Safety Critical Multiprocessor Real-Time Scheduling with Exact Preemption Cost

Falou Ndoye
INRIA Paris-Rocquencourt
Domaine de Voluceau BP 105
78153 Le Chesnay Cedex - France
falou.ndoye@inria.fr

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

***Abstract*—In this paper, we address for safety critical applications the problem of multiprocessor real-time scheduling while taking into account the exact preemption cost. In the framework of multiprocessor real-time partitioned scheduling, we propose a greedy heuristic which balances the load of the tasks on all the processors and minimizes the response time of the applications. That heuristic uses a schedulability condition which is based on the $\oplus$ operation. That operation performs a schedulability analysis while taking into account the exact preemption cost. A performance analysis is achieved which compares the proposed heuristic to the branch and bound exact algorithm and to the worst-fit and best-fit heuristics.**

Keywords-multiprocessor real-time scheduling; partitioned scheduling; exact preemption cost; load balancing.

I. Introduction

For computation power and modularity issues, multiprocessor architectures are necessary to tackle complex applications found in domains such as avionics, automotives, mobile robotics, etc. Some of these applications are safety critical, leading to hard real-time task systems whose number of resources are fixed and constraints must be necessarily satisfied in order to avoid catastrophic consequences. Although fixed priority preemptive real-time scheduling allows a better success ratio than non-preemptive real-time scheduling, preemption has a cost. That cost is usually approximated in the WCET (Worst Case Execution Time) as assumed, explicitly, by Liu and Layland in their pioneering article [1]. However, such approximation is dangerous in a safety critical context since an application could miss some deadlines during its real-time execution even though schedulability conditions have been satisfied. This is why it is necessary to be aware of the exact preemption cost. In this paper, we address the problem of multiprocessor real-time scheduling while taking into account the exact preemption cost in safety critical applications. In the framework of multiprocessor real-time partitioned scheduling, we propose a greedy heuristic [2] using all the processors and which balances the load of the tasks on all the processors. That heuristic tends to minimize the response time (makespan) of the tasks. The schedulability condition is based on the algebraic $\oplus$ operation which performs a schedulability analysis taking into account the exact preemption cost.

The remainder of the paper is organized as follows: Section II presents related work about preemption cost and multiprocessor real-time scheduling. Section III describes the model and the schedulability analysis we propose. Section IV presents the proposed multiprocessor scheduling heuristic as well as its complexity, and Section V presents a performance analysis for that heuristic by comparing it with the Branch and Bound (B&B) exact algorithm, the Worst-Fit (WF) and Best-Fit (BF) heuristics. Finally, Section VI concludes and gives some directions for future work.

II. Related work

A. Exact preemption cost in real-time scheduling

There have been very few studies addressing the exact number of preemptions. Among them, the most relevant are the following. A. Burns, K. Tindell and A. Wellings in [3] presented an analysis that enables the global cost due to preemptions to be factored into the standard equations for calculating the worst case response time of any task, but they achieved that by considering the maximum number of preemptions rather than the exact number. Juan Echagüe, I. Ripoll and A. Crespo also tried to solve the problem of the exact number of preemptions in [4] by computing the schedule using idle times and counting the number of preemptions. However, they did not really determine the execution overhead incurred by the system due to these preemptions. Indeed, they did not take into account the cost of each preemption during the analysis. Hence, this amounts to considering only the minimum number of preemptions because some preemptions are not considered: those due to the increase in the execution time of the task because of the cost of preemptions themselves.

In order to reduce the preemption cost and improve the schedulability of tasks, a lot of work has focused on limited-preemption policies; among these we can cite fixed priority scheduling with deferred preemption (FPSDP) also called cooperative scheduling [5] and fixed priority scheduling with a preemption threshold (FPSPT) [6], [7]. According to FPSDP, each job of a task is a sequence of sub-jobs, where sub-jobs are not preemptive. When a job is being executed, it can only be preempted between two consecutive sub-jobs. For FPSPT, each task is assigned a nominal and a threshold

 ISBN: 978-1-61208-246-2

priority. A preemption will take place only if the preempting task has a nominal priority greater than the preemption threshold of the executing task. None of the previous works considers the exact number of preemptions. Nonetheless, that can affect the correct behavior of the system at run-time, or in any case leads to resources being wasted in terms of time and memory. It is typical and not difficult to determine the constant cost of every preemption which includes the context switch necessary to make the preemption possible together with the choice of the task with the highest priority. However, the exact number of preemptions is difficult to determine since it may vary according to every instance of a task. To our best knowledge there are only few studies that take into account the exact preemption cost in the schedulability conditions, except those presented in [8], [9]. The authors proposed a scheduling operation named $\oplus$ that performs a schedulability analysis while computing the exact number of preemptions. The principle of this operation is presented in Section III-B.

B. Multiprocessor real-time scheduling

The scheduling of real-time tasks on multiprocessor architectures can be achieved according to three main approaches: partitioned scheduling, global scheduling, and semi-partitioned scheduling.

In the partitioned scheduling approach [10], [11] the system of tasks is divided into a number of disjoint subsystems less than or equal to the number of processors in the multiprocessor architecture, and each of these subsystems is allocated to one processor. All the instances, or jobs, of a task are executed on the same processor and no migration is permitted. In this approach, it is necessary to choose a scheduling algorithm for every processor, possibly the same algorithm, and also an allocation algorithm. On the other hand, the allocation problem has been demonstrated to be NP-Hard [12]. This complexity is the main drawback of the partitioned scheduling approach.

Heuristics are considered to be the best suited solutions when the execution time is crucial as in the rapid prototyping phase of the design process. In the case of fixed priority scheduling and independent tasks, Davari and Dhall were the first to propose in [13] two preemptive scheduling algorithms RM-FF (Rate Monotonic First Fit) and RM-NF (Rate Monotonic Next Fit) to solve the multiprocessor real-time scheduling problem. In the proposed algorithms, the uniprocessor RM algorithm [1] is used to verify if a task is schedulable on a processor with respectively First-Fit (FF) and Next-Fit (NF) to solve the allocation problem. Another heuristic, RM-BF (Rate Monotonic Best Fit) was proposed in [14]. It makes it possible to minimize the remaining processor load $(1-U_{p_j})$, called the unutilized capacity of the processor p_j [15], where U_{p_j} is the load of the tasks on p_j. In contrast to RM-BF, RM-WF [14] (Rate Monotonic Worst Fit) maximizes the remaining processor load. All these approaches uses the classical Liu and Layland [1] model of tasks that assumes the preemption cost is approximated in the WCET. In order to tackle this problem [16] presents a first solution to take into account the exact preemption cost in multiprocessor real-time scheduling.

In the global scheduling approach [10], [11], a unique scheduling algorithm is applied globally for every processor of the multiprocessor architecture. All the ready tasks are in a unique queue shared by all the processors. In this queue, the m tasks with the highest priorities are selected to be executed on the m available processors. Besides preemptions, task migrations are permitted. The advantage of the global scheduling approach, is that it allows a better use of the processors. The main drawback of the global scheduling approach, is that each migration nowadays has a prohibitive cost.

In the semi-partitioned scheduling approach [17], [18], derived from the partitioned scheduling approach, each task is allocated to a specific processor as long as the total utilization of the processor does not exceed its schedulable bound. In this approach, some tasks can be portioned for their executions among multiple processors. During run-time scheduling, a portioned task is permitted to migrate among the allocated processors, while the partitioned tasks are executed on specific processors without any migration. The semi-partitioned scheduling approach allows a reduction of the number in migrations. But again, it is necessary to be aware that every migration has a cost.

C. Our choices

The cost of migrations in the global and semi-partitioned scheduling approaches leads us to choose the partitioned scheduling approach. In addition, since the partitioned scheduling approach amounts to transform the multiprocessor scheduling problem into several uniprocessor scheduling problems, we can take advantage of the numerous research results obtained for the uniprocessor scheduling problem. In order to achieve rapid prototyping, we propose an allocation heuristic rather than a metaheuristic [19] or an exact algorithm [20], and a schedulability condition to verify if a task is schedulable on a specific processor. Next-fit (NF) and first-fit (FF) heuristics can not optimize the load of the tasks on the processors, their choice is only based on the first processor which satisfies the schedulability condition. The BF heuristic using the load as a cost function, tries to fill a processor as much as possible before using another one. This technique does not induce load balancing. The only heuristic among the bin-packing heuristics which permits load balancing is WF. But, as with all the other bin-packing heuristics, WF tries to reduce the number of processors and that limits the balancing while multiprocessor architectures used in industrial applications, which we are interested in, have a fixed number of processors. That is why we propose a greedy heuristic similar to the WF heuristic, but which uses all the

 ISBN: 978-1-61208-246-2

available processors. This heuristic aims at minimizing the load on each processor. That induces to balance the load on all the processors. This proposed heuristic will be compared to the WF and BF heuristics and to the B&B exact algorithm.

Although preemptive scheduling algorithms are able to successfully schedule some task systems that cannot be scheduled by non-preemptive scheduling algorithms, the preemption has a cost. Indeed, Liu and Layland [1] assume that the preemption cost is approximated in the WCET. Thus, there are two possible cases: the approximation in time and memory space is high enough and thus will probably lead to wasting resources, or the approximation is low and thus a task system declared schedulable by, let us say RM, may miss some deadlines during its real-time execution. Consequently, we propose using the $\oplus$ operation [8], [9]. This is an algebraic operation that two tasks are schedulable, or not, while taking into account the exact preemption cost.

III. Model and Schedulability analysis

A. Model

Let $\Gamma_n = \{\tau_1, \tau_2, \cdots, \tau_n\}$ be a system of n preemptive, independent and periodic real-time tasks. Every task is denoted by $\tau_i = (r_i^1, C_i, D_i, T_i)$ where r_i^1, C_i, D_i and T_i are the characteristics of the task. r_i^1 is the first activation date, C_i is the EET (Exact Execution Time) without any approximation of the preemption cost, D_i is the relative deadline, and T_i the period of the task τ_i. We assume that $C_i \leq D_i \leq T_i$. Here we use the EET rather than the WCET because we assume that the code associated to a task is purely sequential, i.e. there are no conditional branches. Our hypothesis to consider the EET may seem unrealistic, but since we are considering safety critical applications it is mandatory to known the EET. Of course, when dealing with uncritical applications the WCET can be used. Also, we assume that Γ_n is scheduled according to the rate monotonic (RM) fixed-priority scheduling policy [1] on m identical processors (all the processors have the same computation power). Eventually, we assume that the processors have neither cache nor pipeline, or complex internal architecture. Both preview assumptions are usually made in safety critical applications where determinism is a key issue.

B. Schedulability analysis based on the $\oplus$ operation

Our schedulability analysis is based on the $\oplus$ scheduling operation [9] . This operation is applied to a pair of tasks (τ_i, τ_j), such that τ_i has the highest priority. It gives as a result a task $\mathcal{R}$, that is $\mathcal{R} = \tau_i \oplus \tau_j$. The $\oplus$ takes into account the exact preemption cost incurred by the task τ_j. The schedulability interval, i.e. the interval in which we study the schedulability of the tasks, comes from the theorem 1 below which was introduced by the J. Gossens [21].

Theorem 1: For a system $\Gamma_n = \{\tau_1, \tau_2, \cdots, \tau_n\}$ of n periodic tasks arranged by decreasing priorities with respect to a fixed-priority scheduling policy, let $(s_i^{'})_{i \in \mathbb{N}^*}$ be the sequence defined by:

$$\begin{cases} s_1^{'} = r_1^1 \\ \\ s_i^{'} = r_i^1 + \left\lceil \dfrac{(s_{i-1} - r_i^1)^+}{T_i} \right\rceil \cdot T_i, \quad 2 \leq i \leq n \end{cases} \tag{1}$$

If there exists a valid schedule of Γ_n until the time $s_n^{'} + H_n$ where $H_n = lcm\{T_i \mid i = 1, \cdots, n\}$, and $x^+ = max(x, 0)$, then this schedule is valid and periodic of period H_n from $s_n^{'}$.

Proof: The proof of this theorem is similar to that performed by *J. Goossens* in his Ph.D. thesis [21]. ■

A direct consequence of the previous theorem is that in the case of a valid schedule, the result of the schedule of the i first tasks is periodic of period $H_i = lcm\{T_j \mid j = 1, \cdots, i\}$ from $s_i^{'}$. Thus, the interval which precedes $s_i^{'}$ necessarily contains the *transient phase*, corresponding to the initial part of the schedule and the interval starting at time $s_i^{'}$ with length H_i is isomorphic to the *permanent phase* of the schedule of the i first tasks which repeats identically from the instant $s_i^{'}$.

In order to compute $\mathcal{R} = \tau_i \oplus \tau_j$ with $j = i + 1$, we set $\epsilon = min(r_i^1, r_j^1)$. Since ϵ always exists, the interval $[\epsilon, s_j^{'}]$ defines the transient phase and the interval $[s_j^{'}, s_j^{'} + H_j]$ defines the permanent phase, s_j and H_j are given by the theorem 1. The schedulability study of the tasks is performed in the interval $[\epsilon, s_j^{'} + H_j]$. In this interval, the number of instances of a task τ_j is given by $n_j = \frac{(s_j^{'} + H_j) - r_j^1}{T_j}$.

1) Principle of the $\oplus$ operation: The principle of $\oplus$ applied to a pair of tasks (τ_i, τ_j) consists in replacing the available time units of the highest priority task τ_i with the time units of the lowest priority task τ_j. In order to do that, both tasks are initially referenced to the same time origin ϵ. Then, task τ_i is rewritten according to the number of instances of task τ_j in the interval $[r_j^1, s_j^{'} + H_j]$ of both task periods. This operation allows not only the identification of the available time units in task τ_i, but also the verification that task τ_j does not miss any deadlines.

When the task τ_j is preempted by the task τ_i the exact number of preemptions must be computed for each instance of τ_j by considering all its time units. When τ_j is preempted, we increment its number of preemptions and we add the cost associated with one preemption in the remaining execution of τ_j, i.e. the number of time units that τ_j must execute in order to complete its execution. That scheme is repeated to take into account a preemption generated by a previous preemption, and so on. In contrast to other works presented in the literature, this principle makes it possible to have the exact number of preemptions. The cost associated to that

 ISBN: 978-1-61208-246-2

exact number of preemptions is added to the EET of τ_j to obtain its PET (Preemption Execution Time).

Figure 1 illustrates the PET. In this figure, the PET of task τ_i in the instance $k+1$ is given by $C_i^{k+1} = C_i + 2\alpha$ due to two preemptions, with α being the cost of one preemption. If the amount of PET unit of times fits in the available time

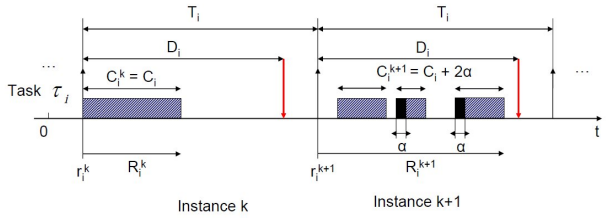

Figure 1. PET of a task

units of task τ_i, the task τ_j is schedulable, giving as a result task $\mathcal{R}$, otherwise it is not schedulable. $\oplus$ is an internal operation, i.e. the result given by $\oplus$ is also a task, that result may be in turn used as the highest priority task in another $\oplus$ operation. Thanks to this property it is possible to consider more than two tasks.

In order to perform the schedulability analysis of the task system $\Gamma_n = \{\tau_1, \tau_2, \cdots, \tau_n\}$, ordered according to the decreasing priorities of the tasks, the $\oplus$ operation is applied from the task with the highest priority to the task with the lowest priority. Consequently, if $\mathcal{R}_n$ is the scheduling task result of Γ_n, then $\mathcal{R}_n$ is obtained by successive iterations:

$$\begin{cases} \mathcal{R}_1 = \tau_1 \\ \mathcal{R}_{i+1} = \mathcal{R}_i \oplus \tau_{i+1}, \quad 1 \leq i < n \end{cases}$$

As such we have $\mathcal{R}_n = ((\tau_1 \oplus \tau_2) \oplus \cdots \oplus \tau_{n-1}) \oplus \tau_n$. The system Γ_n will be said schedulable if and only if all the tasks are schedulable. If this is not the case, then the system Γ_n is said to be not schedulable.

The complexity of $\oplus$ applied to a pair of tasks τ_i and τ_j is $O(l)$ with l is the LCM between the period of τ_i and the period of τ_j.

We denote by $H_j = lcm\{T_l : \tau_l \in hp(\tau_j)\}$ where T_l represents the period of task τ_l and $hp(\tau_j)$ denotes the subsystem of tasks with a priority higher than the priority of τ_j. The number of instances of task τ_i in the permanent phase is given by:

$$\sigma_{perm_j} = \frac{H_j}{T_j} = \frac{lcm\{T_l : \tau_l \in hp(\tau_j)\}}{T_j} \qquad (2)$$

The exact permanent load of a task τ_j, i.e. the load of the task τ_j, while taking into account the exact preemption cost, is given by:

$$U_j^* = \frac{C_j^*}{T_j} \text{ with } C_j^* = \frac{1}{\sigma_{perm_j}} \sum_{l=1}^{\sigma_{perm_j}} C_j^l \qquad (3)$$

In equation 3, C_j^l corresponds to the PET of the l^{th} instance in the permanent phase. As such, the exact permanent load of the system Γ_n composed of n periodic tasks scheduled on a processor p_i is given by:

$$U_{p_i}^* = \sum_{j=1}^{n} U_j^* \qquad (4)$$

2) Example: We apply the $\oplus$ operation to a system of periodic preemptive real-time tasks while taking into account the exact preemption cost. Let us consider such a system $\Gamma_3 = \{\tau_1, \tau_2, \tau_3\}$ of 3 tasks where τ_1 is the task with the highest priority, and τ_3 is the task with the lowest priority. We consider the cost of one preemption to be one time unit for all tasks. The characteristics of the tasks are summarized in table I.

Table I
TASKS' CHARACTERISTICS

Tasks	r_i^1	C_i	D_i	T_i
τ_1	0	3	7	15
τ_2	5	2	6	6
τ_3	3	4	10	10

The $\oplus$ operation is applied to a pair of operands. The left operand called the "executed task" corresponds to the result of the tasks previously scheduled, and the right operand called the "executable task" corresponds to the task to be scheduled. We represent an instance of the executable task by a unique sequence of symbols "**e**", in bold, followed by a sequence of symbols "a". Each symbol "**e**", in bold, in the executable task represents an executable time unit, i.e. the time unit that the task to be scheduled, must execute. Each symbol "a" represents an available time unit. Actually, such representation is repeated indefinitely since the task is periodic. We represent an instance of an executed task by a sequence of symbols "e" followed by a sequence of symbols "a", possibly repeated several times. Each symbol "e" in the executed task represents one executed time unit, i.e. the time unit executed by all the tasks previously scheduled. From the end of the transient phase, given by theorem 1, such representation is repeated according to the LCM of the tasks already scheduled.

The $\oplus$ operation aims at replacing all the available time units of the executed task (left operand) by the executable time units of the executable task (right operand). In order to make both tasks comparable, first the executable task is repeated according to the number of its instances in the schedulabity interval. Second, the executed task is rewritten according to the number of instances of the executable task in the schedulability interval. Therefore, the task resulting of the $\oplus$ operation applied to a pair of tasks, is an executed task represented by a sequence of symbols "e" followed by a sequence of symbols "a", possibly repeated several times.

 ISBN: 978-1-61208-246-2

According to these definitions, each task instance of the system Γ_3 is represented as:

$$\begin{cases} \tau_1 = \{\mathbf{e}, \mathbf{e}, \mathbf{e}, a, a, a, a, a, a, a, a, a, a, a, a\} \\ \tau_2 = \{\mathbf{e}, \mathbf{e}, a, a, a, a\} \\ \tau_3 = \{\mathbf{e}, \mathbf{e}, \mathbf{e}, \mathbf{e}, a, a, a, a, a, a\} \end{cases}$$

The scheduling task result $\mathcal{R}_3$ which describes the schedule of the task system is obtained by the following successive iterations:

$$\begin{cases} \mathcal{R}_1 = \Lambda \oplus \tau_1 \\ \mathcal{R}_i = \mathcal{R}_{i-1} \oplus \tau_i, \quad i = 2, 3 \end{cases}$$

Λ represents a task only composed with symbols "a" since there are no executed time units. $\mathcal{R}_1 = \Lambda \oplus \tau_1$ is computed as follows: according to equation 1 we have $s_1^{'} = 0$ and $H_1 = T_1 = 15$. Thus, the result of $\oplus$ applied to the pair (Λ, τ_1) is periodic of period $H_1 = T_1$ and is repeated indefinitely from $s_1^{'}$. We obtain $\mathcal{R}_1$ by replacing the 3 first available time units of Λ by the 3 executable time units of τ_1. Then, we have:

$\mathcal{R}_1 = \{e, e, e, a, a, a, a, a, a, a, a, a, a, a, a\}_{[0,15]}$

First iteration: Computation of $\mathcal{R}_2 = \mathcal{R}_1 \oplus \tau_2$. Thanks to equation 1, we have:

$$\begin{cases} s_1^{'} = 0 \\ \\ s_2^{'} = 5 + \left\lceil \dfrac{(0-5)^+}{6} \right\rceil \cdot 6 = 5 \end{cases}$$

We have $H_2 = lcm(15, 6) = 30$, thus the transient phase belongs to the interval $[0, 5]$ and the permanent phase belongs to the interval $[5, 35]$. In the schedulability interval $[0, 35]$, $\mathcal{R}_1$ is rewritten as follows:

$$\mathcal{R}_1 = \{e, e, e, a, a\}_{[0,5]}\{a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, e, e, e, a, a\}_{[5,35]}$$

Task τ_2 begins its execution at $t = 5$ corresponding to the beginning of the permanent phase. Its number of instances in the schedulability interval is $n_2 = \frac{(s_2^{'}+H_2)-r_2^1}{T_2} = \frac{(5+30)-5}{6} = \frac{30}{6} = 5$. According to the number of instances of τ_2 in the schedulability interval, $\mathcal{R}_1$ is rewritten as follows:

$$\begin{array}{l} \mathcal{R}_1 = \{e, e, e, a, a\}_{[0,5]}\{a, a, a, a, a, a\} \\ \{a, a, a, a, e, e, \}\{e, a, a, a, a, a\} \\ \{a, a, a, a, a, a\}\{a, e, e, e, a, a\} \end{array} \quad (5)$$

$\mathcal{R}_2 = \mathcal{R}_1 \oplus \tau_2$ is obtained by replacing in the equation 5 for each corresponding instance of τ_2 in $\mathcal{R}_1$, the available time units "a" of $\mathcal{R}_1$ with the executable time units "**e**", in bold, of τ_2. During this replacement a preemption of τ_2 by τ_1 corresponds to the transition ("a" $\rightarrow$ "*e*"). The preemption of τ_2 by τ_1 is denoted by the time unit "**p**" called preemption time unit. When τ_2 is preempted, the next available time unit of $\mathcal{R}_1$ after this preemption is replaced by a preemption time unit "**p**". After replacing all the available time units of τ_1 with the executable time units of τ_2 and after adding the preemption time unit "$\mathbf{p}$" in $\mathcal{R}_1$, we obtain:

$$\begin{array}{l} \mathcal{R}_2 = \{e, e, e, a, a\}_{(0,5)}\{\mathbf{e}, \mathbf{e}, a, a, a, a\} \\ \{\mathbf{e}, \mathbf{e}, a, a, e, e\}\{e, \mathbf{e}, \mathbf{e}, a, a, a\} \\ \{\mathbf{e}, \mathbf{e}, a, a, a, a\}\{\mathbf{e}, e, e, e, \mathbf{p}, \mathbf{e}\} \end{array}$$

For each corresponding instance of τ_2 in $\mathcal{R}_2$, its PET is given by the sum of the number of its executable time units **e**, in bold, and the number of its preemptions time unit "**p**". In the 4 first corresponding instances of τ_2 in $\mathcal{R}_2$, the PETs are the same and equal to 2 (PET=EET) because τ_2 is not preempted in these instances, but in its 5^{th} instance, it is preempted once. That is the reason why its PET is equal to 3. In any corresponding instance of τ_2 in $\mathcal{R}_2$, the PET fits in the available time units left by $\mathcal{R}_1$ in this instance. Thus, the task τ_2 is schedulable while taking into account the exact preemption cost. Actually, we have:

$$\begin{array}{l} \mathcal{R}_2 = \{e, e, e, a, a\}_{[0,5]}\{e, e, a, a, a, a, e, e, a, a, e, e, \\ e, e, e, a, a, a, e, e, a, a, a, a, e, e, e, e, p, e\}_{[5,35]} \end{array}$$

The differences with the previous expression of $\mathcal{R}_2$ is that the executable time units "**e**", in bold, become executed time units "e", and $\mathcal{R}_2$ does not exhibit the corresponding instances of τ_2.

Second iteration: Computation of $\mathcal{R}_3 = \mathcal{R}_2 \oplus \tau_3$. Thanks to equation 1, we have:

$$\begin{cases} s_2^{'} = 5 \\ \\ s_3^{'} = 3 + \left\lceil \dfrac{(5-3)^+}{10} \right\rceil \cdot 10 = 13 \end{cases}$$

We have $H_3 = lcm(lcm(15, 6), 10) = lcm(30, 10) = 30$, thus the transient phase belongs to the interval $[0, 13]$ and the permanent phase belongs to the interval $[13, 43]$. In the schedulability interval $[0, 43]$, $\mathcal{R}_2$ is rewritten as follows:

$$\begin{array}{l} \mathcal{R}_2 = \{e, e, e, a, a, e, e, a, a, a, a, e, e\}_{[0,13]}\{a, a, e, \\ e, e, e, e, a, a, a, e, e, a, a, a, a, e, e, e, e, p, e, e, e, a, \\ a, a, a, e, e\}_{[13,43]} \end{array}$$

Task τ_3 begins its execution during the transient phase at $t = 3$. Its number of instances in the schedulability interval is $n_3 = \frac{(s_3^{'}+H_3)-r_3^1}{T_3} = \frac{(13+30)-3}{10} = \frac{40}{10} = 4$. According to the number of instances of τ_3 in the schedulability interval, $\mathcal{R}_2$ is rewritten as follows:

$$\begin{array}{l} \mathcal{R}_2 = \{e, e, e\}\{a, a, e, e, a, a, a, a, e, e\}_{[3,13]} \\ \{a, a, e, e, e, e, e, a, a, a\}\ \{e, e, a, a, a, a, e, e, \\ e, e\}\{p, e, e, e, a, a, a, a, e, e\} \end{array} \quad (6)$$

$\mathcal{R}_3 = \mathcal{R}_2 \oplus \tau_3$ is obtained by replacing in the equation 6 for each corresponding instance of τ_3 in $\mathcal{R}_2$, the available

time units "a" of $\mathcal{R}_2$ with the executable time units "**e**", in bold, of τ_3. During this replacement, a preemption of the task τ_3 by τ_1 or by τ_2 corresponds to a transition ("a" $\rightarrow$ "e"). When τ_3 is preempted, the next available time unit of $\mathcal{R}_2$ is replaced by a preemption time unit "**p**". After replacing the available time units of $\mathcal{R}_2$ with the executable time units of τ_3 and after adding the preemption time units "**p**" in $\mathcal{R}_2$, we obtain:

$$\mathcal{R}_3 = \{e,e,e\}\{\mathbf{e},\mathbf{e},e,e,\mathbf{p},\mathbf{e},\mathbf{e},a,e,e\}_{[3,13]}$$
$$\{\mathbf{e},\mathbf{e},e,e,e,e,e,\mathbf{p},\mathbf{e},\mathbf{e}\}\{e,e,\mathbf{e},\mathbf{e},\mathbf{e},\mathbf{e},e,e,e,e\}$$
$$\{p,e,e,e,\mathbf{e},\mathbf{e},\mathbf{e},\mathbf{e},e,e\}$$

For each corresponding instance of τ_3 in $\mathcal{R}_3$, its PET is given by the sum of the number of its executable time units "**e**", in bold, and the number of its preemptions time units "**p**". In the 2 first corresponding instances of τ_3 in $\mathcal{R}_3$, the task τ_3 suffers one preemption. Its PETs in every instance are the same and equal to 5. In its other instances there is no preemption of τ_3 and the PETs of τ_3 in these instances are the same and equal to 4 (PET=EET). In any corresponding instance of τ_3 in $\mathcal{R}_3$, the PET fits in the available time units of $\mathcal{R}_2$ in this instance. Thus, the task τ_3 is schedulable while taking into account the exact preemption cost. Finally, we have:

$$\mathcal{R}_3 = \{e,e,e,e,e,e,e,p,e,e,a,e,e\}_{[0,13]}\{,e,e,e,$$
$$e,e,e,e,p,e,e,e,e,e,e,e,e,e,e,e,p,e,e,e,e,e,$$
$$e,e,e,e\}_{[13,43]}$$

The differences with the previous expression of $\mathcal{R}_3$ is that the executable time units "**e**", in bold, become executed time units "e", and $\mathcal{R}_3$ does not exhibit the corresponding instances of τ_3.

Since all the tasks are schedulable then the system $\Gamma_3 = \{\tau_1, \tau_2, \tau_3\}$ is schedulable.

Figure 2 presents the result of the schedule of Γ_3. In this figure, the permanent phase corresponds to the highlighted zone of the schedule and the transient phase corresponds to the interval preceding that zone. The disk represents only the permanent phase in a more compact form. This double representation of the schedule is obtained from the SAS software [22].

IV. Multiprocessor Scheduling Heuristic

The heuristic presented in *Algorithm*1 is a greedy heuristic. The solution is built step by step. In each step a decision is taken and this decision is never questioned during the following steps (no backtracking). The effectiveness of such a greedy heuristic is based on the decision taken to build a new element of the solution. In our case, the decision is taken according to a cost function which aims at minimizing the load.

A. Cost function

The cost function allows the selection of the best processor p_j to schedule a task τ_i. In our case, this cost function

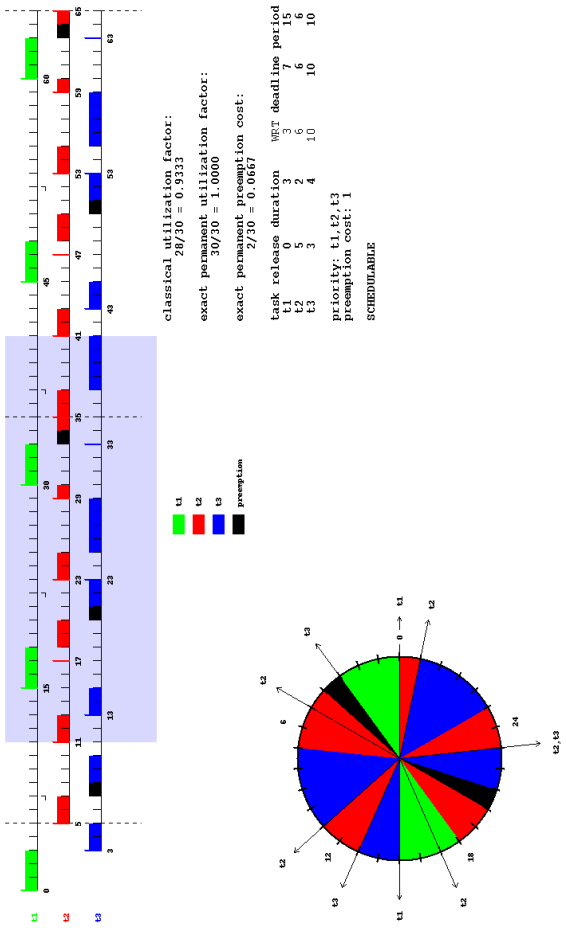

Figure 2. Result of the scheduling of Γ_3, taking into account the exact preemption cost

is the load $U^*_{p_j}$ (equation 4) of the task τ_i and all the tasks already allocated on the processor p_j. The processor which minimizes this cost function for τ_i among all the processors, is considered to be the best processor to schedule the task τ_i.

In the case of the previous example, according to equation 4, the exact permanent load of the system Γ_3 scheduled on a processor p is given by:

$$U^*_p = \frac{3}{15} + \frac{1}{6} \cdot \frac{(2+2+2+2+3)}{5} + \frac{1}{10} \cdot \frac{(5+4+4)}{3} = 1$$

B. Principle of our allocation heuristic

We use a "list heuristic" [23]. In our case, we initialize this list, called the "candidate task system", with the task system given as input. We use for that candidate task system the decreasing order of the task priorities (according to RM fixed-priority scheduling policy [1]). At each step of the heuristic, the task with the highest priority is selected among the candidate task system, and we attempt to allocate it to its best processor according to the cost function presented

previously. The heuristic minimizes the load $U^*_{p_j}$ of the task system on the different processors. It is similar to the WF bin-packing heuristic but all the available processors are used rather than the first processors necessary to schedule the task system.

If Γ_n is the task system with n tasks and m is the number of processors, the complexity in the worst case of our heuristic is equal to $O(n.m.l)$, with $l = lcm\{T_i : \tau_i \in \Gamma_n\}$.

Algorithm 1 Greedy heuristic

1: Initialize the candidate task system W with the task system given as input and in the decreasing order of their priorities, initialize the boolean variable $TasksSchedulable$ to $true$
2: **while** W is not empty and $TasksSchedulable = true$ **do**
3: Select in W the highest priority task τ_i
4: *% Verify on each processor p_j if task τ_i is schedulable.%*
5: **for** j=1 to m **do**
6: **if** task τ_i is schedulable on p_j with the exact preemption cost (scheduling operation $\oplus$ [9]) **then**
7: Compute the cost function of task τ_i on the processor p_j, i.e. the load of p_j using the equation 4 given in subsection III-B
8: **end if**
9: **end for**
10: *% Using the cost function again, choose the best processor for τ_i among all the processors on which τ_i is schedulable.%*
11: **if** τ_i is schedulable on one or several processors **then**
12: Schedule the task τ_i on the processor which minimizes the cost function
13: Remove the task τ_i from W.
14: $TasksSchedulable = true$
15: **else**
16: $TasksSchedulable = false$
17: **end if**
18: **end while**

V. Performance Analysis

Our heuristic is compared with the B&B exact algorithm and the WF and BF heuristics. The B&B enumerates all the possible solutions in order to find the best solution which minimizes the load of the tasks on the processors. In the B&B, WF and BF heuristics, we use the $\oplus$ operation presented in Section III-B as the schedulability condition. We compare the algorithms according to their execution time, their success ratio, the response time of the task systems, i.e. the total execution time of the tasks, and the unutilized capacity of the processors used during the allocation.

A. Execution time of the heuristics

We perform two kinds of tests to compare the execution time of the four algorithms. First, we fix the number of processors to 10 and we vary the number of tasks between 100 to 1000 tasks. Every task system is scheduled with the four algorithms and the corresponding execution times are computed. We obtained the results shown in Figure 3. In the second test, we use a single task system composed of 1000 tasks randomly generated and we vary the number of processors. We obtained the results shown in Figure 4.

In both tests, we notice that the exact algorithm explodes very quickly whereas the heuristics keep a reasonable execution time. Our heuristic up to 1000 tasks is close to the WF and BF heuristics in terms of execution time. However, for higher numbers of tasks less good results are obtained with our heuristic. In Figure 4 we also notice that when the number of processors varies, the execution times of WF and BF are constant, because these heuristics use the minimum number of processors. Another remark about Figure 4 is that the execution time of our heuristic does not increase monotonically with the number of processors, in contrast to Figure 3. Indeed, in our heuristic, increasing the number of processors leads to distributing the tasks on all the processors. That increase in terms of processors, can decrease locally the LCM of the tasks on some processors, and consequently can reduce the execution time of the $\oplus$ operation.

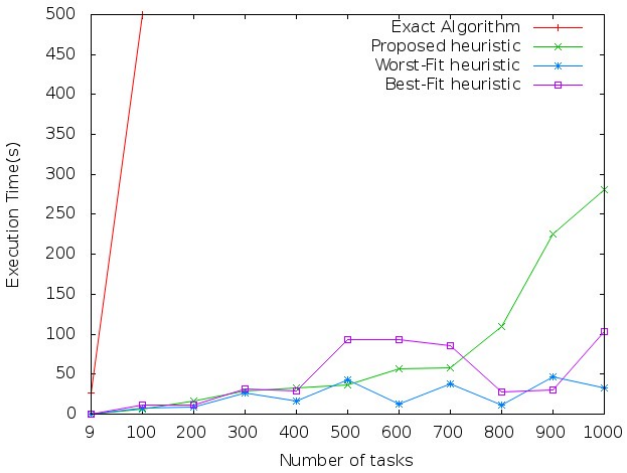

Figure 3. Execution time of the algorithms according the number of tasks

B. Success ratio

In these tests, we compare the success ratio of our heuristic with the B&B exact algorithm and the WF and BF heuristics. The success ratio of an algorithm is defined as follows:

 ISBN: 978-1-61208-246-2

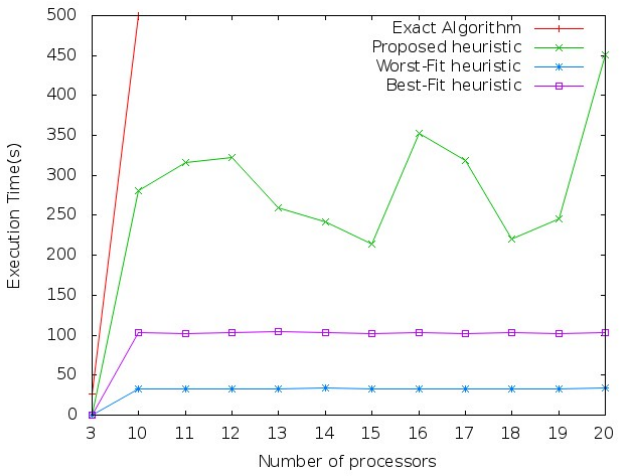

Figure 4. Execution time of the algorithms according to the number of processors

$$\frac{number\ of\ task\ systems\ schedulable}{total\ number\ of\ task\ systems}$$

Due to the complexity of the B&B and in order to compare it with the heuristics, we executed each algorithm on 6 task systems. Each task system is composed at most of 10 randomly generated tasks and is executed on 2 processors. At each execution we determine for each algorithm the number of schedulable task systems.

As shown in Figure 5, we notice that WF and BF give better results than our heuristic in terms of success ratio. This loss in terms of success ratio is largely compensated by the gain in terms of response time of the task systems and by the unutilized capacity of the processors, as described in the subsections V-C and V-D.

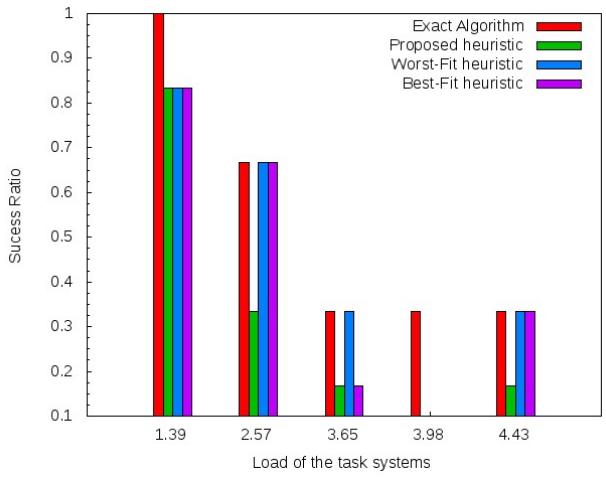

Figure 5. Sucess ratio

C. Response time of the task systems

In these tests, we consider 10 task systems. The number of tasks in the task systems varies between 100 and 1000 randomly generated tasks and each task system is executed on 10 processors. We limit the tests to the WF and BF heuristics and our proposed heuristic because of the complexity of the B&B exact algorithm and we know that the B&B already gives better results than the heuristics. For each task system, we determine the allocation found by each heuristic and for this allocation the response time of the task system, i.e., the total execution time of all the tasks, is computed. We compare the response time of the task systems between the heuristics, as shown in Figure 6.

In this figure, we notice that the allocation found by our heuristic gives a better response time than those found by WF and BF. This is due to the fact that the execution of the tasks is parallelized on all the available processors whereas WF and BF attempts to reduce the number of processors rather than parallelize the execution of the tasks.

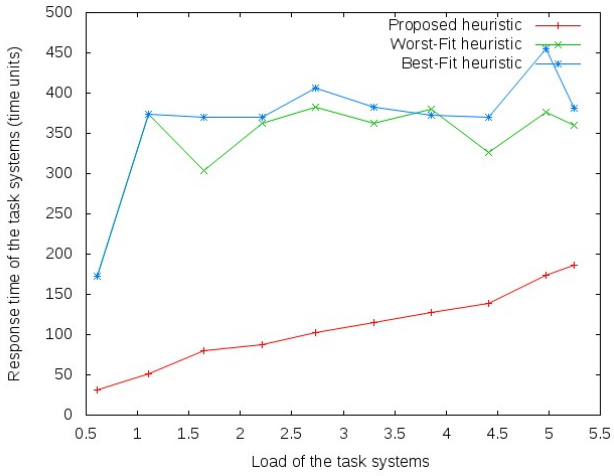

Figure 6. Execution time of the task systems

D. Average of the unutilized capacity of the processors

In these tests, we consider 10 task systems. The number of tasks in the task systems varies between 100 and 1000 randomly generated tasks and each task system is executed on 10 processors. We limit the tests to the WF and BF heuristics and our proposed heuristic because of the complexity of the B&B exact algorithm. In addition, we know that the B&B already gives better results than the heuristics. For each task system we determine the allocation found by each heuristic and for this allocation we compute the average of the remaining processor load $(1 - U_{p_j})$, called untilized capacity, on the processors p_j used in this allocation. We compare the unutilized capacity of the processors used with the heuristics as shown in Figure 7.

In this figure, we observe that the allocation found by our heursitic gives for each processor more flexibility, i.e., more unutilized capacity, than those found by WF and BF. This is due to the fact that our heuristic balances the load on all the available processors, which ensures an execution time slack, whereas the BF heuristic fills the processors as much as possible, which that increases the risk of non schedulabily of a task system at run-time. On the other hand, the WF heuristic, balances the load only on the processors already used and does not consider all the available processors.

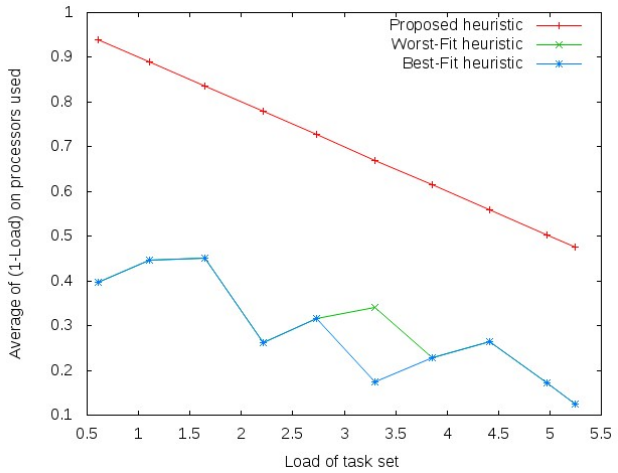

Figure 7. Average of (1-load) on the processors used

VI. Conclusion and future work

In this paper, we have presented a greedy heuristic which allocates and schedules, on a multiprocessor architecture, a system of real-time tasks while balancing the load on the processors. In addition, this heuristic takes into account the exact preemption cost that must be carefully considered in safety critical applications, which is the focus of our work.

We have carried out a performance analysis showing that, up to 1000 tasks, the proposed greedy heuristic, has results close to those of the WF and BF heuristics in terms of execution time. For higher number of tasks less good results can be obtained with our heuristic. On the other, hand the proposed heuristic is better than the WF and BF heuristic in terms of load balancing and flexibility, i.e., more not-utilized capacity, of tasks at run-time. Also the allocation found with our heuristic has better response time than those with WF and BF heuristics.

In future works, we plan to study the multiprocessor real-time scheduling of dependent tasks which leads to deal with data transfers and shared data management.

References

[1] C. L. Liu and J W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environnment. *Journal of the ACM*, vol. 20(1), January 1973.

[2] S. Martello E.G. Coffman, G. Galambos and Daniele Vigo. Bin packing approximation algorithms: Combinatorial analysis. *Handbook of combinatorial optimization*, 1998.

[3] A. Burns, K. Tindell, and A. Wellings. Effective analysis for engineering real-time fixed priority schedulers. *IEEE Trans. Softw. Eng.*, 21:475–480, May 1995.

[4] J. Echague, I. Ripoll, and A. Crespo. Hard real-time preemptively scheduling with high context switch cost. In *Proceedings of 7th Euromicro workshop on Real-Time Systems*, Los Alamitos, CA, USA, 1995. IEEE Computer Society.

[5] Alan Burns. Preemptive priority-based scheduling: An appropriate engineering approach. In *Advances in Real-Time Systems, chapter 10*, pages 225–248. Prentice Hall, 1994.

[6] Y. Wang and M. Saksena. Scheduling fixed-priority tasks with preemption threshold. In *Proceedings of the 6 International Conference on Real-Time Computing Systems and Applications*, RTCSA'99, Washington, DC, USA, 1999.

[7] M. Saksena and Y. Wang. Scalable real-time system design using preemption thresholds, November 2000.

[8] P. Meumeu Yomsi and Y. Sorel. Extending rate monotonic analysis with exact cost of preemptions for hard real-time systems. In *Proceedings of 19th Euromicro Conference on Real-Time Systems, ECRTS'07*, Pisa, Italy, July 2007.

[9] P. Meumeu Yomsi and Y. Sorel. An algebraic approach for fixed-priority scheduling of hard real-time systems with exact preemption cost. Research Report RR-7702, INRIA, August 2011.

[10] Robert I. Davis and Alan Burns. A survey of hard real-time scheduling algorithms and schedulability analysis techniques for multiprocessor systems. Technical report, University of York, Department of Computer Science, 2009.

[11] O.U.P. Zapata and P.M. Alvarez. Edf and rm multiprocessor scheduling algorithms: Survey and performance evaluation. *http://delta.cs.cinvestav.mx/ pmejiamultitechreport.pdf*, Oct 2005.

[12] Garey and Johnson. *Computers and intractability: a guide to the theory of NP-completeness*. W.H. Freeman and Company, New York, NY, USA, 1979.

[13] S.K. Dhall and C.L. Liu. On a real-time scheduling problem. *Operation Research*, vol. 26(1), 1978.

[14] Y. Oh and S.H. Son. Tight performance bounds of heuristics for a real-time scheduling problem. Technical Report CS-93-24, Univ. of Virginia. Dep. of Computer Science, Charlottesville, VA 22903, May 1993.

 ISBN: 978-1-61208-246-2

[15] L. George I. Lupu, P. Courbin and J. Goossens. Multi-criteria evaluation of partitioning schemes for real-time systems. In *The 15th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA'10*, Bilbao, Spain, September 2010.

[16] F. Ndoye and Y. Sorel. Preemptive multiprocessor real-time scheduling with exact preemption cost. In *Proceedings of 5th Junior Researcher Workshop on Real-Time Computing, JRWRTC'11, in conjunction with the 18th International conference on Real-Time and Network Systems, RTNS'11*, Nantes, France, September 2011.

[17] S. Katoa and N. Yammasaki. Semi-partitioning technique for multiprocessor real-time scheduling. In *Proceedings of WIP Session of the 29th Real-Time Systems Symposium (RTSS), IEEE Computer Society*, 2008.

[18] J. H. Anderson, V. Bud, and C. U. Devi. An edf-based scheduling algorithm for multiprocessor soft real-time systems. In *Proceedings of the 17th Euromicro Conference on Real-Time Systems*, pages 199–208, Washington, DC, USA, 2005. IEEE Computer Society.

[19] E. G Talabi. *Metaheuristics*. Wiley, 2009.

[20] J. E. Mitchell. Branch-and-cut algorithms for combinatorial optimization problems. pages pp 65–67, 2002.

[21] J. Goossens. *Scheduling of Hard Real-Time Periodic Systems with Various Kinds of Deadline and Offset Constraints*. PhD thesis, Universit Libre de Bruxelles, 1999.

[22] P. Meumeu Yomsi, L. George, Y. Sorel, and D. de Rauglaudre. Improving the quality of control of periodic tasks scheduled by fp with an asynchronous approach. *International Journal on Advances in Systems and Measurements*, 2(2), 2009.

[23] K. M. Chandy L.T. Adams and J. R. Dickson. A comparison of list schedules for parallel processing systems. *Commun. ACM*, 17:685–690, December 1974.

 ISBN: 978-1-61208-246-2