Preemptive Multiprocessor Real-Time Scheduling with Exact Preemption Cost

Falou Ndoye and Yves Sorel
INRIA Paris-Rocquencourt,
Domaine de Voluceau BP 105, 78153 Le Chesnay Cedex - France
falou.ndoye@inria.fr, yves.sorel@inria.fr

***Abstract*—We propose a greedy heuristic to solve the real-time scheduling problem of periodic preemptive tasks on a multiprocessor architecture while taking into account the exact preemption cost. In the framework of partitioned scheduling, this is achieved by combining an allocation heuristic whose cost function minimizes the makespan, and a schedulability condition based on the $\oplus$ operation which takes into account the exact preemption cost.**

I. Introduction

For computation power and modularity issues, multiprocessor architectures are necessary to tackle complex applications found in domains such that avionics, automotives, mobile robotics, etc. Some of these applications are safety critical, leading to hard real-time task systems whose constraints must be necessarily satisfied in order to avoid catastrophic consequences. Although 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, explicitely, by Liu and Layland in their pioneer article [1]. However, such approximation is dangerous in safety critical context since an application may miss some deadlines during its real-time execution even though schedulability conditions were satisfied. In oder to tackle the problem A. Burns and al. in [2] 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. Other works aim at bounding the number of preemptions [3], [4]. In both cases the exact number of preemptions is not considered leading to waste resources in time and memory. However, that exact number of preemptions is difficult to determine since it may vary according to every instance of a task whereas it is not difficult to determine the constant cost of every preemption which includes the context switch necessary to make possible the preemption and the choice of the task with the highest priority. It is the reasons why it is necessary to take care of the the exact preemption cost. In this paper we address all together the previous issues.

The remainder of the paper is organized as follows: section II presents the related work on multiprocessor real-time scheduling and preemption cost, in section III we describe the model and the schedulability analysis used, section IV presents the proposed heuristic. Finaly the section V concludes and gives some directions for future work.

II. Related work

A. State of works in multiprocessor scheduling

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

In the partitioned scheduling approach [5], [6] the set of tasks is divided into a number of disjoint subsets less than the number of processors in the multiprocessor architecture, and each of these subsets 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 NP-Hard [7]. This complexity is the main drawback of the partitioned scheduling approach.

There exist two classes of methods to solve allocation problems and more generally NP-Hard problems: the exact methods [8], [9] which examine all possible solutions and give the optimal solution (the best solution according to given criteria) but they have a very large execution time, and the approximate methods [9] which give the solutions very quickly compared to the exact methods but these solutions are only near optimal.

For the approximate methods we distinguish heuristics and metaheuristics [9], [10]. Metaheuristics are methods inspired from domains such that biology, chemistry, artificial intelligence, etc. They give near optimal solutions but they have an execution time larger than the heuristics. The heuristics methods are inspired from the considered domain, here the real-time scheduling, but the solutions produced are generally less close to the optimal than those obtained with metaheuristics. Since the allocation problem is NP-Hard heuristics are considered to be the best suited solutions when the execution time is crucial as in the rapid prototype phase of the design process. Davari and Dhall were the first to propose in [11] two preemptive scheduling algorithms RM-FF (Rate Monotinic First Fit) and RM-NF (Rate Monotinic Next Fit) to solve the fixed priority multiprocessor real-time scheduling problem. In the proposed algorithm, the uniprocessor RM algorithm [1] is used to verify if a task is schedulable on a processor, and respectively first-fit and next-fit bin-packing heuristics are used to achieve the allocation to the different processors. In both allocation heuristics the tasks are sorted in decreasing order of their periods before the allocation started. With RM-NF tasks are allocated to a processor, called curent processor, until the RM schedulability condition is violated. In this case the current processor is marked "full" and a new processor is selected. RM-FF tries to allocate, first, a task to the marked processor before allocating it to a new processor. In [10] a greedy heuristic is proposed to solve the problem of allocating tasks on a multiprocessor architecture while reducing the makespan, but the scheduling algorithm is non preemptive.

In the global scheduling approach [5], [6] 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 has a prohibitive cost.

In the semi-partitioned scheduling approach [12], [13], derivated from the partitioned scheduling approach, each task is allocated to a specific processor as long as the 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 partioned tasks are executed on specific processors without any migration. The semi-partitioned scheduling approach allows a reduction of the number of migrations. However, be aware that migrations have a prohibitive cost.

B. Our choices

The migrations cost in the global and semi-partition scheduling approaches lead us to choose the partitioned scheduling. Moreover, since the partitioned sheduling transforms the multiprocessor scheduling problem in several uniprocessor scheduling problems we can take advantage of the numerous research results obtained for the uniprocessor scheduling problem. Because we aim at rapid protyping we propose an allocation heuristic rather than metaheuristic or exact methods, and a schedulability test to verify if a task is schedulable on a specific processor. Bin-packing heuristics try to reduce the number of processors involving an increase of the makespan, i.e. the global response time of tasks on all the processors. On the other hand, multiprocessor architectures used in the industrial applications, we are interested in, have a fixed number of processors. This number of processors may be minimized later on but this is not the primary goal. That is the reason why we propose a greedy heuristic similar to the heuristic given in [10]. This heuristic allocates the tasks on the processors and, in addition, minimizes the makespan. This latter optimization is important when feedback control is intended, like in avionics, automotives, mobile robotics applications, etc.

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 in [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 leads to wasting, the approximation is low and thus a task system declared schedulable by, let say RM, may miss some deadlines during its real-time execution. Consequently, we propose to use the $\oplus$ operation [14], [15]. It is an algebraic operation that verifies either two tasks are schedulable, or not, taking into account the exact preemption cost.

III. Model and Schedulability analysis

Let $\Gamma_n = \{ \tau_1; \tau_2; \cdots; \tau_n \}$ be a system of n periodic real-time tasks where $\tau_i = (r_i^1; C_i; D_i; T_i)$ and $C_i \leq D_i \leq T_i$. Based on the typical characteristics of a periodic task, r_i^1 is the date of first activation, C_i is the WCET without any approximation of the preemption

cost [15], D_i the relative deadline, and T_i is the period of $_i$. We want to schedule the system of tasks Γ_n on m identical processors [1].

We use the scheduling operation $\oplus$ [15] to verify either a task is schedulable, or not, on a processor. This operation applied to a pair of tasks $(x; y)$, such that x has the highest priority, gives as result a task r, that is $r = x \oplus y$. As mentioned before, $\oplus$ takes into account the exact preemption cost suffered by the task y. Here is briefly the principle of that operation which is explained in details in [15]. It consists in replacing the available time units of the highest priority task x with the time units of the lowest priority task y. In order to do that, both tasks are initially referenced to the same time origin. Then, task x is rewritten according to the number of instances of task y in the LCM (Least Common Multiple) of both task periods. That latter operation allows the identification of the available time units in task x, but also the verifcation that task y does not miss its deadlines for each instance. Since task y can be preempted by task x the exact number of preemptions is counted for each instance of y. For each instance of y, using a "fixed point algorithm", the preemption cost is added to the WCET, without any approximation, in order to obtain the PET (Preemption Execution Time). If the amount of PET unit of times fits in the available time units in task x the task y is schedulable giving as result task r, otherwise it is not schedulable. Since $\oplus$ 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.

IV. 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 bactracking). The effectiveness of such greedy heuristic is based on the choice of the decision to built a new element of the solution. In our case the decision is taken according to a cost function which aims at minimizing the makespan.

A. Cost function for allocation

The cost function allows the selection of the best processor p_j to schedule a task $_i$. In our case this cost funtion is defined for a processor p_j and a task $_i$ to be the response time on p_j after the scheduling of $_i$ taking into account the exact preemption cost. The processor which minimizes this cost function of $_i$ among all the processors is considered to be the best processor to schedule the task $_i$. The minimization of the response time on the set of processors has the advantage of reducing the makespan.

[1] All the processors have the same computation power.

B. Principle of our allocation heuristic

We use a "list heuristic" [16] whose order allows the allocation of the tasks to the different processors. In our case, we initialize this list, called “set of canditate tasks”, with the set of tasks according to the decreasing order of their priorities. At each step of the heuristic, the task with the highest priority is selected among the set of candidate tasks, and we attempt to allocate it to its best processor according to the cost function presented previously. Of course, the task is actually scheduled on its best processor. Then, this task is removed from the set of candidate tasks.

We use the scheduling operation $\oplus$ to verify either a task is schedulable, or not, on a processor taking into account the exact preemption cost. The first activation date is crucial for the schedulabilty of a task $_i$. A non schedulable task can become schedulable if its date of first activation is modified [15]. In the proposed heuristic we exploit this property to improve the success ratio of the heuristic, as follows. When we attempt to allocate the task $_i$ to a processor p_j, if that task is not schedulable with its initial date of first activation r_i^1, before attempting to allocate it on another processor, we delay its date of first activation by incrementing its value of one time unit ($r_i^1 = r_i^1 + 1$), and that until the task $_i$ becomes schedulable or r_i^1 exceeds the value of the makespan computed in the precedent step. In this case the task $_i$ is not schedulable on the processor p_j. This principle allows a set of tasks to be schedulable while they where not with their initial date of first activation. However, that iterative search of a date of first activation, which leads to a schedulable task, increases the complexity of the heuristic.

V. Conclusion and future work

In this paper we have presented a greedy heuristic which allocates and schedules, on a multiprocessor architecture, a set of real-time tasks while reducing the makespan. In addition, this heuristic takes into account the exact preemption cost that must be carefully considered in safety critical applications we are interested in.

In future works we plan to study the complexity of the proposed heuristic, as well as its performance, by

Algorithm 1 Greedy heuristic

1: Initialize the candidate tasks W with the set of tasks in the decreasing order of their priorities, initialize the boolean variable $SystemTasksSchedulable$ to $true$
2: **while** W is not empty and $SystemTasksSchedulable = true$ **do**
3: Select in W the highest priority task $_i$
4: % We verify on each processor p_j if task $_i$ is schedulable.
5: **for** j=1 to m (the number of processors) **do**
6: **if** with its initial date of activation r_i^1, the task $_i$ is schedulable on p_j with the exact preemption cost (scheduling operation $\oplus$ [15]) **then**
7: Compute the cost function of task $_i$ on the processor p_j, i.e. the response time of $_i$ on p_j
8: **else**
9: **while** $_i$ is not schedulable on p_j and r_i^1 do not exceed the value of the makespan of the previous step **do**
10: $r_i^1 = r_i^1 + 1$
11: **end while**
12: **if** $_i$ is schedulable on p_j **then**
13: Compute the cost function of $_i$ on p_j with the new date of first activation of $_i$
14: **else**
15: $_i$ is not schedulable on p_j with the exact preemption cost
16: **end if**
17: **end if**
18: **end for**
19: % Now, using again the cost function, we choose the best processor for $_i$ among all the processors on which $_i$ is schedulable.
20: **if** $_i$ is schedulable on one or several processors **then**
21: Schedule the task $_i$ on the processor which minimizes the cost function
22: Remove the task $_i$ from W.
23: $SystemTasksSchedulable = true$
24: **else**
25: $SystemTasksSchedulable = false$
26: **end if**
27: **end while**

comparing it with an exact algorithm. In addition we plan to study the case of dependent tasks.

REFERENCES

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

[2] A. Burns, K. Tindell, and A. Wellings. Effective analysis for engnneering (J.)-63102 9.1Vvd-6311 prior5is scersnment.[2]