To appear in Proc. of the Third Workshop on Parallel and Distributed Real-Time Systems, 1995

Bounding the End-to-End Response Time in Multiprocessor Real-Time Systems

Jun Sun Jane W.S. Liu
Department of Computer Science,
University of Illinois, Urbana-Champaign
Urbana, IL 61801
{jsun, janeliu}@cs.uiuc.edu

Abstract

In a multiprocessor real-time system, a task may execute in turn on several processors before it completes. The task may have different priorities on different processors and execute on some processors more than once. Existing techniques for bounding the response times of such tasks are not effective. In this paper, we present a method to compute tighter upper bounds on their response times.

1 Introduction

In a real-time system, every time-critical task must have a bounded response time. The schedulability of the task can then be verified by comparing a known upper bound of its response time with its relative deadline. If it is less than the relative deadline, the task is schedulable; otherwise it may not. In a multiprocessor real-time system, a task may execute sequentially on several processors before it completes. Such a task is called an *end-to-end task*. An end-to-end task is *recurrent* if it executes on some processors more than once. For example, in a data acquisition system, the field processor executes to collect data; the communication link (also modeled as a processor) transmits the data to the control processor; the control processor processes the data. In response to the data the control processor sends a command back to the field processor, which formats the command and forwards it to an actuator attached to it. We model the entire computation/communication process from when the data is sampled till the command is sent to an actuator as an end-to-end task. This task is recurrent because the task executes twice on the field processor and the communication links.

An end-to-end task can be viewed logically as a chain of subtasks; each subtask is a continuous execution thread of the task on a processor. A recurrent end-to-end task may have more than one subtask executing on a processor. We refer to an end-to-end task as the *parent task* of its subtasks and a subtask as a *sibling subtask* of other subtasks with the same parent task. An end-to-end task is periodic if its execution pattern repeats periodically. In this paper, we focus on periodic end-to-end tasks. Unless stated otherwise, by a task we mean a periodic end-to-end task.

The precedence constraints of subtasks can be enforced by controlling the release times of the subtasks. In a *dynamic system*, each subtask is *dynamically released* immediately after its immediate predecessor completes. In a *static system*, every subtask is released periodically according to its own period and phase. Some method, such as the phase modification technique [1], is used to adjust the phases of subtasks such that when a subtask is released its immediate predecessor is surely completed. In this case, we say that the subtasks are *statically released*.

To the best of our knowledge, existing schedulability analysis techniques can give us only loose upper bounds of the response times of recurrent end-to-end tasks. Rajkumar *et al.* extended the Priority Ceiling Protocol to multiprocessor systems [2]. In their model, each task has a host processor on which most of the computation of that task takes place, and the task has a fixed priority on its host processor. The delay of the execution incurred when the task executes on processors other than its host processor is counted as the blocking time. Consequently existing methods [3, 4] to determine the worst-case response times of periodic tasks on a single processor can be applied. This approach is applicable to some dynamic systems. However it does not give satisfactory upper bounds on response times of end-to-end tasks in general.

In a static system, it is possible to treat all subtasks as independent periodic tasks. Existing methods [3, 4] to determine upper bounds of the response times of periodic tasks on a single processor can be applied to determine upper bounds of the response times of individual subtasks. The sum of these upper bounds gives an upper bound to the response time of their parent task [5]. However, by treating subtasks on each processor as independent periodic tasks, the precedence relation among subtasks is not exploited, and the bounds of the response times based on this assumption can be very pessimistic, especially for recurrent tasks.

This paper presents a method for computing tighter upper bounds on the response times of end-to-end tasks in static systems. Section 2 formally defines the problem solved by the method. Section 3 gives a basic algorithm to compute an upper bound of the response time of a recurrent task. Section 4 presents several improvements over the basic algorithm. Section 5 concludes our paper.

2 Problem Formulation

In our model, the workload on a multiprocessor system consists of a set $\{T_i\}$ of end-to-end tasks, each of which is a periodic task with period p_i, phase f_i, execution time τ_i, and relative deadline D_i. In this paper, we assume that the relative deadline of a task is less than or equal to its period, i.e., $D_i \leq p_i$.

A task T_i is a chain of subtasks $\{T_{i,j}\}$. Each subtask $T_{i,j}$ is one continuous execution thread of T_i on one processor. In other words, if the execution thread of T_i migrates from processor to processor n times, T_i has $(n+1)$ subtasks. We assume that the migration pattern of the execution thread of every task is fixed. Moreover we know (a) the number n_k of subtasks in T_k, (b) the execution time $\tau_{i,j}$ of each subtask $T_{i,j}$, and (c) the processor on which each subtask executes. We also assume that each subtask is assigned a fixed priority, $\phi_{i,j}$, which may or may not be the same as the priorities of other subtasks in T_i. Subtasks on each processor are scheduled preemptively in a priority-driven manner. Unless stated otherwise, subtasks of different tasks are independent.

We confine our attention to the case where the subtasks are statically released according to the phase modification technique. Let $c_{i,j}$ denote an upper bound of the response time of $T_{i,j}$. According to the phase modification technique, a subtask $T_{i,j}$ is released periodically, where the period $p_{i,j}$ of $T_{i,j}$ is equal to p_i, and the phase $f_{i,j}$ of $T_{i,j}$ is equal to $f_i + \sum_{k=1}^{j-1} c_{i,j}$. Clearly with this phase modification the first subtask $T_{i,1}$ is released at f_i, $f_i + p_i$, ... Each instance of $T_{i,j}$ for $j > 1$ is released at a time when the corresponding instance of its immediate predecessor $T_{i,j-1}$ must have completed.

The problem we address here can be stated as follows: Given the parameters of all the subtasks in each task and the priorities assigned to them, we want to determine an upper bound c_i of the response time of task T_i that is as tight as possible.

3 The Basic Algorithm

Let $H_{i,j}$ denote the set of subtasks that (a) are on the same processor as $T_{i,j}$, (b) are of different parent tasks, and (c) have priorities higher than or equal to $T_{i,j}$. Let $L_{i,j}$ denote the set of subtasks that (a) are on the same processor as $T_{i,j}$, (b) are of different parent tasks, and (c) have priorities lower than $T_{i,j}$. Let $\mathcal{H}_{i,j}$ denote the set of tasks that have subtasks in $H_{i,j}$.

A time demand is generated by a task when one of its subtask is released. This time demand can delay the execution and completion of a subtask $T_{i,j}$ if the released subtask is in $H_{i,j}$. To account for the time demand generated by tasks in $\mathcal{H}_{i,j}$ that can delay the execution and completion of $T_{i,j}$, we introduce the *Per-Task Time Demand* (PTTD) function, $M_k^{i,j}(t)$, of each task T_k in $\mathcal{H}_{i,j}$ with respect to $T_{i,j}$. Let $M_k^{i,j}(t_0, t)$ denote the time demand generated by T_k that can delay the execution of $T_{i,j}$ during the time interval $[t_0, t_0+t)$. The PTTD function $M_k^{i,j}(t)$ is an upper bound of $M_k^{i,j}(t_0, t)$ for any time instant t_0, i.e.,

$$M_k^{i,j}(t_0, t) \leq M_k^{i,j}(t) \tag{1}$$

for any time instant t_0. Since only the subtasks of T_k that are in $H_{i,j}$ can delay the execution of $T_{i,j}$, a simple way to bound $M_k^{i,j}(t_0, t)$ is to treat the subtasks of T_k in $H_{i,j}$ as independent subtasks and to let them all be released at time t_0, which is a *critical instant* [3]. In other words, an upper bound is

$$M_k^{i,j}(t) = \sum_{T_{k,l} \in H_{i,j}} \left\lceil \frac{t}{p_k} \right\rceil \tau_{k,l} \tag{2}$$

Eq.(2) gives a loose bound of $M_k^{i,j}(t_0, t)$ since subtasks of T_k are not independent. In the next section we will present improved methods to compute the PTTD function that yields tighter bound.

If $T_{i,j}$ does not have any sibling subtask which executes on the same processor as itself, the time demand analysis method, proposed by Lehoczky [3], can

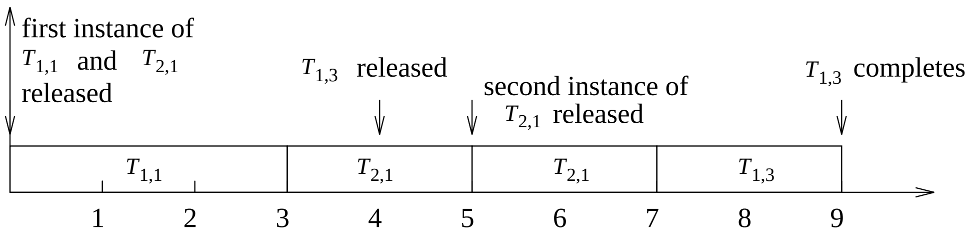

Figure 1: Schedule on P_1 When $T_{1,1}$ and $T_{2,1}$ Are Released at Time 0

be extended to determine an upper bound $c_{i,j}$ of the response time of $T_{i,j}$. According to the extended method, we first compute the time demand function, $W_{i,j}(t)$, which is equal to the execution time of $T_{i,j}$ plus the sum of $M_k^{i,j}(t)$ for every T_k in $\mathcal{H}_{i,j}$, i.e.,

$$W_{i,j}(t) = \tau_{i,j} + \sum_{T_k \in \mathcal{H}_{i,j}} M_k^{i,j}(t) \tag{3}$$

for $t \leq p_{i,j}$. Obviously, if an instance $T_{i,j}$ is released at time t_0, an upper bound $c_{i,j}$ of the response time of $T_{i,j}$ is the first time instant when the time demand $W_{i,j}(t)$ is met by time supply t, i.e.,

$$c_{i,j} = \min\{t > 0 | t = W_{i,j}(t)\} \tag{4}$$

We can use an efficient iterative method to compute the right hand side of (4) if $c_{i,j} \leq p_{i,j}$ [4]. According to this method, S_0 is set to $W_{i,j}(0)$ and S_k for $k = 1, 2, \ldots$ is computed by the following equation.

$$S_k = W_{i,j}(S_{k-1}) \tag{5}$$

According to the Eq.(3) and (5), S_k is monotonically increasing. If S_k converges to a number S^* less than or equal to $p_{i,j}$ after a finite number of iterations, the number S^* is returned as the value of $c_{i,j}$. If S_k fails to converge to S^* less than $p_{i,j}$, we return ∞ as the value of $c_{i,j}$.

If $T_{i,j}$ have some sibling subtasks executing on the same processor as itself, the function $W_{i,j}(t)$ computed by Eq.(3) may no longer be correct in determining an upper bound of the response time of $T_{i,j}$. More specifically, let $SH_{i,j}$ denote the set of subtasks each of which (a) is a sibling subtask of $T_{i,j}$, (b) executes on the same processor as $T_{i,j}$, and (c) has priority higher than or equal to $T_{i,j}$. If $SH_{i,j}$ is not empty, subtasks in $SH_{i,j}$ may interfere with the execution of $T_{i,j}$ in an intricate way, as illustrated by the following example, Example 1. There are two tasks and two processors in this system. The parameters of the subtasks are shown in Table 1.

By definition, $H_{1,3} = \{T_{2,1}\}$, $\mathcal{H}_{1,3} = \{T_2\}$. According to Eq.(2), $M_2^{1,3}(t)$ is equal to $2\lceil \frac{t}{5} \rceil$. According to

T_i	proc	$\phi_{i,j}$	$p_{i,j}$	$\tau_{i,j}$
$T_{1,1}$	P_1	1	20	3
$T_{1,3}$	P_1	5	20	2
$T_{2,1}$	P_1	3	5	2
$T_{1,2}$	P_2	2	20	1

Table 1: Subtask Parameters for Example 1

Eq.(3), $W_{1,3}(t)$ is equal to $2 + 2\lceil \frac{t}{5} \rceil$. After applying the iterative method based on Eq.(5), we obtain $c_{1,3}$ equal to 4. In this case, however, $SH_{1,3}$ is not empty, and it contains $T_{1,1}$. Suppose both $T_{1,1}$ and $T_{2,1}$ have phases equal to 0. Since $c_{1,1}$ is equal to 3 and $c_{1,2}$ is equal to 1, according to the phase modification technique, $T_{1,3}$ should be released 4 time units after $T_{1,1}$ is released, i.e., $f_{1,3} = 4$. As shown in the schedule in Figure 1, the response time of $T_{1,3}$ is 5, greater than the obtained upper bound $c_{1,3}$.

One way to account for this interference is to treat the subtasks in T_i as independent subtasks as well. The subtasks in $SH_{i,j}$ delay the completion of $T_{i,j}$ by the maximum amount when they are released at the same time as $T_{i,j}$ is released. Consequently we can bound this delay by introducing in the right hand side of Eq.(3) an extra term $\Delta_{i,j}$, which is equal to the sum of the execution times of the subtasks in $SH_{i,j}$, i.e.,

$$\Delta_{i,j} = \sum_{T_{i,k} \in SH_{i,j}} \tau_{i,k} \tag{6}$$

Rather than the time demand function $W_{i,j}(t)$, we use instead

$$W'_{i,j}(t) = \tau_{i,j} + \Delta_{i,j} + \sum_{T_k \in \mathcal{H}_{i,j}} M_k^{i,j}(t) \tag{7}$$

for $t \leq p_{i,j}$.

We call the algorithms that compute the response times based on the PTTD functions the PTTDF algorithms. All PTTDF algorithms share the same high-level structure, as described in Figure 2. They may use different methods to compute the PTTD functions. We call the algorithm based on the Eq.(2), (6) and (7) the basic PTTDF algorithm.

The Structure of the PTTDF Algorithms

Input :

1. Task set $\{T_i\}$. For each task T_i, the period p_i and the execution time τ_i;
2. Subtask set $\{T_{i,j}\}$ associated with the task set $\{T_i\}$. For each subtask $T_{i,j}$, the execution time $\tau_{i,j}$, the priority $\phi_{i,j}$, and the processor $T_{i,j}$ executes on.

Output : An upper bound c_i of the response time of T_i.

Algorithm :

1. For every subtask $T_{i,j}$
   (a) Compute $\mathcal{H}_{i,j}$;
   (b) For every task T_k in $\mathcal{H}_{i,j}$
   Compute the PTTD function $M_k^{i,j}(t)$;
   (c) Compute $SH_{i,j}$ and $\Delta_{i,j}$;
   (d) Compute $W'_{i,j}(t)$ given by Eq.(7);
   (e) Compute $c_{i,j}$ using the iterative method based on Eq.(5);
2. For every task T_i
   $c_i = \sum_{j=1}^{n_i} c_{i,j}$

Figure 2: Pseudo-Code of the PTTDF Algorithms

4 Improvements Over the Basic PTTDF Algorithm

There are two ways to improve Eq.(2) to obtain tighter bounds of $M_k^{i,j}(t_0,t)$ and thus tighter bounds of the response times. The PTTD function $M_k^{i,j}(t)$ computed by Eq.(2) assumes that the subtasks of each task in $\mathcal{H}_{i,j}$ are independent. The actual time demand may be less than the sum in the right hand side of Eq.(2) because of the precedence constraints among subtasks. The example, Example 2, described in Table 2 illustrates this point.

According to Eq.(2), $M_1^{2,1}(t)$ is equal to $7\lceil\frac{t}{15}\rceil$. $SH_{2,1}$ is empty, and hence $\Delta_{2,1}=0$. The time demand functions $W_{2,1}(t)$ is equal to $2+7\lceil\frac{t}{15}\rceil$. Applying the iterative method, we find that $c_{2,1}$ is equal to

$T_{i,j}$	proc	$p_{i,j}$	$\tau_{i,j}$	$\phi_{i,j}$
$T_{1,1}$	P_1	15	3	3
$T_{1,3}$	P_1	15	4	1
$T_{2,1}$	P_1	8	2	5
$T_{1,2}$	P_2	15	3	3
$T_{1,4}$	P_2	15	3	3

Table 2: Subtask Parameters for Example 2

∞. Task T_2 is not schedulable; so is the system.

However, subtask $T_{1,1}$ and $T_{1,3}$ can never be released at the same time. Moreover, since the phases of the subtasks are adjusted according to the phase modification technique, subtask $T_{1,3}$ is always released $(c_{1,1}+c_{1,2})$ time units after $T_{1,1}$ is released. Because $c_{1,1}\geq\tau_{1,1}$ and $c_{1,2}\geq\tau_{1,2}$, we can be sure that subtask $T_{1,3}$ can never be released earlier than $(\tau_{1,1}+\tau_{1,2})$ time units after $T_{1,1}$. If the phase of $T_{1,1}$ is 0 and the phase of $T_{1,3}$ is pessimistically set to $\tau_{1,1}+\tau_{1,2}$, the time demand of T_1 with respect to $T_{2,1}$ is shown as the stair-case function in Figure 3(a). On the other hand, since the relative deadline of a task is less than its period, if T_1 is schedulable, the next instance of $T_{1,1}$ will not be released before the current instance of $T_{1,4}$ completes. We can therefore conclude that, if T_1 is schedulable, the the current instance of subtask $T_{1,1}$ can never be released earlier than $(\tau_{1,3}+\tau_{1,4})$ time units after the previous instance of $T_{1,3}$ is released. The time demand of T_1 with respect to $T_{2,1}$, when the phases of $T_{1,3}$ and $T_{1,1}$ are 0 and $(\tau_{1,3}+\tau_{1,4})$, respectively, is shown in Figure 3(b). Let $M_{1,1}^{2,1}(t)$ denote the stair-case function in Figure 3(a), and $M_{1,3}^{2,1}(t)$ denote the one in Figure 3(b). For any phase assignments to subtasks in T_1 and at any time instant t, the amount of processor time demanded by T_1 on processor P_1 during the time interval $[0,t)$ is less than either $M_{1,1}^{2,1}(t)$ or $M_{1,3}^{2,1}(t)$, or both. Hence, we can bound $M_1^{2,1}(0,t)$ by $\max\{M_{1,1}^{2,1}(t), M_{1,3}^{2,1}(t)\}$ as shown in Figure 3(c). The time demand function $W'_{2,1}(t)$ based on this tighter bound is shown in dotted line in Figure 3(c) as well, and the upper bound $c_{2,1}$ is obtained equal to 6. $T_{2,1}$ is schedulable. The pseudo-code for this improved method to compute the PTTD function is shown in Figure 4.

The second improvement to obtain the tighter PTTD function $M_k^{i,j}(t)$ is to account the effect caused subtask $T_{k,l}$ in $L_{i,j}$, as illustrated by the example in Figure 5. In this example, Example 3, the subtasks of T_1 that execute on the same processor as $T_{i,j}$ are denoted as the solid line segments, and the other subtasks of T_1 are denoted by dashed line segments. The

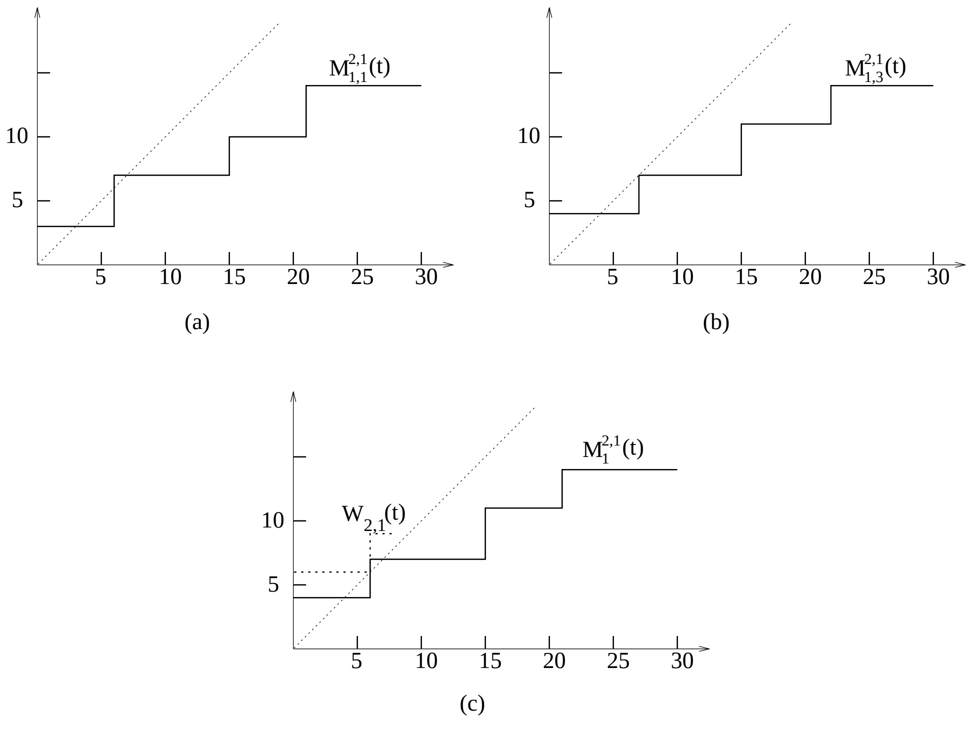

Figure 3: A Simple System

Improved method to compute the PTTD Functions

Input :

1. Subtask $T_{i,j}$.
2. Task T_k in $\mathcal{H}_{i,j}$ and its subtasks $T_{k,l}$.

Output : $M_k^{i,j}(t)$

Algorithm :

1. For every $T_{k,l}$ in $H_{i,j}$
   (a) Set the phase $f_{k,l}$ of $T_{k,l}$ equal to 0.
   (b) If $l \neq 1$, set $f_{k,1}$ equal to $f_{k,n_k} + \tau_{k,n_k}$.
   (c) Set $f_{k,m}$ equal to $f_{k,m-1} + \tau_{k,m-1}$ if $m \neq l$ and $m \neq 1$.
   (d) Compute the total time $M_{k,l}^{i,j}(t)$ demanded in time interval $[0, t)$ by all subtasks $T_{k,l}$ that are in $H_{i,j}$.
2. Return $\max\{M_{k,l}^{i,j}(t)\}$ for all $T_{k,l}$ in $H_{i,j}$ as the PTTD function $M_k^{i,j}(t)$.

Figure 4: Pseudo-Code of the Improved Method to Bound the PTTD Functions

length of each segment represents the processing time of its corresponding subtask, and the height of the line reflects its priority. The period of T_1 is 50.

According to the first improvement to compute the PTTD functions, function $M_{1,3}^{i,j}(t)$ is depicted in Figure 6(a), and the function $M_{1,9}^{i,j}(t)$ is depicted in Figure 6(b). The PTTD function $M_1^{i,j}(t)$ thus obtained is shown as the stair-case function in Figure 6(c). However, if $T_{i,j}$ is preempted by $T_{1,3}$, $T_{1,5}$ will complete later than $T_{i,j}$ because it is released later than $T_{i,j}$ and has a priority lower than $T_{i,j}$. Since the phases of all the subtasks are adjusted according the phase modification technique, the current instance of $T_{1,6}$ will be released after the current instance of $T_{1,5}$ completes, which implies that the current instance of $T_{1,6}$ will be released after $T_{i,j}$ completes. Obviously the current instance of $T_{1,9}$ will be released after $T_{i,j}$ completes also. In other words, if an instance of $T_{i,j}$ is preempted by $T_{1,3}$, it will not be preempted by $T_{1,9}$ before it completes.

On the other hand, suppose that $T_{i,j}$ is preempted by $T_{1,9}$. If T_1 is schedulable and its relative deadline is equal to its period, the next instance of $T_{1,1}$ will be released after the current instance of $T_{1,9}$ completes. In this case, the next instance of $T_{1,1}$, which is in $L_{i,j}$, will be released after $T_{i,j}$ is released, and hence it will com-

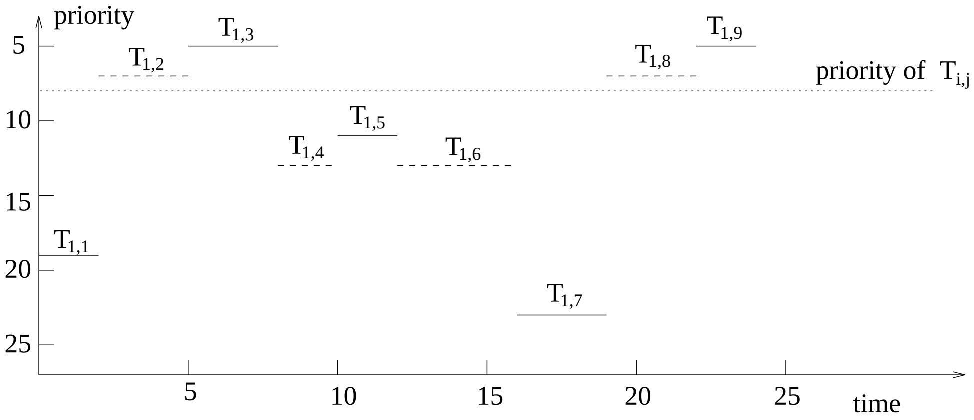

Figure 5: Subtasks of T_1 in Example 3

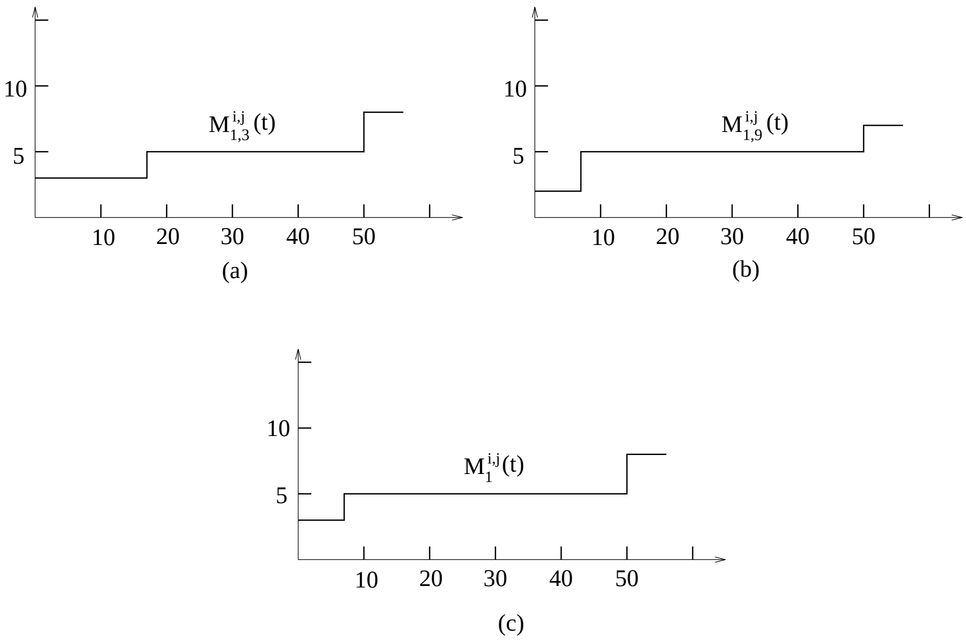

Figure 6: Time Demand Curves of T_1 in Example 3

Second improved method to bound the PTTD Functions

Input :

1. Subtask $T_{i,j}$.
2. Task T_k in $\mathcal{H}_{i,j}$ and its subtasks $T_{k,l}$.

Output : $M_k^{i,j}(t)$

Algorithm :

1. Let H' denote the set of subtasks $T_{k,l}$ in $H_{i,j}$ that do not have a predecessor subtask in $L_{i,j}$.
2. For every $T_{k,l}$ in $H_{i,j}$
   (a) Set the phase $f_{k,l}$ of $T_{k,l}$ equal to 0.
   (b) If $l \neq 1$, set $f_{k,1}$ equal to $f_{k,n_k} + \tau_{k,n_k}$.
   (c) Set $f_{k,m}$ equal to $f_{k,m-1} + \tau_{k,m-1}$ if $m \neq l$ and $m \neq 1$.
   (d) Let t' be the release time of the first instance of some $T_{k,l}$ in $L_{i,j}$ if T_k has subtasks in $L_{i,j}$. Otherwise let t' be ∞. $M_{k,l}^{i,j}(t)$ is equal to the time demand generated by all the subtasks of T_k that are in $H_{i,j}$ during the time interval $[0, t)$ for $0 \leq t \leq t'$; $M_{k,l}^{i,j}(t)$ is equal to $M_{k,l}^{i,j}(t')$ plus the time demand generated by subtasks in H' during the time interval $[t', t)$ for $t > t'$.
3. Return $\max\{M_{k,l}^{i,j}(t)\}$ for all $T_{k,l}$ in $H_{i,j}$ as the PTTD function $M_k^{i,j}(t)$.

Figure 7: Pseudo-Code of the Second Improved Method to Bound the PTTD Functions

plete after $T_{i,j}$. Based on a similar argument as the above case, we can conclude that if $T_{i,j}$ is preempted by $T_{1,9}$ it will not be preempted by $T_{1,3}$ before it completes. In summary, because of the existence of $T_{1,1}$ and $T_{1,5}$, $T_{i,j}$ will be preempted by no more than one instance of $T_{1,3}$ or $T_{1,9}$, i.e., $M_1^{i,j}(t) = \max\{2, 3\} = 3$.

To account the effect caused by the subtasks in $L_{i,j}$, we introduce a new subtask set H' before Step 1 in the first improved method to bound the PTTD functions in Figure 4. Let H' denotes the set of subtasks $T_{k,l}$ in $H_{i,j}$ that do not have a predecessor subtask in $L_{i,j}$. Obviously the time demand generated by subtasks in H' is not affected by any subtasks in $L_{i,j}$. Consequently Step 2(d) in the algorithm in Figure 4 is modified such that after the first instance of some subtask in $L_{i,j}$ is released no further time demand is generated except that generated by the subtasks in H'. The pseudo-code of this further improved method is shown in Figure 7.

5 Summary

In this paper we proposed an algorithm to bound the response times of recurrent tasks in multiprocessor system where subtasks are statically released according to the phase modification technique. We introduced the concept of a per-task time demand function to facilitate the analysis of the worst-case behavior and the computation of the worst-case response times. In addition to the basic method of computing the PTTD functions, two improvements were also proposed.

In a multiprocessor real-time system, the interference among the executions of tasks are much more complicated than on a single-processor system. In general, it is impossible to identify the worst case without any over-restrictive assumptions. In this paper we bound the interference of a task in isolation, extend the time demand analysis to combine the bounds of the interference to determine the response time of a subtask, and eventually obtain a bound of the response time of a task. This approach controls the analysis complexity and yields reasonable bounds.

Acknowledgements

This work was partially supported by the NSF Contract No. NSF CCR-92-24269.

References

[1] R. Bettati and J. W.S. Liu, "End-to-End Scheduling to Meet Deadlines in Distributed Systems". In *The 12th International Conference on Distributed Computing Systems*, pages 452-459, Yokohama, Japan, June 1992.

[2] R. Rajkumar and L. Sha and J. P. Lehoczky, "Real-Time Synchronization Protocols for Multiprocessors", In *IEEE Real-Time Systems Symposium*, pages 259-269, 1988.

[3] J. Lehoczky and L. Sha and Y, Ding, "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior", In *IEEE Real-Time Symposium*, pages 166-171, 1989.

[4] M. Joseph and P. Pandya, "Finding Response Times in a Real-Time System", In *The Computer Journal of the British Computer Society*, 29(5):390-395, October 1986.

[5] J. Sun and R. Bettati and J. W.S. Liu, "An End-to-End Approach to Scheduling Periodic Tasks with Shared Resources in Multiprocessor Systems", In *IEEE Workshop on Real-Time Operating Systems and Software*, pages 18-22, 1994.