Dynamic Scheduling Algorithm for Parallel Real-time Graph Tasks

Manar Qamhieh, Serge Midonnet
Université Paris-Est, France
{manar.qamhieh,serge.midonnet}@univ-paris-est.fr

Laurent George
ECE-Paris, France
lgeorge@ece.fr

***Abstract*—In this paper, we propose a dynamic global scheduling algorithm for a previously-presented specific model of real-time tasks called "Parallel Graphs" [1], based on the Least Laxity First priority assignment policy "LLF", we apply LLF policy on each subtask in the graphs individually, taking in consideration their precedence constraints. This model of tasks is a combination of graphs and parallelism, in which each subtask in the graph can execute sequentially or parallel according to its number of processors defined by the model. So we study parallelism possibilities in order to find the best structure of the tasks according to the practical specifications of the system.**

I. Introduction

Physical constraints such as chip size and continuous heating forced processors' manufacturers to produce multi-processor systems, and as they are constantly growing, software parallelism has been widely studied and applied in practice.

However, parallelism in real-time embedded systems is still a rising challenge with many open questions to be studied. There are parallel task models exist in practice, such as the fork-join model used in OpenMP [2] and has been studied in [3], and a more general model of parallel tasks has been proposed recently in [4] which overcomes the restrictions of the fork-join model. Those models consider the tasks as a sequence of parallel and sequential tasks.

The graph model of real-time tasks is a general representation of the models described previously. In our previous work [1], we proposed a new model of real-time tasks called "Parallel Graphs", which is a combination of graphs and parallel tasks. By this we added an inner parallelism level to the graph as will be described in II. In this paper, we will extend our previous work by proposing parallelizing options a dynamic scheduling algorithm for the parallel graphs.

In this paper, firstly we will describe our task model in section II. In section III we will discuss the various parallelizing possibilities for parallel graphs. Section IV will present our dynamic scheduling algorithm. Finally in section V, we will finish the paper by concluding and giving perspectives.

II. Task Model

In [1] we presented previously a new model of real-time tasks called "parallel graphs". In this model, Each parallel real-time task is represented by a directed acyclic graph (DAG), which is a collection of subtasks and directed edges representing the execution flow of the subtasks and the precedence constraints between them.

Each parallel graph task τ_i consists of a set of q_i subtasks and it is denoted as:
$\tau_i = (\{\tau_{i,1}, \tau_{i,2}, ..., \tau_{i,q_i}\}, D_i, P_i)$, where D_i is the deadline of the graph and P_i is its period. Each subtask $\tau_{i,k}$ is represented as the following:
$\tau_{i,k} = \{c_{i,k}, m_{i,k}\}$, where $c_{i,k}$ is the total worst execution time of the subtask, and $m_{i,k}$ is the maximum degree of parallelism of $\tau_{i,k}$, which means that $\tau_{i,k}$ can be scheduled on $m_{i,k}$ parallel processors at the most.

Figure 1 shows an example of parallel graph task.

Precedence constraint means that each subtask can start its execution when all of its predecessors have finished theirs. If there is an edge from subtask $\tau_{i,u}$ to $\tau_{i,v}$, then we can say that $\tau_{i,u}$ is a predecessor of $\tau_{i,v}$, and $\tau_{i,v}$ has to wait for $\tau_{i,u}$ to finish its execution before it can start its own. Each subtask in the graph may have multiple predecessors, and multiple successors as well, but each graph should have a single source and a single sink vertex.

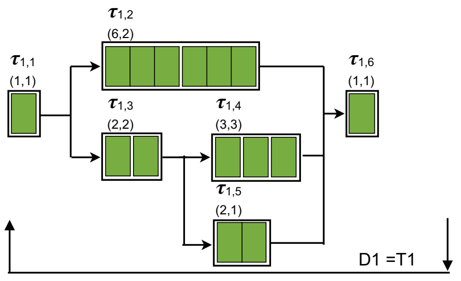

Figure 1. Example of the parallel graph model.

In this work, we study the global scheduling of n synchronous periodic parallel real-time graphs with implicit deadlines on m identical processor system. A task set is denoted as $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$, where each graph has period equals to its deadline. The schedulability is studied on the hyper period of each taskset.

A. Notation

Definition 1: Critical path [1] of a graph τ_i is the longest path in the graph through its subtasks when respecting their dependencies.

$$CP_i = \sum_{j \in \text{critical subtasks}} c_{i,j}$$

The critical path of the graph τ_1 from Figure 1 is $(\tau_{1,1}, \tau_{1,2}, \tau_{1,6})$ and $CP_1 = 8$.

Definition 2: The worst case execution time of a graph C_i is the total execution time of all the subtasks in the graph τ_i when executed sequentially.

$$C_i = \sum_{j=1}^{q_i} c_{i,j}$$

Definition 3: Laxity of a parallel graph L_i is the difference between its deadline and its critical path execution time.

$$L_i = D_i - CP_i$$

III. Different parallelizing algorithms

In our previous paper [1], we described a parallelizing algorithm which finds the best structure of the parallel graph according to the response time of the graph, by maximizing the number of parallelized subtasks in the graph. This algorithm does not consider the number of processors in the system, so there might be better structures of the parallel graphs when considering the actual specifications of the system like the number of processors, the scheduling policy, etc.

In this section we will discuss 2 possible parallelizing algorithms, a maximizing and minimizing algorithms according to the level of parallelism.

A. Maximizing parallelism

The parallelizing algorithm proposed in [1] is an iterative algorithm whose aim is to parallelize the maximum number of necessary subtasks in the graph up to their maximum level of parallelism, on the basis of the following algorithm:

- Find the critical path of the graph using the depth-first search algorithm.
- Parallelize all the subtasks in the critical path up to their maximum level of parallelism.
- Repeat the 2 previous steps until there is a critical path with no parallelizable critical subtasks.

B. Minimizing parallelism

Another approach can be considered for this type of graph which is the reverse of the one in III-A. We can find the best structure of the graph according to the number of processors in the system, by trying to stretch the graph as long as possible in order to execute on a minimum number of processors without missing their deadline, and by filling the laxity of the graph with a maximum number of subtasks.

According to the following equation, a parallel graph τ_i can execute sequentially on 1 processor if:

$$\frac{D_i}{C_i} \geq 1 \tag{1}$$

Where D_i is the deadline of the graph task, and C_i is defined in II-A.

If this test fails, then we have to reduce the sequential execution time of the graph by parallelizing some of its subtasks using the following algorithm:

- Apply equation 1 on the parallel graph τ_i.
- If the test succeeds, then τ_i can execute on a minimum number of processors without missing its deadline.
- If the test fails, we calculate the critical path of τ_i, parallelize the critical subtasks in order to reduce its sequential worst case execution time C_i.
- Repeat the first step on the newly parallelized graph τ_i until the test succeeds.

C. Other possibilities

Choosing the best structure of the parallel graph is not an easy process. It is controlled by a lot of restrictions and limitations of the system, for example, using the maximizing parallelism algorithm will reduce the response time of the graph if executed on a system with a large number of processors, and theoretically, it will increase the number of preemptions and job migrations while scheduling (depends on the used priority assignment algorithms).

The minimizing algorithm will increase the response time of the graph while reducing the number of processors needed, which will decrease the energy consumption as a result.

There is a large number of parallelizing structures for the parallel graph task, which affects the schedulability of the tasks, the migration and preemption costs. In the future we aim to study those various possibilities and their effects by comparing them using a real-time simulation tool, and taking into account the different characteristics of embedded systems, such as the number of processors, energy consumption, etc.

IV. Scheduling Parallel Graphs

In this section, we propose a global preemptive scheduling algorithm on an implicit-deadline parallel graph task set Γ_i of n graphs, on the hyper period of the taskset:

$$hyper(\Gamma_i) = LCM(\tau_j), \forall j : 1 \leq j \leq q_i$$[1]

Since the active subtasks of each graph share the same period and deadline, we decided to use a dynamic priority assignment policy based on the least laxity first technique (LLF), which gives higher priority to tasks with lower laxity (slack time). Scheduling algorithms based on this priority assignment are optimal on mono-processor systems but not on multiprocessor systems, unless laxity priorities are verified all the time during the scheduling process to make sure delayed tasks gain higher priorities in time.

A. Scheduling algorithm

After applying a parallelizing algorithm on each graph τ_i in Γ_i, the generated graph task contains both sequential and parallel subtasks. A sequential subtask needs one processor to execute on, while parallel subtasks execute on multiple processors at the same time. Here we should say that we are interested in input-data parallelism, in which the same code is repeated on multiple processors while only the input data are changed.

As described in Section II, each graph task τ_i consists of q_i real-time subtask each has a WCET of $c_{i,j}$, and due to the precedence constraint on the subtasks of the

[1]LCM is the Least Common Multiple

same graph, a subtask $\tau_{i,j}$ cannot be activated unless all of its predecessor subtasks finish their execution. Because of that not all of the subtasks in the graph are activated at the same time or at the very instant of activating the graph.

There are 2 types of laxity in τ_i, a general laxity of the graph as whole, denoted as L_i and described in II-A, and a subtask laxity for each subtask $\tau_{i,j} \in \tau_i$.

As explained in our previous work in [1], when a parallelizing algorithm is applied on a graph task τ_i, we can also calculate the laxity of each subtask in τ_i, by calculating the earliest and the latest finishing time of the execution time of each of them, the difference between those 2 time values is the laxity of the subtask. A subtask with laxity equaling to 0 is a critical subtask in the graph.

In order to organize the scheduling process of the graph set, we will consider 3 types of subtasks: active jobs, executing jobs and completed jobs. The active list contains the jobs that are activated either by the activation of the original graph (jobs of the starting subtask in the graph), or when the predecessors of a subtask complete their execution. When a job starts its execution, it will be moved to the executing list until its execution is over when it will be moved to the completed list. If an executing job is interrupted by a higher priority job, it will be moved back to the active list.

For each instant in time t where

$$0 \leq t \leq hyper(\Gamma_i),$$

we calculate the dynamic priority $Pr_{i,j}(t)$ for each active job of subtask $\tau_{i,j}$ of each graph task in Γ_i, where:

$$Pr_{i,j}(t) = L_i + L_{i,j} - (t - A_{i,j}) \qquad (2)$$

where $A_{i,j}$ is the activation time of $\tau_{i,j}$, subtasks with lower $Pr(t)$ values have higher priorities.

The scheduling algorithm:

- At t = 0:
  Each graph task $\tau_i \in \Gamma$ is activated (since we consider synchronous activation), which means all the starting subtasks $\tau_{i,1}$ are activated as well and added to the active list. Then we calculate $Pr(0)$ for all of the subtasks in the active list using Equation 2.
  According to the number of processors in the system, we start executing the subtasks with the highest priority which are moved to the executing list at the same time.
  If an executing subtask $\tau_{i,j}$ is a parallel subtask according to the parallelizing algorithm, then it will need $m_{i,j}$ available processors in order to enable all of its parallel parts to execute concurrently.
- $\forall t$ where $0 < t \leq hyper(\Gamma_i)$:
  If an executing job finishes its execution at this instant of time, it will be moved to the completed list. And if all the predecessors of a subtask are in the completed list, then its job will be activated at this instant of time and added to the active list.
  Since we use dynamic priority assignment, at each instant of time we re-calculate $Pr_{i,j}(t)$ for all subtasks $\tau_{i,j}$ in the active list. By this, the priority of delayed active subtasks will be increased in time since their laxity decreased.
  According to the newly assigned priorities of the active subtasks, we fill the processors available in the system, and if there are active subtasks with higher priority than the ones already executing, they are allowed to interrupt their execution.
- The scheduling algorithm of the graph set Γ will fail if -at any instance of time t- a job in the active list reaches a $Pr(t) = 0$ without having an available processor to execute on, since at $t + 1$ this job will have a negative laxity and miss its deadline.

In the case of equal priorities between 2 active jobs, we choose the executing one randomly, but if an active job and an executing job have the same priority, we give the priority to the executing job to continue its execution without allowing the active one to interrupt the execution of the other. In that way we reduce the number of unnecessary cost of preemptions and migration.

Using this dynamic global scheduling algorithm, we scheduled each subtask in the graphs individually, by assigning priorities as shown in the previous algorithm. Precedence constraints between the subtasks due to the structure of the graph were visible only in the activation process. However, this scheduling algorithm was different from the previous scheduling techniques applied on the real-time tasks of the graph model seen in the literature. For example in [4], the authors propose to use a decomposition algorithm in order to assign local deadlines to the subtasks in the task, and to schedule each segment of tasks as independent tasks on multi-processor systems.

B. Scheduling example

In this section, we will apply the above-mentioned scheduling algorithm described in IV-A on a graph task set consisting of 2 graphs on a 2-processor system.

graph task set $\Gamma = \{\tau_1, \tau_2\}$
$\tau_1 = (\{\tau_{1,1}(1,1), \tau_{1,2}(3,1), \tau_{1,3}(2,2), \tau_{1,4}(1,1)\}, 10, 10)$
$\tau_1 = (\{\tau_{2,1}(1,1), \tau_{2,2}(1,1), \tau_{2,3}(1,1), \tau_{2,4}(1,1)\}, 5, 5)$

The graphs in the task set have implicit deadlines, and all subtasks of the same graph share the same period and deadline. The scheduling algorithm is studied on the hyper period of the task set:

$$hyper(\Gamma) = LCM(10, 5) = 10$$

Figure 2 shows the graphs of the task set with the precedence constraints.

Graph τ_1 has a parallel subtask $\tau_{1,3}$ which needs 2 processors available at the same time in order fo it to execute, or its execution will be delayed otherwise. All the other subtasks in this example are sequential.

By applying the critical path calculations described previously in [1], we find that the laxity of the graph τ_1 equals to 5, and 2 for τ_2, and all the subtasks of both

Table I
SCHEDULING TABLE OF 2 GRAPH TASKS

	Active subtasks in order of priority			
Time	Highest			Lowest
t=0	$Pr_{2,1} = 2$	$Pr_{1,1} = 5$		
t=1	$Pr_{2,2} = 2$	$Pr_{2,3} = 2$	$Pr_{1,2} = 5$	$Pr_{1,3} = 6$
t=2	$Pr_{2,4} = 2$	$Pr_{1,2} = 4$	$Pr_{1,3} = 5$	
t=3	$Pr_{1,2} = 4$	$Pr_{1,3} = 4$		
t=4	$Pr_{1,3} = 3$	$Pr_{1,2} = 4$		
t=5	$Pr_{2,1} = 2$	$Pr_{1,2} = 3$	$Pr_{1,3} = 3$	
t=6	$Pr_{2,2} = 2$	$Pr_{2,3} = 2$	$Pr_{1,3} = 2$	
t=7	$Pr_{1,3} = 1$	$Pr_{2,4} = 2$		
t=8	$Pr_{2,4} = 1$	$Pr_{1,4} = 1$		
t=9				
t=10				

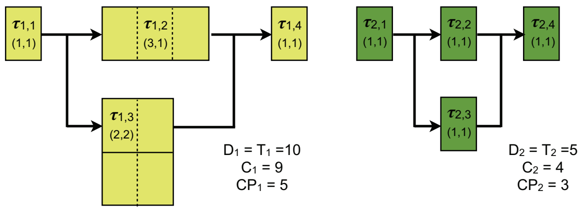

Figure 2. Graph taskset example.

graphs don't have local laxities (they are critical subtasks), except for subtask $\tau_{1,3}$ which has a laxity $L_{1,3} = 1$.

At $t = 0$, the first subtasks of the each graph are activated ($\tau_{1,1}$& $\tau_{2,1}$), according to the mentioned-above equation 2, we can calculate the priority of the subtasks as the following:

$$Pr_{1,1} = 5 + 0 - (0 - 0) = 5$$
$$Pr_{2,1} = 2 + 0 - (0 - 0) = 2$$

According to the results, $\tau_{2,1}$ has higher priority than $\tau_{1,1}$, but since we have 2 processors available, both subtasks will be scheduled. Those calculations will be repeated at each instant of time in the hyper period of the task set, unless a deadline miss occurs before the end of the period. Table IV-B shows the priorities of the active subtasks of both graphs over the hyper period of the task set, while Figure 3 shows the final scheduling of the subtasks on the 2 processors of the system. We can notice that our proposed global preemptive scheduling algorithm based on LLF has succeeded in scheduling the taskset without any subtask misses its deadline.

V. PERSPECTIVE AND CONCLUSION

In this paper, we have introduced a dynamic global scheduling algorithm on multi-processor systems, for a specific real-time task set of parallel graph models, based on the Least Laxity First "LLF" job priority assignment in order to schedule each subtask in the graphs according to their laxity while considering the global deadline and period of the original graph with no need to assign a local deadline for them, we have also shown by an example the schedulability of this algorithm on a task set of the graph model.

The graph model of tasks has been studied recently in the literature, but adding the parallelism constraint to this model has raised a schedulability challenge and made it more complicated. That is why we presented 2 parallelizing algorithms in this paper, both algorithms depending on the constraints of the embedded systems which we aim to study in more details in the future.

In order to provide valid results to show the performance of our proposed scheduling algorithm, we started implementing it on a simulation tool called "YARTISS" [5], developed by a real-time team in the research laboratory of Universit Paris-Est. By using this simulator we will be able to compare the performance of our own scheduling algorithm with other techniques and algorithms used in the literature, which will allow us to enhance its performance with respect to the practical issues of real embedded systems such as a limited number of processors, optimizing the schedulability in order to reduce the energy consumption by reducing the number of migrations and preemptions.

In parallel, we would like to study real-time scheduling anomalies and provide real-time feasibility tests for the proposed algorithm, in order to support the simulation results.

Finally, we hope to apply our model of tasks on real embedded systems, and propose adjustable techniques in order to enhance their performance such as schedulability and energy consumption.

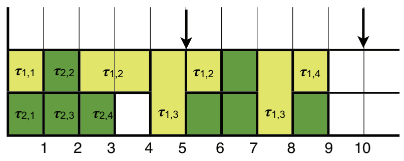

Figure 3. Graph taskset scheduled using LLF.

REFERENCES

[1] M. Qamhieh, S. Midonnet, and L. George, "A Parallelizing Algorithm for Real-Time Tasks of Directed Acyclic Graphs Model," in *RTAS Work-In-Progress Session*, 2012.

[2] "Openmp." [Online]. Available: http://www.openmp.org

[3] K. Lakshmanan, S. Kato, and R. (Raj) Rajkumar, "Scheduling Parallel Real-Time Tasks on Multi-core Processors," in *Proceedings of the 31st IEEE Real-Time Systems Symposium*, 2010.

[4] A. Saifullah, K. Agrawal, C. Lu, and C. Gill, "Multi-core Real-time Scheduling for Generalized Parallel Task Models," in *The 32nd IEEE Real-Time Systems Symposium*, 2011.

[5] Y. Chandarli, F. Fauberteau, D. Masson, S. Midonnet, and M. Qamhieh, "YARTISS: A Tool to Visualize, Test, Compare and Evaluate Real-time Scheduling Algorithms," in *WATERS*, 2012.