Scheduling non-preemptive hard real-time tasks with strict periods

Mohamed MAROUF
INRIA Rocquencourt
Domaine de Voluceau BP 105
78153 Le Chesnay Cedex - France
Email: mohamed.marouf@inria.fr

Yves SOREL
INRIA Rocquencourt
Domaine de Voluceau BP 105
78153 Le Chesnay Cedex - France
Email: yves.sorel@inria.fr

Abstract

Non-preemptive real-time scheduling and the corresponding schedulability analyses have received considerable less attention in the research community, compared to preemptive real-time scheduling. However, non-preemptive scheduling is widely used in industry, especially in the case of hard real-time systems where missing deadlines leads to catastrophic situations and where resources must not be wasted. In many industries such as avionics tasks may have strict periods, i.e. the start times of their executions must be separated by a fixed period. Indeed, this strict periodicity is generally required by sensors and actuators which may have accurate periods.

In this paper we consider separately the case where tasks have harmonic periods and the case where tasks have non-harmonic periods. Thus, the general case becomes a combination of both cases. In the harmonic case we give schedulability conditions to verify that a set of tasks is schedulable. In the non-harmonic case, in order to prove that a set of tasks is schedulable we propose local schedulability conditions *that we apply iteratively to each task of the set in order to verify that this current task, added to a sub-set of tasks already scheduled, leads to a schedulable set of tasks.*

Keywords: Hard real-time systems, Non-preemptive, Strict periods, scheduling heuristic.

1 Introduction

We consider hard real-time systems running on a uniprocessor platform where it is mandatory that all the tasks complete their executions before their deadlines. After the pioneering work of Liu and Layland [1], a lot of works has been done in the area of hard real-time scheduling to analyze and predict the schedulability of a preemptive task set under different scheduling policies and several task models. Note that all these works assume that the cost of scheduler and particularly the cost of the preemption is approximated in the WCETs (Worst Case Execution Time) of the tasks. Although preemptive scheduling is more efficient than non-preemptive scheduling, this latter is important for various reasons. Non-preemptive scheduling algorithms are easier to implement than preemptive algorithms, and can exhibit lower overhead at run-time. Preemption destroys program locality and affects the cache behavior, making the execution times more difficult to characterize and predict [2, 3]. The overhead of preemptive scheduling algorithms is more difficult to characterize and predict than that of non-preemptive scheduling algorithms. Since the scheduling overhead is often neglected in scheduling models, a non-preemptive scheduler will be closer to the model than a preemptive scheduler. In the former case, the cost of the scheduler itself could be taken into account in schedulability conditions. Contrary to non-preemptive scheduling, preemptive scheduling must guarantee the exclusive access to the shared resources and data. In automatic control applications, the input-output delay and jitter are minimized for all tasks when using a non-preemptive scheduling discipline, since there is no cost due to preemption which increases this delay [4]. This simplifies the techniques for delay compensation in the control design. In many practical real-time scheduling problems involving I/O scheduling, the properties of the hardware and software either make preemption impossible of prohibitively expensive [5]. For these reasons, designers often use non-preemptive approaches even if the numerous theoretical results of the preemptive approach do not extend easily to the non-preemptive approach [6].

In hard real-time systems some sensors and actuators must have accurate periods. In order to produce (resp. receive) data at the right period, the corresponding real-time tasks must have strict periods. Strict period means that if the task τ_i has the period T_i then $S_i^k = S_i^1 + (k \cdot T_i)$ [7], where S_i^1 and S_i^k are respectively the start time of the first and the $(k)^{th}$ repetitions of the task τ_i, these repetitions are called *jobs*. On the other hand, these sensor and actuator tasks always cooperate with other tasks which may themselves have strict or non strict periods.

In this paper, in order to simplify the problem, we assume that all the periods are strict rather than a combination of strict and non strict periods.

In order to schedule a set of non-preemptive tasks with strict periods, it is enough to study the behaviors of these tasks for a time interval equal to the LCM (Least Common

Multiple), called the hyper-period [8].

This paper is organized as follows: in section 2 we present the related work and the strict periodic tasks model. Section 3 is devoted to the schedulability analysis for harmonic tasks. In section 4 we present the schedulability analysis for non-harmonic tasks, which is the general case of strict periodic tasks. Finally, section 5 presents a conclusion and further work.

2 Related work and tasks model

2.1 Related work

Preemption related problems have received considerable attention in the real-time community. For example there exist a lot of uniprocessor schedulability conditions for popular algorithms like RM and EDF [1]. Unfortunately, these schedulability conditions become, at best, necessary conditions [9] in the non-preemptive case. However, non-preemption related problems must not be ignored since their resolutions may have great advantages in term of schedulability as pointed out previously. On the other hand, these problems are NP-Hard in the strong sense as Jeffay, Stanat and Martel [5] showed. Baruah and Chakraborty [10] analyzed the schedulability of the non-preemptive recurring task model and showed that there exists polynomial time approximation algorithms for both preemptive and non-preemptive scheduling. Buttazzo and Cervin [4] used the non-preemptive task model to reduce jitter. A comprehensive schedulability analysis of non-preemptive systems was performed by George, Rivierre, and Spuri [9]. The main difference between these works and the works proposed in this paper lies in the type of periods we consider, i.e. strict periods. We remind the reader that usually periods are such that the difference between the start times of two task instances may vary whereas it is a constant in our case.

There are some works in the case of non-preemptive tasks with strict periods. Al Sheikh and al. study the partition scheduling on an IMA (Integrated Modular Avionics) platform where the avionic functions are strictly periodic. They gave an exact algorithm with excessive computation time, based on a linear programming formulation, to solve the problem. Korst and al. proved in [11] a necessary and sufficient schedulability condition for two tasks, which becomes a sufficient condition for more than two tasks as proved by Kermia in [12]. However, as mentioned in [13], this later condition is very restrictive. In [14] Eisenbrand and al. proposed scheduling algorithms in the case of harmonic and non-harmonic tasks.

In this paper we first propose global schedulability conditions in the case of harmonic periods, whereas other works use local schedulability conditions, i.e. where a set of tasks is already scheduled and a new task is added to this set. In the general case of combination harmonic and non-harmonic periods, our proposed schedulability conditions are less restrictive than those proposed in the previous works which reject a lot of schedulable tasks.

2.2 Strict periodic tasks model

We consider real-time systems of non-preemptive tasks with strict periods. We assume that every task has a deadline equal to its period. A non-preemptive task τ_i denoted by $\tau_i(C_i, T_i, S_i^1)$ with the strict period T_i is characterized by:

- a first start time S_i^1.
- a strict period T_i such as the start time of the k^{th} instance of task τ_i is given by $S_i^k = S_i^1 + (k \cdot T_i)$,
- a deadline D_i equal to the period,
- a WCET (worst case execution time) $C_i \leq T_i$.

Afterwards, when the first start time is not given a task τ_i is denoted by $\tau_i(C_i, T_i)$.

Figure 1 shows an example of task with a strict period.

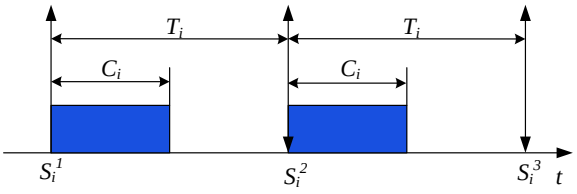

Figure 1: Model for non-preemptive task with a strict period

We assume that periods and WCETs are multiple of a unit of time, i.e. they are integers representing some cycles of the processor clock. If a task τ_i with execution time C_i is said to start at time unit t, it starts at the beginning of time unit t and completes at the end of time unit $t + C_i - 1$. Reciprocally, a time interval $[t_1, t_2]$ denotes a set of consecutive time units, given by $\{t_1, t_1 + 1, ..., t_2\}$.

3 Scheduling harmonic tasks

Let consider a set of harmonic tasks $\Gamma_n = \{\tau_i(C_i, T_i), i = 1, ..., n\}$. Without any loss of generality we consider that Γ_n is ordered according to the increasing values of the tasks periods, i.e. $T_i \leq T_j$ for $i < j$.

The schedulability analysis of harmonic tasks is a particular case of our study, in fact the GCD (Greatest Common Divisor) of all the tasks period is equal to the smallest task period T_1 and their hyper-period which is the LCM of of all the tasks period is equal to the greatest tasks period T_n. We can distinguish between the three following cases:

1. all tasks have distinct periods: $\forall i \neq j,\ T_i \neq T_j$,
2. some tasks have the same periods: $\exists i \neq j,\ T_i = T_j$,

(a) tasks with the same periods have the same WCETs,

(b) tasks with the same periods have different WCETs.

The study of harmonic tasks is based on a bin tree [14], where each bin has a size equal to the smallest period T_1. The initial bin contains a free slot of size equal to $T_1 - C_1$. At the i^{th} iteration of the scheduling process, each parent bin has $\frac{T_i}{T_{i-1}}$ children bins, and the final children bins number is equal to $\frac{T_i}{T_{i-1}} \cdot \frac{T_{i-1}}{T_{i-2}} \cdots \frac{T_2}{T_1} = \frac{T_i}{T_1}$. To schedule the candidate task $\tau_i(C_i, T_i)$, a local schedulability condition must be satisfied, i.e. there exist at least one children bin (among the $\frac{T_i}{T_1}$ children bins) containing a free slot of size bigger or equal to C_i (figure 2).

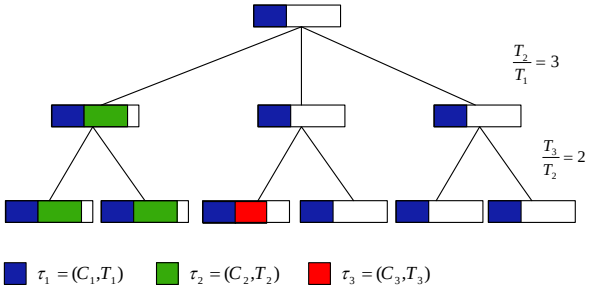

Figure 2: Bin tree for scheduling harmonic tasks

3.1 Case of all tasks with distinct periods

Theorem (1) gives a necessary and sufficient schedulability condition for the first case.

Theorem 1 *Let $\Gamma_n = \{\tau_i(C_i, T_i), i = 1, ..., n\}$ be a set of harmonic tasks such as $\forall i \neq j,\ T_i \neq T_j$. Γ_n is schedulable* ***if and only if***

$$\forall i = 2..n,\ C_1 + C_i \leq T_1 \tag{1}$$

Proof
We have

$$\begin{aligned} \forall i \neq j,\ T_i \neq T_j \quad &\Leftrightarrow \quad \forall i = 2..n,\ T_{i-1} < T_i \\ &\Leftrightarrow \quad \forall i = 2..n,\ \frac{T_i}{T_{i-1}} \geq 2 \end{aligned}$$

Thus, each parent bin of the bin tree (figure 2) has at least two children bins.

To prove the sufficiency of the condition (1), let assume that $\forall i = 2..n,\ C_1 + C_i \leq T_1$, we have to prove that Γ_n is schedulable. We first schedule the task $\tau_1(C_1, T_1)$ and consequently obtain a free slot of size $T_1 - C_1$. This parent bin has at least 2 children which contain the same free slot of size $T_1 - C_1$. As $T_2 \leq T_1 - C_1$, we can schedule τ_2 in one of these children bins and thus it remains at least one child bin with a free slot of size $T_1 - C_1$. This later bin will have at least 2 children which contain the same free slot of size $T_1 - C_1$. As $T_3 \leq T_1 - C_1$, we can schedule τ_3 in one of these children bins and thus it remains at least one child bin with a free slot of size $T_1 - C_1$. We proceed similarly until τ_n.

To prove the necessity of the condition (1), we have by hypothesis a set of harmonic tasks $\Gamma_n = \{\tau_i(C_i, T_i), i = 1, ..., n\}$ is schedulable, we have to prove that $\forall i = 2..n,\ C_1 + C_i \leq T_1$. We first schedule the task $\tau_1(C_1, T_1)$ and consequently obtain a free slot of size $T_1 - C_1$. This parent bin has at least 2 children which contain the same free slot of size $T_1 - C_1$. To schedule τ_2 we must have $C_2 \leq T_1 - C_1$. After scheduling τ_2 we have at least one remaining bin with a free slot of size $T_1 - C_1$, which is the largest free slots size among the children bins. This later bin will have at least 2 children bin which contain the same free slot of size $T_1 - C_1$, thus to schedule τ_3 we must have $C_3 \leq T_1 - C_1$. We proceed similarly until τ_n, and finally we have $\forall i = 2..n,\ C_i \leq T_1 - C_1$ which gives $\forall i = 2..n,\ C_1 + C_i \leq T_1$.

□

3.2 Case of some tasks with same periods

We first study the case where some tasks have the same periods and the same WCETs. Let consider a set of tasks $\Gamma_n = \{\tau_{i,j}(C_{i,j}, T_{i,j}),\ i = 1..n, j = 1..m_i\}$ where $C_{i,j} = C_i$ and $T_{i,j} = T_i$ for $j = 1..m_i$ and $\forall i = 2..n,\ T_i \neq T_{i-1}$. Γ_n contains m_i tasks with the same period T_i and the same WCET C_i. The following theorem gives a local sufficient schedulability condition for this set of tasks.

Theorem 2 *A set of harmonic tasks $\Gamma_n = \{\tau_{i,j}(C_{i,j}, T_{i,j}),\ i = 1..n, j = 1..m_i\}$ where $C_{i,j} = C_i$ and $T_{i,j} = T_i$ for $j = 1..m_i$ and $\forall i = 2..n,\ T_i \neq T_{i-1}$ is schedulable* **if**

$$a_n \geq 0 \text{ and } \forall i = 1..n-1,\ a_i > 0 \tag{2}$$

where

$$a_1 = \left\lceil \frac{T_1 - m_1 \cdot C_1}{T_1} \right\rceil$$

and

$$a_i = a_{i-1} \cdot \frac{T_i}{T_{i-1}} - \left\lceil \frac{m_i}{\left\lfloor \frac{T_1 - m_1 \cdot C_1}{C_i} \right\rfloor} \right\rceil, i = 2..n$$

Proof
Let consider a set of tasks $\Gamma_n = \{\tau_{i,j}(C_{i,j}, T_{i,j}),\ i = 1..n, j = 1..m_i\}$ where $C_{i,j} = C_i$ and $T_{i,j} = T_i$ for $j = 1..m_i$ and $\forall i = 2..n,\ T_i \neq T_{i-1}$.

The m_1 tasks $\tau_{1,j}$ are schedulable if $T_1 > m_1 \cdot C_1$. Thus, the first bin that we call the *initial bin*, contains one free slot of size equal to $T_1 - m_1 \cdot C_1$. Let a_i be the number of bins with a free slot of size equal to $T_1 - m_1 \cdot C_1$.

For $i = 1$, the number of initial bin is equal to 1, and thus $a_1 = 1$.

For $i = 2$, a children bin can contain $\left\lfloor \frac{T_1 - m_1 \cdot C_1}{C_2} \right\rfloor$ tasks $\tau_{2,j}$, so l_2 children bins can contain the m_2 tasks $\tau_{2,j}$ where

$$l_2 = \left\lceil \frac{m_2}{\left\lfloor \frac{T_1 - m_1 \cdot C_1}{C_i} \right\rfloor} \right\rceil .$$

Thus, the number of *available* children bins identical to the initial bin is equal to

$$a_2 = \frac{T_2}{T_1} - \left\lceil \frac{m_2}{\left\lfloor \frac{T_1 - m_1 \cdot C_1}{C_2} \right\rfloor} \right\rceil$$

If $a_2 > 0$ then we have at least one children bin identical to the initial bin.

For $i = 3$, we have a_2 bins identical to the initial bin which have $a_2 \cdot \frac{T_3}{T_2}$ children bin identical to the initial bin, so l_3 children bins can contain the m_3 tasks $\tau_{3,j}$ where

$$l_3 = \left\lceil \frac{m_3}{\left\lfloor \frac{T_1 - m_1 \cdot C_1}{C_3} \right\rfloor} \right\rceil .$$

Thus, the number of available bins identical to the initial bin is equal to

$$a_3 = a_2 \cdot \frac{T_3}{T_2} - \left\lceil \frac{m_3}{\left\lfloor \frac{T_1 - m_1 \cdot C_1}{C_2} \right\rfloor} \right\rceil$$

If $a_3 > 0$ than we have at least one children bin identical to the initial bin.

For $i = k$, we assume that $a_{k-1} > 0$. we have a_{k-1} bins identical to the initial bin which have $a_{k-1} \cdot \frac{T_k}{T_{k-1}}$ children bin identical to the initial bin, so l_k children bins can contain the m_k tasks $\tau_{k,j}$ where

$$l_k = \left\lceil \frac{m_k}{\left\lfloor \frac{T_1 - m_1 \cdot C_1}{C_k} \right\rfloor} \right\rceil .$$

Thus, the number of available bins identical to the initial bin is equal to

$$a_k = a_{k-1} \cdot \frac{T_k}{T_{k-1}} - \left\lceil \frac{m_k}{\left\lfloor \frac{T_1 - m_1 \cdot C_1}{C_k} \right\rfloor} \right\rceil$$

In order to schedule the m_{k+1} tasks $\tau_{k+1,j}$, we shall have at least one available bin identical to the initial bin, and thus $a_k > 0$

Finally, for $i = n$, a_n can be equal to 0 because $\tau_{n,j}$ are the last tasks so we do not need any available bins to schedule other tasks, and thus $a_n \geq 0$. ⊡

The following theorem gives a global schedulability condition based on the condition (2).

Theorem 3 *A set of harmonic tasks* $\Gamma_n = \{\tau_{i,j}(C_{i,j}, T_{i,j}),\ i = 1..n, j = 1..m_i\}$ *where* $C_{i,j} = C_i$ *and* $T_{i,j} = T_i$ *for* $j = 1..m_i$ *and* $\forall i = 2..n,\ T_i \neq T_{i-1}$ *is schedulable* **if**

$$\frac{T_n}{T_1} - \sum_{i=2}^{n} \frac{T_n}{T_{i-1}} \left\lceil \frac{m_i}{\left\lfloor \frac{T_1 - m_1 \cdot C_1}{C_i} \right\rfloor} \right\rceil > 0 \qquad (3)$$

with $T_1 - m_1 \cdot C_1 > 0$.

Proof

Let consider $\alpha_i = \frac{T_i}{T_{i-1}}$ and $\beta_i = \left\lceil \frac{m_k}{\left\lfloor \frac{T_1 - m_1 \cdot C_1}{C_i} \right\rfloor} \right\rceil$ for $i = 2..n$. Thus $a_i = \alpha_i \cdot a_{i-1} - \beta_i$.

As $\alpha_i > 0$ and $\beta_i > 0$ we have

$$\begin{aligned} a_n > 0 \quad &\Rightarrow \alpha_n \cdot a_{n-1} - \beta_n > 0 \\ &\Rightarrow \alpha_n \cdot a_{n-1} > \beta_n \\ &\Rightarrow \alpha_n \cdot a_{n-1} > 0 \\ &\Rightarrow a_{n-1} > 0 \end{aligned} \qquad (4)$$

so

$$(a_n > 0) \Rightarrow (a_{n-1} > 0) \Rightarrow ... \Rightarrow (a_2 > 0) \Rightarrow (a_1 > 0)$$

then

$$\forall i = 1..n,\ a_i > 0 \Leftrightarrow a_n > 0$$

$$\begin{aligned} a_n &= \alpha_n ... \alpha_2 \cdot a_1 - \alpha_n ... \alpha_3 \cdot \beta_2 - ... - \alpha_n \cdot \beta_{n-1} - \beta_n \\ &= a_1 \prod_{i=2}^{n} \alpha_i - \sum_{i=2}^{n} \beta_i \prod_{j=i}^{n} \alpha_i \end{aligned}$$

as $a_1 = 1$ and

$$\prod_{i=a}^{b} \alpha_i = \frac{T_a}{T_{a-1}} \cdot \frac{T_{a+1}}{T_a} ... \frac{T_{b-1}}{T_{b-2}} \cdot \frac{T_b}{T_{b-1}} = \frac{T_b}{T_{a-1}}$$

then

$$a_n = \frac{T_n}{T_1} - \sum_{i=2}^{n} \frac{T_n}{T_{i-1}} \beta_i$$

$$a_n = \frac{T_n}{T_1} - \sum_{i=2}^{n} \frac{T_n}{T_{i-1}} \left\lceil \frac{m_i}{\left\lfloor \frac{T_1 - m_1 \cdot C_1}{C_i} \right\rfloor} \right\rceil$$

Finally,

$$\begin{aligned} a_n > 0 \quad &\Leftrightarrow \forall i = 1..n,\ a_i > 0 \\ &\Leftrightarrow \textit{condition}\ (3) \end{aligned}$$

⊡

In the general case of harmonic tasks where some tasks may have the same periods but different WCETs, we have the following corollary which is an extension of the theorem 3.

Corollary 1 *A set of harmonic tasks* $\Gamma_n = \{\tau_{i,j}(C_{i,j}, T_{i,j}), i = 1..n, j = 1..m_i\}$ *where* $\forall j = 1..m_i,\ T_{i,j} = T_i$ *and* $\forall i \neq j,\ T_i \neq T_j$ *is schedulable* **if**

$$\frac{T_n}{T_1} - \sum_{i=2}^{n} \frac{T_n}{T_{i-1}} \left\lceil \frac{m_i}{\left\lfloor \frac{T_1 - m_1 \cdot C_1}{\overline{C_i}} \right\rfloor} \right\rceil > 0 \qquad (5)$$

where $T_1 - m_1 \cdot C_1 > 0$ *and* $\overline{C_i} = \max_{j=1..m_i} C_i$.

4 Scheduling non-harmonic tasks

In this section we study the schedulability of a set of tasks in the general case where periods are not necessarily harmonic. Korst and al. proved in [11] that two tasks $\tau_1(C_1, T_1)$ and $\tau_2(C_2, T_2)$ are schedulable if and only if:

$$C_1 + C_2 \leq GCD(T_1, T_2). \qquad (6)$$

It has been proved in [12, 13] that the necessary and sufficient schedulability condition (6) becomes a sufficient condition in the case of more than two tasks, thus a set of tasks $\Gamma = \{\tau_i(C_i, T_i),\ i = 1..n\}$ is schedulable if:

$$\sum_{i=1}^{n} C_i \leq GCD(\forall i, T_i). \qquad (7)$$

We distinguish in our study the three following cases:

1. $\frac{T_c}{g}$ even number
2. $\frac{T_c}{g}$ odd number
3. general case

where g is the GCD of all the periods tasks, and T_c is the period of the candidate task as explained in section 4.1.

We distinguish three cases: $\frac{T_c}{g}$ even and odd numbers and the general case which is a combination of these two cases. In this case, in order to prove that a set of tasks is schedulable we propose *local schedulability conditions* that we apply iteratively to each task of the set in order to verify that this current task, added to a sub-set of tasks already scheduled, leads to a schedulable set of tasks.

4.1 Scheduling strategy

We consider a set of n tasks $\mathbf{\Gamma_n} = \{\tau_i(C_i, T_i),\ i = 1..n\}$ and a subset $\mathbf{\Gamma'}$ of Γ_n containing the tasks that satisfies the schedulability condition (7). We call $\mathbf{g}$ the GCD of the periods T_i of all the tasks τ_i belonging to Γ'. $\mathbf{\Gamma''} = \Gamma_n \setminus \Gamma'$ is the complementary set of Γ' in Γ_n such as $\forall \tau_i \in \Gamma''$, the tasks of $\Gamma' \cup \{\tau_i\}$ do not satisfy the condition (7). In order to verify if a set of tasks Γ_n is schedulable, we first build the set of tasks Γ' then the scheduling process operates iteratively as follows: for each task τ_c of Γ'', called *candidate* task, we apply a local schedulability condition presented below, to the set of tasks $\Gamma'' \cup \{\tau_c\}$. If this condition is satisfied then the schedulable set of tasks is $\Gamma' \cup \{\tau_c\}$ else the schedulable tasks set is Γ'. Finally, we obtain the subset of schedulable tasks $\Gamma' \cup \Gamma''_s \subseteq \Gamma''$.

In this section, all the theorems are based on the computation of these free slots that we call *periodic free slots*.

Definition 1 *We call a* periodic free slot $\phi(C, T)$ *a free slot of size* $C \leq T$ *which is indefinitely repeated with a period* T.

The following example illustrates the free periodic slots after scheduling three tasks.

Example

Let consider the set of tasks $\Gamma_3 = \{\tau_1(1,3), \tau_2(1,6), \tau_3(1,9)\}$ to be scheduled. As $C_1 + C_2 + C_2 = 3 \leq GCD(T_1, T_2, T_3) = 3$, the condition (7) is satisfied and thus Γ_3 is schedulable as shown in the figure 3.

As we can see, there is one periodic free slot $\phi_1(1,6)$ and two periodic free slots $\phi_{2,3}(1,9)$.

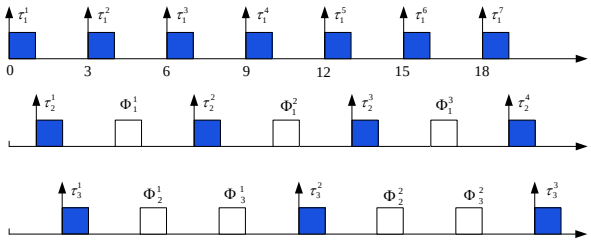

Figure 3: Scheduling diagram and periodic free slots

4.2 Case of $\frac{T_c}{g}$ odd number

Here we give a schedulability condition when $(\frac{T_c}{g})$ is an odd number, i.e. $\exists k \in \mathbb{N}^*,\ T_c = (2 \cdot k + 1) \cdot g$.

Theorem 4 *Let* $\Gamma_n = \{\tau_i(C_i, T_i), i = 1, ..., n\}$ *be a set of tasks scheduled according to the condition (7). A candidate task* τ_c *is schedulable if* $\exists \tau_i \in \Gamma_n$ *such as*

$$C_c \leq C_i \cdot \delta\left(T_c mod(T_i)\right) \qquad (8)$$

where mod is the $modulo$ function and δ is the Kronecker symbol:

$$\delta(i) = \begin{cases} 1 \ if \ \ i = 0 \\ 0 \ \ otherwise \end{cases}$$

The condition (8) is equivalent to $C_c \leq C_i$ and $T_c mod(T_i) = 0$.

Proof

Let $\Gamma_n = \{\tau_i(C_i, T_i), i = 1, ..., n\}$ be a set of tasks scheduled according to the condition (7): $\sum_{i=1}^{n} C_i \leq GCD(\forall i, T_i)$.

Each instance τ_i^k is executed in the time interval

$$I_i^k = [kT_i + S_i^1, kT_i + S_i^1 + C_i[$$

Let $n_i = \frac{T_i}{g}$, then

$$I_i^k = [(kn_i)g + S_i^1, (kn_i)g + S_i^1 + C_i[$$

Thus, in each time interval $[mg, (m+1)g[$ of length g, an instance τ_i^k can be executed in the time interval $[mg + S_i^1, mg + S_i^1 + C_i[\subset [mg, (m+1)g[$.

So if we schedule the first instances of the tasks τ_i into the time interval $[0, g[$ such as: $S_1^1 = 0$, $S_i^1 = \sum_{k=1}^{(i-1)} C_k$, then there will be no overlaps between the tasks $\tau_i \in \Gamma_n$.

This is a short proof of the sufficiency of the condition 7.

Now let consider a task τ_i such as τ_i^1 is executed into the time interval $[0, g[$: $0 \leq S_i^1 < g - C_i$. The second instance of this task will be executed after $(\frac{T_i}{g})$ time intervals of length g. Thus, it will leave $(\frac{T_i}{g} - 1)$ unused (free) time intervals $[mg + S_i^1, mg + S_i^1 + C_i[$.

Let $\phi(C_i, T_i)$ be a periodic free slot with start times $S_\phi = S_i^1 + mg$, $1 \leq m \leq (\frac{T_i}{g} - 1)$. These periodic free slots do not overlap the instances τ_i^k because they have the same period T_i.

To schedule the task τ_c into ϕ we must guarantee that $C_c \leq C_i$ and the period T_c is multiple of the period of ϕ T_i, thus $T_c \, mod(T_i) = 0$. ⊡

Theorem 5 *Let $\Gamma_n = \{\tau_i(C_i, T_i), i = 1, ..., n\}$ be a set of tasks scheduled according to the condition (7), and τ_c a candidate task which satisfies the condition (8). τ_c can be scheduled $\mathcal{N}_c$ times with the initial start times S_c^1 given by*

$$\begin{cases} \mathcal{N}_c = \frac{T_c}{T_i}(\frac{T_i}{g} - 1) \left\lfloor \frac{C_i}{C_c} \right\rfloor \\ S_c^1 = S_i^1 + k \cdot g + l \cdot T_i + m \cdot C_c \end{cases} \quad (9)$$

with

$$\begin{cases} 1 \leq k \leq \frac{T_i}{g} - 1 \\ 0 \leq l \leq \frac{T_c}{T_i} - 1 \\ 0 \leq m \leq \left\lfloor \frac{C_i}{C_c} \right\rfloor - 1 \end{cases}$$

Proof

As shown in the previous proof, the instances τ_c^k are scheduled into periodic free slots $\phi(C_i, T_i)$, thus one free slot can contain

$$\mathcal{N}_c' = \left\lfloor \frac{C_i}{C_c} \right\rfloor$$

instances τ_c^k, and the possible start times of these instances are given by

$$S_c'^1 = S_i^1 + m \cdot C_c$$

where $0 \leq m \leq \left\lfloor \frac{C_i}{C_c} \right\rfloor - 1$.

As there are $(\frac{T_i}{g} - 1)$ free slots in $[0, T_i[$, thus

$$\mathcal{N}_c'' = (\frac{T_i}{g} - 1)\mathcal{N}_c' = (\frac{T_i}{g} - 1) \left\lfloor \frac{C_i}{C_c} \right\rfloor$$

These free slots are separated by a time interval g, thus

$$S_c''^1 = S_c'^1 + k \cdot g = S_i^1 + k \cdot g + m \cdot C_c$$

with $1 \leq k \leq \frac{T_i}{g} - 1$.

The τ_c^k is now scheduled on a time interval T_i. If $\frac{T_c}{T_i} > 1$ then there are $\frac{T_c}{T_i}$ time intervals of length T_i where we can also schedule the instances τ_c^k. Thus

$$\mathcal{N}_c = \frac{T_c}{T_i}\mathcal{N}_c'' = \frac{T_c}{T_i}(\frac{T_i}{g} - 1) \left\lfloor \frac{C_i}{C_c} \right\rfloor$$

and the initial start times are given by

$$S_c^1 = S_c''^1 + l \cdot T_i = S_i^1 + k \cdot g + l \cdot T_i + m \cdot C_c$$

with $0 \leq l \leq \frac{T_c}{T_i} - 1$. ⊡

4.3 Case of $\frac{T_c}{g}$ even number

Here we give a schedulability condition and the corresponding initial start times, when $(\frac{T_c}{g})$ is even number, i.e. $\exists k \in \mathbb{N}$, $T_c = 2 \cdot k \cdot g$. This case is a little different from the previous one.

Theorem 6 *Let $\Gamma_n = \{\tau_i(C_i, T_i), i = 1, ..., n\}$ be a set of tasks scheduled according to the condition (7). A candidate task τ_c is schedulable if $\exists \tau_i \in \Gamma_n$ such as*

$$C_c \leq C_i \cdot \delta \, (T_c mod(2g) + T_i mod(2g)) \quad (10)$$

The condition (10) is equivalent to $C_c \leq C_i$ and $T_c mod(2g) = T_i mod(2g) = 0$.

Proof

Let $\Gamma_n = \{\tau_i(C_i, T_i), i = 1, ..., n\}$ be a set of tasks scheduled according to the condition (7). Let τ_i be a task of Γ_n such as $T_i \; mod(2g) = 0$, and τ_c the candidate task. Let $n_i = \frac{T_i}{2g}$.

The start times of the free slots of τ_i are given by:

$$S_k = S_i^1 + kg$$

where $1 \leq k \leq \frac{T_i}{g} - 1$, which can be written as

$$S_\phi^k \in \{S_i^1 + (2k+1) \cdot g, \; \forall k \geq 0\} \cup$$

$$\{S_i^1 + 2k \cdot g, \; \forall k \geq 1, \; k \, mod(n_i) \neq 0\}$$

The first interval $I_1 = \{S_i^1 + (2k+1) \cdot g, \; \forall k \geq 0\}$ describes a periodic free slot

$$\phi(C_i, 2g)$$

with a start time

$$S_\phi = S_i^1 + g$$

.

However, $I_2 = \{S_i^1 + 2k \cdot g, \forall k \geq 1, \, k \, mod(n_i) \neq 0\}$ can not contain one periodic free slot because its periodicity is broken for $k \; mod(n_i) = 0$, thus we have $(\frac{T_i}{2g} - 1)$ periodic free slots

$$\phi(C_i, T_i)$$

with the start times

$$S_\phi = S_i^1 + (2k)g,\ k = 1..(\frac{T_i}{2g} - 1)$$

Thus τ_c is schedulable if $C_c \leq C_i$ and T_c multiple of T_i or $2g$. As T_i is multiple of $2g$, then τ_c is schedulable if $C_c \leq C_i$ and T_c multiple of $2g$. ⊡

Theorem 7 *Let $\Gamma_n = \{\tau_i(C_i, T_i), i = 1, ..., n\}$ be a set of tasks scheduled according to the condition (7), and τ_c a candidate task which satisfies the condition (10). τ_c can be scheduled $\mathcal{N}_c$ times with the initial start times S_c^1 given by*

if $T_c\ mod\ (T_i) = 0$:

$$\begin{cases} \mathcal{N}_c = \frac{T_c}{T_i}(\frac{T_i}{g} - 1)\left\lfloor \frac{C_i}{C_c} \right\rfloor \\ S_c^1 = S_i^1 + k \cdot g + l \cdot T_i + m \cdot C_c \end{cases} \quad (11)$$

with

$$\begin{cases} 1 \leq k \leq \frac{T_i}{g} - 1 \\ 0 \leq l \leq \frac{T_c}{T_i} - 1 \\ 0 \leq m \leq \left\lfloor \frac{C_i}{C_c} \right\rfloor - 1 \end{cases}$$

if $T_c\ mod\ (T_i) \neq 0$ and $T_c\ mod\ (2g) = 0$:

$$\begin{cases} \mathcal{N}_c = \frac{T_c}{2g}\left\lfloor \frac{C_i}{C_c} \right\rfloor \\ S_c^1 = S_i^1 + (2k+1) \cdot g + m \cdot C_c \end{cases} \quad (12)$$

with

$$\begin{cases} 0 \leq k \leq \frac{T_c}{2g} - 1 \\ 0 \leq m \leq \left\lfloor \frac{C_i}{C_c} \right\rfloor - 1 \end{cases}$$

Proof

For $T_c\ mod\ (T_i) = 0$, the proof is similar to the one of theorem 5.

For $T_c\ mod\ (T_i) \neq 0$ and $T_c\ mod\ (2g) = 0$:
As we have seen in the proof of theorem 6, a scheduled task τ_i generates one periodic free slot $\phi(C_i, 2g)$ and $(\frac{T_i}{2g} - 1)$ periodic free slot $\phi(C_i, T_i)$.

As $T_c mod(T_i) \neq 0$, we can not use $\phi(C_i, T_i)$ to schedule the instances τ_c^k. However, we can use $\phi(C_i, 2g)$:

$\phi(C_i, 2g)$ generates

$$\mathcal{N}_c' = \frac{T_c}{2g}$$

periodic free slot $\phi(C_i, (\frac{T_c}{2g})2g) = \phi(C_i, T_c)$.

We have seen that the start time of $\phi(C_i, 2g)$ is $S_\phi = S_i^1 + g$, thus the start times of $\phi(C_i, T_c)$ are given by

$$S_\phi' = S_i^1 + (2k+1)g,\ 0 \leq k \leq \frac{T_c}{2g} - 1$$

Into each free slot of $\phi(C_i, T_c)$ we can schedule $\left\lfloor \frac{C_i}{C_c} \right\rfloor$ tasks τ_c, thus the number of schedulable tasks τ_c is equal to

$$\mathcal{N}_c = \mathcal{N}_c' \left\lfloor \frac{C_i}{C_c} \right\rfloor = \frac{T_c}{2g} \left\lfloor \frac{C_i}{C_c} \right\rfloor$$

and the initial start times of τ_c are given by

$$S_c^1 = S_c'^1 + m \cdot C_c$$

$$= S_i^1 + (2k+1) \cdot g + m \cdot C_c,\ 0 \leq m \leq \left\lfloor \frac{T_c}{2g} \right\rfloor - 1$$

4.4 General case

The following theorem is a combination of the theorem 4 and theorem 6.

Theorem 8 *Let $\Gamma_n = \{\tau_i(C_i, T_i), i = 1, ..., n\}$ be a set of tasks that satisfy the condition (7). Let τ_c be the task to be scheduled. τ_c is schedulable **if**:*

$$C_c \leq \sum_{i=1}^{n} C_i \cdot \delta\left[T_c mod(T_i) \cdot (T_c mod(2g) + T_i mod(2g))\right] \quad (13)$$

whith $T_i > g$.

Example

Let consider a set of six tasks $\Gamma = \{\tau_1(2, 9),\ \tau_2(1, 12),\ \tau_3(1, 18),\ \tau_4(1, 27)\}$. As $g = GCD(9, 12) = 3$ and $C_1 + C_2 = 3$, the condition (7) is satisfied: $C_1 + C_2 \leq GCD(T_1, T_2)$. However, it is not satisfied neither by $\{\tau_1, \tau_2, \tau_3\}$ nor by $\{\tau_1, \tau_3, \tau_4\}$: $C_1 + C_2 + C_3 = 4 \not\leq GCD(T_1, T_2, T_3) = 3$ and $C_1 + C_2 + C_4 = 4 \not\leq GCD(T_1, T_2, T_4) = 3$

So $\Gamma' = \{\tau_1(2, 9),\ \tau_2(1, 12)\}$ and $\Gamma'' = \{\tau_3(1, 18),\ \tau_4(1, 27)\}$.

Let $S_1^1 = 0$ and $S_2^1 = 2$.

For $\tau_c = \tau_3$, the condition (10) is satisfied by τ_2 and τ_c: $C_c = 1$ and

$$C_2 \cdot \delta\left(T_c mod(2g) + T_2 mod(2g)\right)$$

$$= 1 \cdot \delta\left(18 mod(6) + 12 mod(6)\right) = 1.$$

Thus the task τ_3 is schedulable.

As $T_3\ mod(T_2) = 6 \neq 0$, then according to the condition(14) we have

$$\begin{cases} \mathcal{N}_3 = \frac{T_3}{2g}\left\lfloor \frac{C_2}{C_3} \right\rfloor = 3 \\ S_3^1 = S_2^1 + (2k+1) \cdot g + m \cdot C_c = 2 + 3(2k+1) = 6k + 5 \end{cases} \quad (14)$$

with $0 \leq k \leq (\frac{T_3}{2g} - 1 = 2)$

τ_3 is scheduled with $S_3^1 = 2 + 3(1) = 5$, and we consider two periodic free slots $\phi(1, 18)$ with the start times $S_\phi \in \{11, 17\}$

For $\tau_c = \tau_4$, the condition (8) is satisfied by τ_1 and τ_c: $C_c = 1$ and $C_1 \cdot \delta\left(T_c mod(T_1)\right) = 2 \cdot \delta\left(27 mod(9)\right) = 2$. Thus, the task τ_4 is schedulable.

According to the condition(9), we have

$$\begin{cases} \mathcal{N}_4 = 12 \\ S_4^1 = 3k + 9l + m \end{cases} \quad (15)$$

with

$$\begin{cases} 1 \leq k \leq 2 \\ 0 \leq l \leq 2 \\ 0 \leq m \leq 1 \end{cases}$$

Thus the initial start times $S_4^1 \in \{3, 4, 6, 7, 12, 13, 15, 16, 21, 22, 24, 25\}$. Let $S_4^1 = 3$ and we consider 11 periodic free slots $\phi(1, 27)$ with the start times $S_\phi \in \{4, 6, 7, 12, 13, 15, 16, 21, 22, 24, 25\}$.

5 Conclusion and further work

In this paper we presented a schedulability analysis in the case of non-preemptive tasks with strict periods. We started by studying schedulability analysis for harmonic tasks and gave a schedulability condition for different cases. For non-harmonic tasks we gave a local schedulability condition which assumes that a set of task is already scheduled and a new task is to be scheduled. We also gave the scheduling conditions for each case and illustrate them by an example.

An interesting problem will be to study the scheduling analysis for strict periodic tasks onto multiprocessor platforms.

References

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

[2] H. Ramaprasad and F. Mueller. Tightening the bounds on feasible preemption points. In *RTSS '06: Proceedings of the 27th IEEE International Real-Time Systems Symposium*, pages 212–224, Washington, DC, USA, 2006. IEEE Computer Society.

[3] H. Ramaprasad and F. Mueller. Bounding worst-case response time for tasks with non-preemptive regions. In *RTAS '08: Proceedings of the 2008 IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 58–67, Washington, DC, USA, 2008. IEEE Computer Society.

[4] G. Buttazzo and A. Cervin. Comparative assessment and evaluation of jitter control methods. In *Proc. 15th International Conference on Real-Time and Network Systems*, Nancy, France, March 2007.

[5] K. Jeffay, D. F. Stanat, and C. U. Martel. On non-preemptive scheduling of period and sporadic tasks. In *Proceedings of the 12 th IEEE Symposium on Real-Time Systems*, pages 129–139, December 1991.

[6] F. Balarin, L. Lavagno, P. Murthy, and A. Sangiovanni-vincentelli. Scheduling for embedded real-time systems. *IEEE Design and Test of Computers*, 15(1):71–82, 1998.

[7] L. Cucu and Y. Sorel. Schedulability condition for systems with precedence and periodicity constraints without preemption. In *Proceedings of 11th Real-Time Systems Conference, RTS'03*, Paris, March 2003.

[8] J. Korst. *Periodic multiprocessor scheduling*. PhD thesis, Eindhoven university of technology, Eindhoven, the Netherlands, 1992.

[9] L. George, N. Rivierre, and M. Spuri. Preemptive and Non-Preemptive Real-Time UniProcessor Scheduling. Research Report RR-2966, INRIA, 1996. Projet REFLECS.

[10] S.K. Baruah and S. Chakraborty. Schedulability analysis of non-preemptive recurring real-time tasks. *Parallel and Distributed Processing Symposium, International*, 0:149, 2006.

[11] Jan H. M. Korst, Emile H. L. Aarts, Jan Karel Lenstra, and Jaap Wessels. Periodic multiprocessor scheduling. In *PARLE (1)*, pages 166–178, 1991.

[12] O. Kermia and Y. Sorel. Schedulability analysis for non-preemptive tasks under strict periodicity constraints. In *Proceedings of 14th International Conference on Real-Time Computing Systems and Applications, RTCSA'08*, Kaohsiung, Taiwan, August 2008.

[13] M. Marouf and Y. Sorel. Schedulability conditions for non-preemptive hard real-time tasks with strict period. In *Proceedings of 18th International Conference on Real-Time and Network Systems, RTNS'10*, Toulouse, France, November 2010.

[14] Friedrich Eisenbrand, Nicolai Hahnle, Martin Niemeier, Martin Skutella, José Verschae, and Andreas Wiese. Scheduling periodic tasks in a hard real-time environment. In *37th International Colloquium on Automata, Languages and Programming (ICALP2010)*, volume 37, pages 299–311. Springer-Verlag, 2010.