cdot U_i^t$$

We are now done with the first part, i.e., deriving an upper bound for Y^t. This can easily be transformed into an upper bound on the term we are interested in:

$$Y^t \cdot \frac{T_i}{\triangle_i^t} + V_i^t \leq (U_i - 2 \cdot U_i^t) \cdot \frac{T_i}{\triangle_i^t} + V_i^t \tag{13}$$

For the rest of the proof, we try to bound the right-hand side from above by $\Theta(N)$ which will complete the proof. The key is to bring it into a form that is suitable to use the assumption that τ_i is a *light* task.

As a first step, we use that the synthetic deadline of τ_i^t is the period T_i reduced by the total computation time of τ_i's body subtasks, i.e., $\triangle_i^t = T_i - (C_i - c_i^t)$, cf. Equation (6). Further, we use the definitions $U_i = C_i/T_i$, $U_i^t = c_i^t/T_i$ and $V_i^t = c_i^t/\triangle_i^t$ to derive:

$$(U_i - 2 \cdot U_i^t) \cdot \frac{T_i}{\triangle_i^t} + V_i^t = \frac{C_i - c_i^t}{T_i - (C_i - c_i^t)}$$

Since $c_i^t > 0$, we can find a simple upper bound of the right-hand side:

$$\frac{C_i - c_i^t}{T_i - (C_i - c_i^t)} = \frac{T_i}{T_i - (C_i - c_i^t)} - 1 < \frac{T_i}{T_i - C_i} - 1$$

Since τ_i is a *light* task, we have

$$U_i \leq \frac{\Theta(N)}{1 + \Theta(N)}$$

and by applying $U_i = C_i/T_i$ to the above, we can obtain

$$\frac{T_i}{T_i - C_i} - 1 \leq \Theta(N)$$

Thus, we have established that $\Theta(N)$ is an upper bound of $Y^t \cdot \frac{T_i}{\triangle_i^t} + V_i^t$ with which we started in (13). ■

From Lemmas 5 and 6 it follows directly the desired property:

Lemma 7. *If task set τ with $U(\tau) \leq \Theta(N)$ is partitioned by* SPA1*, then any* ***tail subtask*** *of a* light *task of τ can meet its deadline.*

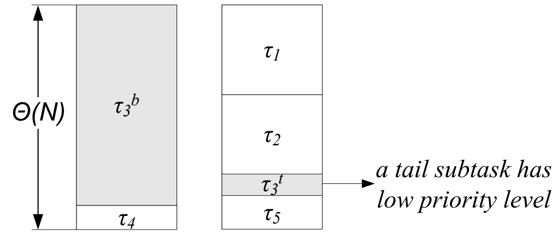


Figure 5. The tail subtask of a task with large utilization may have a low priority level

C. Utilization Bound

By Lemma 1 we know that a task set τ can be successfully partitioned by the partitioning algorithm of SPA1 if $U(\tau)$ is no larger than $\Theta(N)$. If τ has been successfully partitioned, by Lemma 3 and 4 we know that all the non-split task and body subtasks are schedulable. By Lemma 7 we know a tail subtask τ_i^k is also schedulable if τ_i is a light task. Since, in general, it is a priori unknown which tasks will be split, we pose this constraint of being light to all tasks in τ to have a sufficient schedulability test condition:

Theorem 2. *Let τ be a task set only containing* light *tasks. τ is schedulable with* SPA1 *on M processors if*

$$U(\tau) \leq \Theta(N) \tag{14}$$

In other words, the utilization bound of SPA1 is $\Theta(N)$ for task sets only containing tasks with utilization no larger than $\Theta(N)/(1+\Theta(N))$.

$\Theta(N)$ is a decreasing function with respect to N, which means the utilization bound is higher for task sets with fewer tasks. We use N^* to denote the maximal number of tasks (subtasks) assigned to on each processor, so $\Theta(N^*)$, which is strictly larger than $\Theta(N)$ also serves as the utilization bound on each processor. Therefore we can use $\Theta(N^*)$ to replace $\Theta(N)$ in the derivation above, and get that the utilization bound of SPA1 is $\Theta(N^*)$ for task sets only containing tasks with utilization no larger than $\Theta(N^*)/(1+\Theta(N^*))$. It is easy to see that there is at least one task assigned to each processor, and two subtasks of a task can not be assigned to the same processor. Therefore the number of tasks executing one each processor is at most $N - M + 1$, which can be used as an over-approximation of N^*.

V. The Second Algorithm SPA2

In this section we introduce our second semi-partitioned fixed-priority scheduling algorithm SPA2, which has the utilization bound of $\Theta(N)$ for task sets without any constraint.

As discussed in the beginning of Section IV, the key point for our algorithms to achieve high utilization bounds is to make each split task getting a priority as high as possible on its host processor. With SPA1, the tail subtask of a task with very large utilization could have a relatively low priority on its host processor, as the example in Figure 5 illustrates. This is why the utilization bound of SPA1 is not applicable to task sets containing heavy tasks.

Table I
An example task set

Task	C_i	T_i	Heavy Task?	Priority
τ_1	3	4	yes	highest
τ_2	4.25	10	no	middle
τ_3	4.25	10	no	lowest

To solve this problem, we propose the second semi-partitioned algorithm SPA2 in this section. The main idea of SPA2 is to pre-assign each heavy task whose tail subtask might get a low priority, before partitioning other tasks, therefore these heavy tasks will not be split.

Note that if one simply pre-assigns all heavy tasks, it is still possible for some tail subtask to get a low priority level on its host processor. Consider the task set in Table I with 2 processors, and for simplicity we assume $\Theta(N) = 0.8$, and $\Theta(N)/(1+\Theta(N)) = 4/9$. If we pre-assign the heavy task τ_1 to processor P_1, then assign τ_2 and τ_3 by the partitioning algorithm of SPA1, the task partitioning looks as follows:

1) $\tau_1 \mapsto P_1$,
2) $\tau_3 \mapsto P_2$,
3) τ_2 can not be entirely assigned to P_2, so it is split into two subtasks $\tau_2^1 = \langle 3.75, 10, 10 \rangle$ and $\tau_2^2 = \langle 0.5, 10, 6.25 \rangle$, and $\tau_2^1 \mapsto P_2$,
4) $\tau_2^2 \mapsto P_1$.

Then the tasks on each processor are scheduled by RMS. We can see that the tail subtask τ_2^2 has the lowest priority on P_1 and will miss its deadline due to the higher priority task τ_1. However, if we do not pre-assign τ_1 and just do the partitioning with SPA1, this task set is schedulable.

To overcome this problem, a more sophisticated pre-assigning mechanism is employed in our second algorithm SPA2. Intuitively, SPA2 pre-assigns exactly those heavy tasks for which pre-assigning them will not cause any tail subtask to miss deadline. This is checked using a simple test. Those heavy tasks that don't satisfy this test will be assigned (and possibly split) later together with the light tasks. The key for this to work is, that for these heavy tasks, we can use the property of failing the test in order to show that their tail subtasks will not miss the deadlines either.

A. SPA2*: Partitioning and Scheduling*

We first introduce some notations. If a heavy task τ_i is pre-assigned to a processor P_q in SPA2, we call τ_i as a *pre-assigned task*, otherwise a *normal task*, and call P_q as a *pre-assigned processor*, otherwise a *normal processor*.

The partitioning algorithm of SPA2 contains three steps:

1) We first pre-assign the heavy tasks that satisfy a particular condition to one processor each.

```
1:  if $U(\tau) > \Theta(N)$ then abort
2:  $PQ$ := $[P_1, P_2, ..., P_M]$
3:  $PQ_{pre}$ := $\emptyset$
4:  $UQ$ := $\emptyset$
5:  $\Psi[1...M]$ := all zeros
6:  for $i := 1$ to $N$ do
7:      if $\tau_i$ is heavy $\wedge$
          $\sum_{j>i} U_j \leq (|PQ| - 1) \cdot \Theta(N)$ then
8:          $P_q$ := pop_front($PQ$)
9:          Pre-assign $\tau_i$ to $P_q$
10:         push_front($P_q$, $PQ_{pre}$)
11:         $\Psi[q] := \Psi[q] + U_i$
12:     else
13:         push_front($\tau_i^1$, $UQ$)
14:     end if
15: end for
16: while $UQ \neq \emptyset$ do
17:     $\tau_i^k$ := pop_front($UQ$)
18:     if $\exists P_q \in PQ : \Psi[q] \neq \Theta(N)$ then
19:         $P_q$ := the element in $PQ$ with the minimal $\Psi$
20:     else
21:         $P_q$ := pop_front($PQ_{pre}$)
22:     end if
23:     if $U_i^k + \Psi[q] \leq \Theta(N)$ then
24:         $\tau_i^k \mapsto P_q$
25:         $\Psi[q] := \Psi[q] + U_i^k$
26:         if $P_q$ came from $PQ_{pre}$ then
27:             push_front($P_q$, $PQ_{pre}$)
28:         end if
29:     else
30:         split $\tau_i^k$ into two parts $\tau_i^k$ and $\tau_i^{k+1}$ such that
            $U_i^k + \Psi[q] = \Theta(N)$
31:         $\tau_i^k \mapsto P_q$
32:         $\Psi[q] = \Theta(N)$
33:         push_front($\tau_i^{k+1}$, $UQ$)
34:     end if
35: end while
```

Algorithm 2: The partitioning algorithm of SPA2.

2) We do task partitioning with the remaining (i.e. normal) tasks and remaining (i.e. normal) processors using SPA1 until all the normal processors are full.
3) The remaining tasks are assigned to the pre-assigned processors; the assignment selects one processor to assign as many tasks as possible, until it becomes full, then select the next processor.

The precise description of the partitioning algorithm of SPA2 is shown in Algorithm 2. We first introduce the data structures used in the algorithm:

- PQ is the list of all processors. It is initially $[P_1, P_2, ..., P_M]$ and processors are always taken out and put back in the front.

Table II
An example demonstrating SPA2

Task	C_i	T_i	Heavy Task?	Priority
τ_1	0.5	10	no	highest
τ_2	4.5	10	yes	
τ_3	6	10	yes	
τ_4	4	10	no	
τ_5	3	10	no	
τ_6	6	10	yes	
τ_7	3	10	no	lowest

- PQ_{pre} is the list to accommodate pre-assigned processors, initially empty.
- UQ is the list to accommodate the unassigned tasks after Step 1). Initially it is empty, and during Step 1), each task τ_i that is determined not to be pre-assigned will be put into UQ (already in its subtask form τ_i^1).
- $\Psi[1...M]$ is an array, which has the same meaning as in SPA1: each element $\Psi[q]$ in the array $\Psi[1...M]$ denotes the sum of the utilization of tasks that have been assigned to processor P_q.

In the following we use the task set example in Table II with 4 processors to demonstrate how the partitioning algorithm of SPA2 works. For simplicity, we assume $\Theta(N) = 0.7$, then the utilization threshold for light tasks $\Theta(N)/(1 + \Theta(N))$ is around 0.41. The initial state of the data structures are as follows:

- $PQ = [P_1, P_2, P_3, P_4]$
- $PQ_{pre} = \emptyset$
- $UQ = \emptyset$
- $\Psi[1...4] = [0, 0, 0, 0]$

In Step 1) (lines 6 to 15), each task τ_i in τ is visited in increasing index order, i.e., decreasing priority order. If τ_i is a heavy task, we evaluate the following condition (line 7):

$$\sum_{j>i} U_j \leq (|PQ| - 1) \cdot \Theta(N) \qquad (15)$$

in which $|PQ|$ is the number of processors left in PQ so far. A heavy task τ_i is determined to be pre-assigned to a processor if this condition is satisfied. The intuition for this is: If we pre-assign this task τ_i, then there is enough space on the remaining processors to accommodate all remaining lower priority tasks. That way, no lower priority tail subtask will end up on the processor which we assign τ_i to.

In our example, we first visit the first task τ_1^1. It is a light task, so we put it to the front of UQ (line 13). The next task τ_2 is heavy, but Condition (15) with $|PQ| = 4$ is not satisfied, so we put τ_2^1 to the front of UQ. The next task τ_3 is heavy, and Condition (15) with $|PQ| = 4$ is satisfied. Thus, we pre-assign τ_3 to P_1, and put P_1 to the front of PQ_{pre} (lines 8 to 10). τ_4 and τ_5 are both light tasks, so we put them to UQ respectively. τ_6 is heavy, and Condition (15) with $|PQ| = 3$ (P_1 has been taken out from PQ and put into PQ_{pre}) is satisfied, so we pre-assign τ_5 to P_2, and put P_2 to the front of PQ_{pre}. The last task τ_7 is light, so it is put to the front of UQ. So far, the Step 1) phase has been finished, and the state of the data structures is as follows:

- $PQ = [P_3, P_4]$
- $PQ_{pre} = [P_2, P_1]$
- $UQ = [\tau_7^1, \tau_5^1, \tau_4^1, \tau_2^1, \tau_1^1]$
- $\Psi[1...4] = [0.6, 0.6, 0, 0]$

Note that the processors in PQ_{pre} are in decreasing priority order of the pre-assigned tasks on them, and the tasks in UQ are in decreasing priority order.

Step 2) and 3) are both in the while loop of line 16 $\sim$ 35. In Step 2), the remaining tasks (which are now in UQ) are assigned to normal processors (the ones in PQ). Only as soon as all processors in PQ are full, the algorithm enters Step 3), in which tasks are assigned to processors in PQ_{pre} (decision in lines 18 to 22).

The operation of assigning a task τ_i^k (lines 23 to 34) is basically the same as in SPA1. If τ_i^k can be entirely assigned to P_q without task splitting, then $\tau_i^k \mapsto P_q$ and $\Psi[q]$ is updated (lines 24 to 28). If P_q is a pre-assigned processor, P_q is put back to the front of PQ_{pre} (lines 26 to 28), so that it will be selected again in the next loop iteration, otherwise no putting back operation is needed since we never take out elements from PQ, but just select the proper one in it (line 19).

If τ_i^k can not be assigned to P_q entirely, τ_i^k is split into a new τ_i^k and another subtask τ_i^{k+1}, such that P_q becomes full after the new τ_i^k being assigned to it, and then we put τ_i^{k+1} back to UQ (see lines 29 to 33).

Note that there is an important difference between assigning tasks to normal processors and to pre-assigned processors. When tasks are assigned to normal processors, the algorithm always selects the processor with the minimal Ψ (the same as in SPA1). In the contrast, when tasks are assigned to pre-assigned processors, always the processor at the front of PQ_{pre} is selected, i.e., we assign as many tasks as possible to the processor in PQ_{pre} whose pre-assigned task has the lowest priority, until it is full. As will be seen later in the schedulability proof, this particular order of selecting pre-assigned processors, together with the evaluation of Condition (15), is the key to guarantee the schedulability of heavy tasks.

With our running example, the remaining tasks are first assigned to the normal processors P_3 and P_4 in the same way as by SPA1. Thus, $\tau_7^1 \mapsto P_3$, then $\tau_5^1 \mapsto P_4$, then $\tau_4^1 \mapsto P_3$, then τ_2^1 is split into $\tau_2^1 = \langle 4, 10, 10\rangle$ and $\tau_2^2 = \langle 0.5, 10, 6\rangle$, and $\tau_2^1 \mapsto P_4$. So far, all normal processors are full, and the state of the data structures is as follows:

- $PQ = [P_3, P_4]$ (both P_3 and P_4 are full)
- $PQ_{pre} = [P_2, P_1]$
- $UQ = [\tau_2^2, \tau_1^1]$
- $\Psi[1...4] = [0.6, 0.6, 0.7, 0.7]$

Then the remaining tasks in UQ are assigned to the pre-assigned processors. At first $\tau_2^2 \mapsto P_2$, after which P_2 is not full and still at the front of PQ_{pre}. So the next task τ_1^1 is also assigned to P_2. There is no unassigned task any more, so the algorithm terminates.

It is easy to see that any task set below the desired utilization bound can be successfully partitioned by SPA2:

Lemma 8. *Any task set with*

$$U(\tau) \leq \Theta(N)$$

can be successfully partitioned to M processors with SPA2.

After describing the partitioning part of SPA2, we also need to describe the scheduling part. It is the same as SPA1: on each processor the tasks are scheduled by RMS, respecting the precedence relations between the subtasks of a split task, i.e., a subtask is ready for execution as soon as the execution of its preceding subtask has been finished. Note that under SPA2, each body subtask is also with the highest priority on its host processor, which is the same as in SPA1. So we can view the scheduling on each processor as the RMS with a set of independent tasks, in which each subtask's deadline is shortened by the sum of the execution time of all its preceding subtasks.

B. Properties

Now we introduce some useful properties of SPA2.

Lemma 9. *Let τ_i be a heavy task, and there are η pre-assigned tasks with higher priority than τ_i. Then we know*

- *If τ_i is a pre-assigned task, it satisfies*

$$\sum_{j>i} U_j \leq (M - \eta - 1) \cdot \Theta(N) \tag{16}$$

- *If τ_i is not a pre-assigned task, it satisfies*

$$\sum_{j>i} U_j > (M - \eta - 1) \cdot \Theta(N) \tag{17}$$

Proof: The proof directly follows the partitioning algorithm of SPA2. ■

Lemma 10. *Each pre-assigned task has the lowest priority on its host processor.*

Proof: Without loss of generality, we sort all processors in a list Q as follows: we first sort all pre-assigned processors in Q, in decreasing priority order of the pre-assigned tasks on them; then the normal processors follow in Q in an arbitrary order. We use P_x to denote the x^{th} processor in Q. Suppose τ_i is a heavy task pre-assigned to P_q.

τ_i is a pre-assigned task, and the number of pre-assigned task with higher priority than τ_i is $q - 1$, so by Lemma 9 we know the following condition is satisfied:

$$\sum_{j>i} U_j \leq (M - q) \cdot \Theta(N) \tag{18}$$

In the partitioning algorithm of SPA2, normal tasks are assigned to pre-assigned processors only when all normal processors are full, and the pre-assigned processors are selected in increasing priority order of the pre-assigned tasks on them, so we know only when the processors $P_{q+1}...P_M$ are all full, normal tasks can be assigned to processor P_q. The total capacity of processors $P_{q+1}...P_M$ are $(M-q)\cdot\Theta(N)$ (in our algorithms a processor is full as soon as the total utilization on it is $\Theta(N)$), and by (18), we know when we start to assign tasks to P_q, the tasks with lower priority than τ_i all have been assigned to processors $P_{q+1}...P_M$, so all normal tasks (subtasks) assigned to P_q have higher priorities than τ_i. ■

Lemma 11. *Each body subtask has the highest priority on its host processor.*

Proof: Given a body subtask $\tau_i^{b_j}$ assigned to processor P_{b_j}. Since task splitting only occur when a processor is full, and all the normal tasks are assigned in increasing priority order, we know $\tau_i^{b_j}$ has the highest priority among all normal tasks on P_{b_j}. Additionally, by lemma 10 we know that if P_{b_j} is a pre-assigned processor, the pre-assigned task on P_{b_j} also has lower priority than $\tau_i^{b_j}$. So we know P_{b_j} has the highest priority on P_{b_j}. ■

C. Schedulability

By Lemma 11 we know that under SPA2 each body subtask has the highest priority on its host processor, so we know all body subtasks are schedulable.

The scheduling algorithm of SPA2 is still RMS, and the deadline of a non-split task still equals to its period, so the schedulability of non-split tasks can be proved in the same way as in SPA1 (Lemma 3).

In the following we will prove the schedulability of tail subtasks. Suppose τ_i is split into B body subtasks and one tail subtask. Recall that we use $\tau_i^{b_j}, j\in[1,B]$ to denote the j^{th} body subtask of τ_i, and τ_i^t to denote τ_i's tail subtask. X^t, Y^t and X^{b_j} are defined the same as in Section IV-B.

First we recall Lemma 5, which is used to prove the schedulability of tail subtasks in SPA1: if a tail subtask τ_i^t satisfies

$$Y^t\cdot T_i/\triangle_i^t + V_i^t \leq \Theta(N) \tag{19}$$

τ_i^t can meet its deadline. This conclusion also holds for SPA2, since the scheduling algorithm on SPA2 is also RMS, which is only the relevant property required by the proof of Lemma 5. So proving the schedulability of tail subtasks is reduced to proving Condition (19) for tail subtasks under SPA2.

We call τ_i^t a *tail-of-heavy* if τ_i is heavy, otherwise a *tail-of-light*. In the following we prove Condition (19) for τ_i^t in three cases:

1) τ_i^t is a tail-of-light, and P_t is a normal processor,
2) τ_i^t is a tail-of-light, and P_t is a pre-assigned processor,
3) τ_i^t is a tail-of-heavy.

Case 1) can be proved in the same way as in SPA1, since both the partitioning and scheduling algorithm of SPA2 *on normal processors* are the same as SPA1. Actually one can regard the partitioning and scheduling of SPA2 on normal processors as the partitioning and scheduling with a subset of tasks (those are assigned to normal processors) on a subset of processors (normal processors) of SPA1. So the schedulability of τ_i^t in this case can be proved by exactly the same reasoning as for Lemma 6.

Now we prove Case 2), where τ_i^t is a tail-of-light, and P_t is a pre-assigned processor.

Lemma 12. *Suppose τ_i^t is a tail-of-light assigned to a pre-assigned processor P_t under* SPA2. *We have*

$$Y^t\cdot T_i/\triangle_i^t + V_i^t \leq \Theta(N)$$

Proof: By Lemma 10 we know τ_i^t has higher priority than the pre-assigned task of P_t, so X^t is no smaller than the utilization of this pre-assigned task. And since a pre-assigned task must be heavy, we have

$$X^t > \frac{\Theta(N)}{1+\Theta(N)} \tag{20}$$

On the other hand, since τ_i is light, we know

$$\frac{C_i}{T_i} \leq \frac{\Theta(N)}{1+\Theta(N)}$$

We use c_i^B to denote the total execution time of all τ_i's body tasks. Since $c_i^B < C_i$ and $\Theta(N) < 1$, we have

$$c_i^B < \frac{1}{1+\Theta(N)}\cdot T_i$$

$$\Leftrightarrow T_i\Big(1-\frac{1}{1+\Theta(N)}\Big) < T_i - c_i^B$$

$$\Leftrightarrow \frac{T_i}{T_i - c_i^B}\Big(\Theta(N) - \frac{\Theta(N)}{1+\Theta(N)}\Big) < \Theta(N)$$

$$\Leftrightarrow \frac{T_i}{\triangle_i^t}\Big(\Theta(N) - \frac{\Theta(N)}{1+\Theta(N)} - U_i^t\Big) + V_i^t < \Theta(N) \tag{21}$$

By (21) and (20) we have

$$\frac{T_i}{\triangle_i^t}(\Theta(N) - X^t - U_i^t) + V_i^t < \Theta(N)$$

and since the total utilization on each processor is bounded by $\Theta(N)$, i.e.,

$$Y^t \leq \Theta(N) - X^t - U_i^t$$

finally we have $Y^t\cdot T_i/\triangle_i^t + V_i^t < \Theta(N)$. ■

Now we prove Case 3), where τ_i^t is a tail-of-heavy. Note that in this case P_t can be either a pre-assigned or a normal processor.

Lemma 13. *τ_i^t is the tail subtask of a normal heavy task τ_i, then we have*

$$Y^t\cdot T_i/\triangle_i^t + V_i^t \leq \Theta(N)$$

Proof: By the property in Lemma 9 concerning normal heavy tasks we know τ_i satisfies the condition

$$\sum_{j>i} U_j > (M - \eta - 1) \cdot \Theta(N)$$

in which η is the number of pre-assigned tasks with higher priority than τ_i.

We use $\mathcal{M}$ to denote the set of all processors, so $|\mathcal{M}| = M$, and use $\mathcal{H}$ to denote the set of the pre-assigned processors on which the pre-assigned tasks' priorities are higher than τ_i, so $|\mathcal{H}| = \eta$, so we have:

$$\sum_{j>i} U_j > (M - |\mathcal{H}| - 1) \cdot \Theta(N) \tag{22}$$

By Lemma 10 we know any normal task assigned to a pre-assigned processor has higher priority than the pre-assigned task of this processor. Therefore, τ_i's body and tail subtasks are all assigned to processors in $\mathcal{M} \setminus \mathcal{H}$. Moreover, when we start to assign τ_i, all tasks with lower priority than τ_i have already been assigned (pre-assigned) to processors in $\mathcal{M} \setminus \mathcal{H}$, since pre-assigned tasks have already been assigned before dealing with the normal tasks, and all normal tasks are assigned in increasing priority order.

We use $\mathcal{K}$ to denote the set of processors in $\mathcal{M} \setminus \mathcal{H}$ that contain neither τ_i's body nor tail subtask, and for each processor $P_k \in \mathcal{K}$ we use X^k to denote the total utilization of the tasks with lower priority than τ_i assigned to P_k. Then we have

$$X^t + \sum_{j \in [1,B]} X^{b_j} + \sum_{k \in [1,|\mathcal{K}|]} X^k = \sum_{j>i} U_j$$

Since $|\mathcal{K}| = M - |\mathcal{H}| - (B+1)$, and $\forall P_k \in \mathcal{K}, X_k \leq \Theta(N)$, we have

$$X^t + \sum_{j \in [1,B]} X^{b_j} \geq \sum_{j>i} U_j - (M - |\mathcal{H}| - (B+1)) \cdot \Theta(N) \tag{23}$$

By Inequalities (22) and (23) we have

$$X^t + \sum_{j \in [1,B]} X^{b_j} > B \cdot \Theta(N) \tag{24}$$

Now we look at processor P_t, the total utilization on which is bounded by $\Theta(N)$, so we have:

$$Y^t \leq \Theta(N) - X^t - U_i^t \tag{25}$$

By (24) and (25) we have

$$Y^t \leq \Theta(N) - (B \cdot \Theta(N) - \sum_{j \in [1,B]} X^{b_j}) - U_i^t$$

and since $U_i^t + \sum_{j \in [1,B]} U_i^{b_j} = U_i$, we have

$$Y^t \leq \Theta(N) - B \cdot \Theta(N) - U_i + \left(\sum_{j \in [1,B]} X^{b_j} + \sum_{j \in [1,B]} U_i^{b_j} \right) \tag{26}$$

Since each body task has the highest priority on its host processor, and the total utilization of any processor containing a body subtask is $\Theta(N)$, we have

$$\sum_{l \in [1,B]} X^{b_l} + \sum_{l \in [1,B]} U_i^{b_l} = B \cdot \Theta(N) \tag{27}$$

By (26) and (27) we have

$$Y^t \leq \Theta(N) - U_i$$

$$\Leftrightarrow Y^t \cdot T_i / \triangle_i^t + V_i^t \leq (\Theta(N) - U_i) \cdot T_i / \triangle_i^t + V_i^t$$

By applying $U_i = C_i / T_i$ and $V_i^t = c_i^t / \triangle_i^t$ to the RHS of the above inequality, we get

$$Y^t \cdot T_i / \triangle_i^t + V_i^t \leq \Theta(N) \cdot T_i / \triangle_i^t - C_i / \triangle_i^t + c_i^t / \triangle_i^t \tag{28}$$

We use c_i^B to denote the sum of the execution time of all τ_i's body subtasks, so we have $c_i^t + c_i^B = C_i$ and $\triangle_i^t = T_i - c_i^B$. We apply these to the RHS of (28) and get

$$Y^t \cdot T_i / \triangle_i^t + V_i^t \leq \frac{T_i \Theta(N) - c_i^B}{T_i - c_i^B} \tag{29}$$

Since $\Theta(N) < 1$, we have

$$c_i^B > c_i^B \cdot \Theta(N)$$

$$\Leftrightarrow T_i \cdot \Theta(N) - c_i^B < T_i \cdot \Theta(N) - c_i^B \cdot \Theta(N)$$

$$\Leftrightarrow \frac{T_i \cdot \Theta(N) - c_i^B}{T_i - c_i^B} < \Theta(N) \tag{30}$$

So by Inequalities (29) and (30) we have

$$Y^t \cdot T_i / \triangle_i^t + V_i^t < \Theta(N)$$

■

D. Utilization Bound

Now we have known that any task set τ with $U(\tau) \leq \Theta(N)$ can be successfully partitioned on M processors by SPA2 (Lemma 8). In last subsection, we have shown that under the scheduling algorithm of SPA2, the body subtasks are schedulable since they are always with the highest priority level on their host processors; the non-split tasks are also schedulable since the utilization on each processor is bounded by $\Theta(N)$. The schedulability for the tail subtasks is proved by case distinction, in which the schedulability for the *light* tail subtasks on *normal* processors can be proved by the same reasoning as for Lemma 6, for the *light* tail subtasks on *pre-assigned* processors is proved by Lemma 12, and for the *heavy* tail subtasks is proved by Lemma 13. So we have the following theorem:

Theorem 3. *τ is schedulable by* SPA2 *on M processors if*

$$U(\tau) \leq \Theta(N)$$

So $\Theta(N)$ is the utilization bound of SPA2 for any task set.

For the same reason as presented at the end of Section IV-B, we can use $\Theta(N^*)$, the the maximal number of tasks (subtasks) assigned to on each processor, to replace $\Theta(N)$ in Theorem 3.

E. Task Splitting Overhead

With the algorithms proposed in this paper, a task could be split into more than two subtasks. However, since the task splitting only occurs when a processor is full, for any task set that is schedulable by SPA2, the number of task splitting is at most $M-1$, which is the same as in previous semi-partitioned fixed-priority scheduling algorithms [17], [14], [15], and as shown in case studies conducted in [17], this overhead can be expected to be negligible on multi-core platforms.

VI. Conclusions and Future Work

In this paper, we have developed a semi-partitioned fixed-priority scheduling algorithm for multiprocessor systems, with the well-known Liu and Layland's utilization bound for RMS on single processors. The algorithm enjoys the following property. If the utilization bound is used for the schedulability test, and a task set is determined schedulable by fixed-priority scheduling on a single processor of speed M, it is also schedulable by our algorithm on M processors of speed 1 (under the assumption that each task's execution time on the processors of speed 1 is still smaller than its deadline). Note that the utilization bound test is only sufficient but not necessary. As future work, we will challenge the problem of constructing algorithms holding the same property with respects to the exact schedulability analysis.

References

[1] J. Anderson, V. Bud, and U.C. Devi. An edf-based scheduling algorithm for multiprocessor soft real-time systems. In *Euromicro Conference on Real-Time Systems (ECRTS)*, 2005.

[2] J. Anderson and A. Srinivasan. Mixed pfair/erfair scheduling of asynchronous periodic tasks. In *Journal of Computer and System Sciences*, 2004.

[3] B. Andersson. Global static priority preemptive multiprocessor scheduling with utilization bound 38%. In *International Conference on Principles of Distributed Systems (OPODIS)*, 2008.

[4] B. Andersson, S. Baruah, and J. Jonsson. Static priority scheduling on multiprocessors. In *IEEE Real-Time Systems Symposium (RTSS)*, 2001.

[5] B. Andersson and K. Bletsas. Sporadic multiprocessor scheduling with few preemptions. In *Euromicro Conference on Real-Time Systems (ECRTS)*, 2008.

[6] B. Andersson, K. Bletsas, and S. Baruah. Scheduling arbitrary-deadline sporadic task systems multiprocessors. In *IEEE Real-Time Systems Symposium (RTSS)*, 2008.

[7] B. Andersson and J. Jonsson. The utilization bounds of partitioned and pfair static-priority scheduling on multiprocessors are 50%. In *Euromicro Conference on Real-Time Systems (ECRTS)*, 2003.

[8] B. Andersson and E. Tovar. Multiprocessor scheduling with few preemptions. In *IEEE conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2006.

[9] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel. Proportionate progress: A notion of fairness in resource allocation. In *Algorithmica*, 1996.

[10] John Carpenter, Shelby Funk, Philip Holman, Anand Srinivasan, James Anderson, and Sanjoy Baruah. *A Categorization of Real-Time Multiprocessor Scheduling Problems and Algorithms*. 2004.

[11] U. Devi and J. Anderson. Tardiness bounds for global edf scheduling on a multiprocessor. In *IEEE Real-Time Systems Symposium (RTSS)*, 2005.

[12] S. Kato and N. Yamasaki. Real-time scheduling with task splitting on multiprocessors. In *IEEE conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2007.

[13] S. Kato and N. Yamasaki. Portioned edf-based scheduling on multiprocessors. In *International Conference on Embedded Software (EMSOFT)*, 2008.

[14] S. Kato and N. Yamasaki. Portioned static-priority scheduling on multiprocessors. In *International Parellel and Distributed Processing Symposium (IPDPS)*, 2008.

[15] S. Kato and N. Yamasaki. Semi-partitioned fixed-priority scheduling on multiprocessors. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2009.

[16] S. Kato, N. Yamasaki, and Y. Ishikawa. Semi-partitioned scheduling of sporadic task systems on multiprocessors. In *Euromicro Conference on Real-Time Systems (ECRTS)*, 2009.

[17] K. Lakshmanan, R. Rajkumar, and J. Lehoczky. Partitioned fixed-priority preemptive scheduling for multi-core processors. In *Euromicro Conference on Real-Time Systems (ECRTS)*, 2009.

[18] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. In *Journal of the ACM*, 1973.

[19] D. Oh and T. P. Baker. Utilization bounds for n-processor rate monotone scheduling with static processor assignment. In *Real-Time Systems*, 1998.