Graph-to-Segment Transformation Technique minimizing the number of processors for Real-time Multiprocessor Systems

Manar Qamhieh, Serge Midonnet, Laurent George

Graph-to-Segment Transformation Technique minimizing the number of processors for Real-time Multiprocessor Systems

Manar Qamhieh, Serge Midonnet, Laurent George
LIGM, Université Paris-Est, France
{manar.qamhieh,serge.midonnet}@univ-paris-est.fr, lgeorge@ieee.org

***Abstract*—In energy-aware real-time systems, there are techniques to optimize the energy consumption through turning off idle processors, Dynamic Power Management (DPM) is an example of such techniques. In this paper we propose a transformation technique of graph task model into the multi-thread segment task minimizing the number of processors needed to schedule the tasks. This reduces the energy consumption of the system when DPM like technique is used. The Directed Acyclic Graph task is presented as a directed graph of subtasks under precedence constraints, while the other task is called the multi-threaded segment model, in which a task is a sequence of segments, and each segment has a number of threads and an intermediate deadline. The graph model is more general than the segment model, but it is more complicated to schedule and analyze, because in the segment model, the threads of each segment are scheduled as independent sequential tasks on multiprocessor platform. Due to the dependencies between the subtasks of a graph, it could have a number of possible segment combinations. In this paper, we propose a graph-to-segment transformation technique, which generates a multi-thread segment task with minimum number of processors required to execute on multiprocessor platform.**

I. Introduction

Chip manufacturers tend to build multi-processors and multi-core processors as a solution to overcome the physical constrains of the manufacturing process, such as chip's size and heating. As a result, parallel programming has gained a higher importance and parallel programming APIs have been used for many years, such as OpenMP[1], pThreads and others.

For systems having energy constraints, Dynamic Power Management (DPM) is considered as a powerful technique to reduce energy consumption [2]. Its principle is to turn off the idle processors in the system. Minimizing the number of processors required to run an application is thus a way to reduce energy consumption when DPM is used. In this paper, we focus on the problem of scheduling real-time parallel graphs on identical multiprocessors. We propose to transform the graphs into multi-thread segments to solve the real-time scheduling problem. we show that this transformation minimizes the number of processors required to schedule the taskset. This reduces the energy consumption when idle processors are turned off with DPM.

The remainder of this paper is organized as the following, in Section II, we present our task model and the target task model, with few definitions and notations used in the rest of the paper in Section III. Section IV describes a related work used in our transformation technique. Section V explains the proposed transformation technique followed by the analysis in Section VI. and we finish with perspective and the conclusion in Section VII.

II. Task Model

A. Directed Acyclic Graph model

We consider a parallel constrained-deadline real-time task to be scheduled on heterogeneous multiprocessor platform. Each task is represented as a Directed Acyclic Graph (DAG), where the nodes represent the different subtasks in the graph, and the edges represent the precedence constraints.

Each graph τ_i consists of a set of n_i subtasks and a deadline D_i and a period T_i, where $D_i \leq T_i$. Each subtask is denoted by $\tau_{i,j}$, $1 \leq j \leq n_i$, with n_i is the total number of subtasks in the graph τ_i. Each subtask $\tau_{i,j}$ has a worst case execution time denoted by $c_{i,j}$, and all subtasks share the same deadline and period.

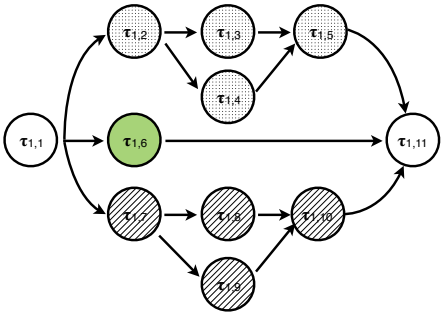

Figure 1. A example of a graph task τ_1 on which we will apply the transformation technique.

their

As shown in Figure 1, the precedence constraints of a graph model mean 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 $\tau_{i,u}$ 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.

B. The multi-thread segments model

In this model, a real-time implicit deadline task τ'_i is represented as a set of n_i sequential segments $\Gamma_i = \{\sigma_{i,1}, \sigma_{i,2}, ..., \sigma_{i,n_i}\}$. Each segment $\sigma_{i,j}$ is a collection of $n_{i,j}$ threads, each thread $\tau^k_{i,j}$ has a worst case execution time of $C^k_{i,j}$. All the segments share the same period T_i, which is the global period of the task. However, the global deadline of the task D_i has to be split into intermediate deadlines $d_{i,j}$ for each $\sigma_{i,j}$, where $\sum d_{i,j} = D_i$. As a result, the threads of each segment can be scheduled as independent real-time tasks on multiprocessor platform, because the threads of the same segment can be considered as independent threads, each with an offset and an intermediatheirte deadline. Segment $\sigma_{i,j}$ is activated when all its predecssors finish their execution, so its offset is $\sum_{l<j} d_{i,l}$.

In the multi-thread segment model, each segment executes individually on the system without interference from the other segments of the same task.

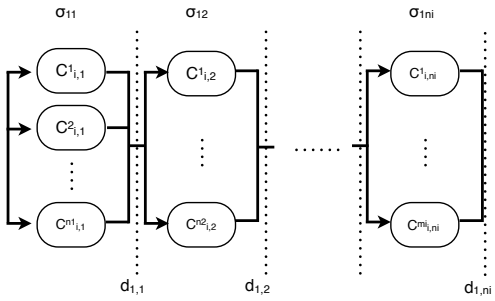

Figure 2. parallel real-time task of sequential segment model.

Figure 2 shows the model of multi-thread segment task.

In this work, we consider multi-thread model of parallelism, in which threads can execute in parallel independently from each other. This model of parallelism is different from the gang model, in which all threads have to execute simultaneously on multiple processors.

III. Definitions and notations

Graph Model:

- *Worst case execution time* C_i of graph τ_i is the maximum execution time of the graph if all its subtasks are executed sequentially on one processor.

$$C_i = \sum_{1 \le j \le n_i} c_{i,j}$$

- *Graph's critical path* is a directed path through the graph with the maximum execution requirement among all other paths in the graph.
- *The critical paththeir length* P_i is the time needed to execute the critical path of the graph, which is the minimum execution time of the graph τ_i.

$$P_i = \sum_{\tau_{i,j} \in critical} c_{i,j}$$

- *Critical subtask* $\overline{\tau_{i,j}}$ is subtask $\tau_{i,j}$ forming the critical path of the graph. If a critical subtask in a graph is delayed, the response time of the graph will be increased as a result.

Multi-thread Segment Model:

- For segment $\sigma_{i,j}$ in τ'_i, the maximum execution time $C_{i,k}$ is the total execution time of its threads, $C^m_{i,j}$ is its minimum execution time.
  $C_{i,j} = \sum_{1 \le k \le n_{i,j}} C^k_{i,j}$.
  $C^m_{i,j} = \max_{1 \le k \le n_{i,j}} C^k_{i,j}$.

Common definitions:

- *The density* δ_i of a real-time task τ_i is the average computing capacity needed to execute a job of τ_i between its arrival time and its deadline.

$$\delta_i = \frac{c_i}{D_i}$$

- *The average density* δ^*_i of a task (either graph or segments) τ_i is the total worst case execution time of the task C_i divided by its deadline D_i.

$$\delta^*_i = \frac{C_i}{D_i}$$

- *The upper bound density* $\widehat{\delta_i}$ of a segment $\sigma_{i,j}$ of task τ_i is the maximum density can be granted to $\sigma_{i,j}$. It can be achieved when the segment has an intermediate deadline $d_{i,j}$ equal to the the maximum execution time of its threads.

$$\widehat{\delta_i} = \frac{\max_{\tau^k_{i,j} \in \sigma_{i,j}} C^k_{i,j}}{d_{i,j}}$$

IV. Related work

A common parallel task model is the fork-join model, and it has been studied as a real-time task model in [3]. This model is defined as a master thread that forks into a number of parallel threads, then they all join again after a period of time into the master thread. According to this model, a parallel task is a sequence of alternative parallel and sequential segments.The authors of this paper proposed a stretching algorithms to schedule the tasks of this model.

As a generalization of the fork-join model, the authors in [4] proposed the multi-thread segment model (described above). A task of this model has a global deadline shared between all the segments of the task, and the challenge is to split the global deadline of the task into a group of

intermediate deadlines for each segment in the task, and find so as the threads of each segment can be scheduled as independent sequential tasks with intermediate deadline.

Based on the previous paper, the authors in [5] proposed an optimization technique for the intermediate deadline assignment algorithm. The optimization technique is proved to be optimal regarding the number of processors. According to their optimization, their algorithm assigns intermediate deadline to segments by imposing the average density of the task to the maximum number of segments. Minimizing the density of the segments will minimize the number of processors needed to schedule the task.

The proposed optimization technique is applied to multi-thread segment tasks, the authors proposed a solution to apply it on the graph model. However, their solution has an exponential complexity and suggests to test all the possible segments of a graph. In this work, we will propose a simpler solution, to transform the graphs into a segment task, which maintains the optimality of the deadline assignment technique, to maximize the number of idle processors in the system which can be turned off using techniques like DPM[2].

V. Transformation technique

A. Motivation

Due to the special structure of the DAG model, and due to the inter-subtask parallelism (multiple subtasks in a graph execute in parallel according to their precedence constraints), a single DAG task has multiple combinations of sequential segment model. Figure 3(a) shows a simple graph task τ_i, which consists of five subtasks (the number inside the square denotes the index of the subtask, for example, number 1 refers to $\tau_{i,1}$). The critical path of τ_i is $\{\tau_{i,1}, \tau_{i,2}, \tau_{i,3}, \tau_{i,5}\}$, and subtask $\tau_{i,4}$ forms another path in the graph which executes in parallel with the critical path, since both paths are activated at the same time t and they share the same starting and ending subtasks. As shown in the example, subtask $\tau_{i,4}$ can be executed in parallel with either subtask $\tau_{i,2}$ or $\tau_{i,3}$ or both at the same time. This will form three possible combinations of sequential segments based on the original graph. Figure 3 shows all possible segment tasks for the graph in Figure 3(a).

The optimal intermediate deadline assignment algorithm proposed by [5] is applied on the multi-thread segment model. The optimality of the algorithm is related to the number of processors needed to schedule the task. This iterative algorithm assigns intermediate deadlines for the segments of the task, to be scheduled with the minimum number of processors. The authors propose possible generalizations of the tasks into graphs. In order to maintain the optimality of their algorithm when applied to graphs, all the structures of generated segments have to be studied. The proposed solution has an exponential complexity based on the number of constraints and subtasks in the graph.

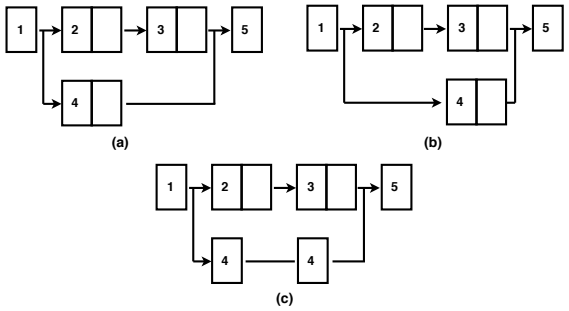

Figure 3. The three possible combinations of sequential segments of a graph task.

The process of choosing the best sequential segment combination depends on the desired criteria. This means that the sequential segment combination, most suitable for optimizing the number of processors on which the graph executes, might not be the most feasible combination for example, and so on. However, in this work, we aim to provide a technique to transform a graph into a sequential segment task with average fair density distribution among the segments. Applying this rule, we are choosing the sequential segment combination which is adapted to the optimal intermediate deadline assignment algorithm proposed by [5]. For example, the sequential segments in Figure3(c) is the combination with the most fair density distribution among the segments, since subtask $\tau_{i,4}$ will be executed in parallel with both $\tau_{i,2}$ and $\tau_{i,3}$.

B. Average density bounds

For scheduling real-time tasks on multiprocessor systems, the following necessary feasibility test should apply:

A task set Γ is feasible on m identical multiprocessor platform if, at any time t, we have $\delta(t) \leq m$.

This feasibility test can be applied to a parallel real-time task model. Since the subtasks of each graph with precedence constraints or the threads of segments can be considered as independent subtasks with certain offsets and deadlines executing on multiprocessor platforms. According to this, the maximum execution time of the task C_i can be either less, equal or greater than its deadline D_i.

$C_i \leq D_i$: If the total execution time of a parallel task is less than its deadline, the whole graph task τ_i can execute sequentially on one processor, since its density is less than one.

$$C_i \leq D_i \ \rightarrow \ \frac{C_i}{D_i} \leq \frac{D_i}{D_i} \ \rightarrow \ \delta_i \leq 1$$

According to this, we are interested in the case where $C_i \geq D_i$ in this work. And since the critical path length P_i of a parallel task is the minimum execution time of the

task, the deadline of the task D_i should be greater or equal to P_i, this can be considered as a necessary feasibility condition.

$P_i \leq D_i$: If the condition is not satisfied then the task τ_i is not feasible, because the density of the critical path will be greater than 1, which means τ_i is not feasible on single processor.

$$P_i \geq D_i \ \rightarrow \ \frac{P_i}{D_i} \geq \frac{D_i}{D_i} \ \rightarrow \ \delta_i \geq 1$$

As the result, we can conclude the following bounds for the average density of the task:

$$\begin{aligned} P_i \ &\leq \ D_i \ < \ C_i \\ \frac{C_i}{P_i} \ &\geq \ \frac{C_i}{D_i} \ > \ \frac{C_i}{C_i} \\ 1 \ &< \ \delta_i^* \ \leq \ \frac{C_i}{P_i} \end{aligned}$$

We will use the upper bound of the average density in our transformation, so as to give the generated segments the highest possible average density.

C. Graph-to-Segments transformation technique:

The basic idea of the graph-to-segment transformation is to transform the graph task into a set of sequential segments, the subtasks of the graph will be distributed among the different segments, based on the precedence constraints of the subtasks and the densities of the segments.

In our technique, we define the critical path of the graph as the principal path in the graph, since it is the longest path in the graph with no laxity. All the other paths in the graph will have laxity values greater or equal to zero. The critical subtasks will define the initial segments of the set. We call it an initial number of segments because we might need to create more segments by the end of the technique. If there a fork or join event between two subtasks in a segment, this segment has to be split into two segments at the time of the event. Figure 1 shows a graph task τ_1 which consists of 11 subtasks and 3 execution paths. $\tau_{1,1}$ is the starting subtask and $\tau_{1,11}$ is the ending subtasks, according to our model, those subtasks do not execute in parallel with the rest of the graph, and their upper bound density is always equal to one.

Table I shows the WCETs for each subtask in τ_1. For task τ_1, there are three pathsAvgDensity ρ executing in parallel:

- $\varrho_1 = \{\tau_{1,2}, \tau_{1,3}, \tau_{1,4}, \tau_{1,5}\}$
- $\varrho_2 = \{\tau_{1,6}\}$
- $\varrho_3 = \{\tau_{1,7}, \tau_{1,8}, \tau_{1,9}, \tau_{1,10}\}$

The path ϱ_1 is the critical path of the graph.

Figure 4 shows the time diagram of the graph τ_1 and the execution behavior of the subtasks of the graph and their precedence constraints, as well as the laxity of each path. The figure shows also the initial segments of the graph, generated from the subtasks of the critical path. For each segment $\sigma_{i,j}$, its length is the maximum WCET of its subtasks $C_{i,j}^m$, and its upper bound density $\widehat{\delta_{i,j}}$ is the total WCETs of its subtasks divided by its length. According to the example in Figure 4, the upper bound densities for each segment is represented by the following set: $\{1, 1, 1.5, 1, 1\}$, where the first value in the set is the upper bound density of $\sigma_{1,1}$ and so on.

$\tau_{i,j}$	$c_{i,j}$	$\tau_{i,j}$	$c_{i,j}$
$\tau_{1,1}$	1	$\tau_{1,2}$	2
$\tau_{1,3}$	2	$\tau_{1,4}$	1
$\tau_{1,5}$	2	$\tau_{1,6}$	4
$\tau_{1,7}$	1	$\tau_{1,8}$	2
$\tau_{1,9}$	1	$\tau_{1,10}$	1
$\tau_{1,11}$	1		

Table I
WCETs OF THE SUBTASKS IN τ_1

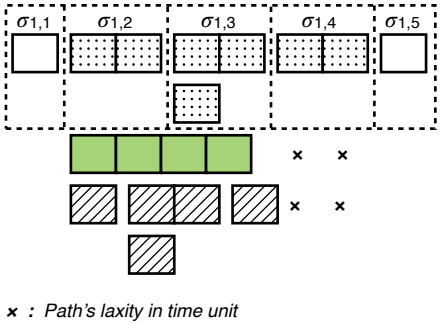

Figure 4. The time diagram of τ_1 showing the WCET of each subtask.

However, there is no parallelism in the starting and the ending subtasks of a graph, so they are executed sequentially on one processor only, and the density of these segments is always one, and there is no need to include them in the transformation algorithm.

D. Transformation algorithm:

In this section, we will present our graph-to-segment transformation technique, then we will apply it on the graph τ_1 in Figure 1. In this transformation we will use the upper bound value of the average density of the graph described above, which is $\delta_i^* = \frac{C_i}{P_i}$.

- Determine the critical path of a given real-time graph τ_i. Each critical subtask in the graph $\overline{\tau_{i,j}}$ will form a segment $\sigma_{i,j}$ in the new output multi-thread segment task. The minimum execution time of $\sigma_{i,j}$ is the execution time of $\overline{\tau_{i,j}}$.
- Calculate the average density of δ_1^* of task τ_1.
- Impose the average density δ_i^* on all the initial segments of the task where $\delta_i^* = \frac{C_i}{P_i}$, so as the density of each segment $\sigma_{i,j}$ is equal to δ_i^*. The minimum execution time $C_{i,j}^m$ of the segment is fixed, so we can

distribute the execution time of the rest of the task τ_i on the segments.

$$\delta_i^* = \frac{X_{i,j} + C_{i,j}}{C_{i,j}^m}$$

$$X_{i,j} = \delta_i^* * C_{i,j}^m - C_{i,j}$$

$X_{i,j}$ is the needed execution time to be added to segment $\sigma_{i,j}$ so as to maintain its density equal to δ_i^*.

- For each path ϱ_k in the graph other than the critical path, we will calculate a distribution factor f_k. This factor indicates how much execution capacity each path has in relation to the total capacity of the other paths (all the graph without the critical segments).

$$f_k = \frac{C_{i,k}}{\sum_{\varrho_l \in \rho} C_{i,l}}$$

Using this factor f_k, the execution time of each path ϱ_k will be fairly distributed among the segments of the critical path.

$$X_{i,j} = \sum_{\varrho_k \in \rho} f_k * C_{i,k}$$

Algorithm 1 Graph-to-Segment transformation algorithm.

Input: τ_i is a graph task
$\varrho_c := critical\_path(\tau_i)$
$\delta_i^* := \frac{C_i}{D_i}$
$C_{i,j}^m := \max C_{i,j}^k$
$C_{i,j} := \sum C_{i,j}^k$
for $\forall$ *path* $\varrho_l \in \tau_i$ **do**
$f_l = \frac{C_{i,l}}{\sum_{\varrho_h \in \rho} C_{i,h}}$
end for
for $\forall$ *segment* $\sigma_{i,j} \in \varrho_c$ **do**
$\delta_{i,j}^* \leftarrow \delta_i^*$
$X_{i,j} \leftarrow \delta_i^* * C_{i,j}^m - C_{i,j}$
$X_{i,j} = \sum_{\varrho_k \in \rho} f_k * C_{i,k}$
end for

E. Example:

- Based on the Graph task τ_1 in Figure 1, the critical path $\varrho_1 = \{\tau_{1,1}, \tau_{1,2}, \tau_{1,3}, \tau_{1,5}, \tau_{1,11}\}$. We can identify the segments of the output task as $\Gamma_1 = \{\sigma_{1,s}(\tau_{1,1}), \sigma_{1,1}(\tau_{1,2}), \sigma_{1,3}(\tau_{1,2}, \tau_{1,4}), \sigma_{1,3}(\tau_{1,5}), \sigma_{1,e}(\tau_{1,11})\}$, as shown in Figure 5.
- We are interested in the segments with parallel paths, which are $\sigma_{1,1}, \sigma_{1,2}, \sigma_{1,3}$. we calculate the average density of the graph:
$\delta_1^* = \frac{16}{6} = 2.67$.
We impose this density to all the segments as shown in Figure 5.

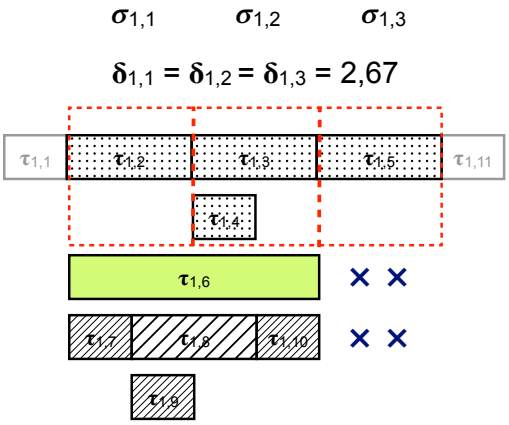

Figure 5. The first iteration of the transformation technique applied on graph τ_1.

- For each segment $\sigma_{i,j}$, we calculate the total execution time $X_{i,j}$ needed to maintain its density:
$X_{1,1} = (2.67 * 2) - 2 = 3.34$
$X_{1,2} = (2.67 * 2) - 3 = 2.34$
$X_{1,3} = (2.67 * 2) - 2 = 3.34$

- There are two paths in the graph execute in parallel with the critical path: $\rho = \{\varrho_1(\tau_{1,6}), \varrho_2(\tau_{1,7}, \tau_{1,8}, \tau_{1,9}, \tau_{1,10})\}$. The distribution factor for each path is calculated as the following:
$f_1 = \frac{4}{9} = 0.44$
$f_1 = \frac{5}{9} = 0.56$

- For each segment and according to the capacity load factor of each path, we will split the subtasks on the segments as the following:
$X_{1,1} = X_{1,3} = (0.44 * 3.34) + (0.56 * 3.34)$
$X_{1,1} = X_{1,3} = (1.48) + (1.86)$
Where 1.48 is the execution time from path ϱ_1 and 1.86 is from ϱ_2.
$X_{1,2} = (0.44 * 2.34) + (0.56 * 2.34)$
$X_{1,2} = (1,04) + (1,3)$

- Figure 6 shows the final result of the algorithm, which transformed the graph τ_1 into a sequence of segments, the upper bound density of each segment is as the following:

$\widehat{\delta_{1,s}} = \widehat{\delta_{1,e}} = 1$
$\widehat{\delta_{1,1}} = \widehat{\delta_{1,2}} = \widehat{\delta_{1,3}} = \delta_1^* = 2.67$

VI. Analysis of the Transformation Technique

The graph-to-segment transformation technique we proposed above, is adapted to the optimal intermediate deadline assignment algorithm provided in [5], which maintains the optimality of the deadline assignment algorithm, when it is

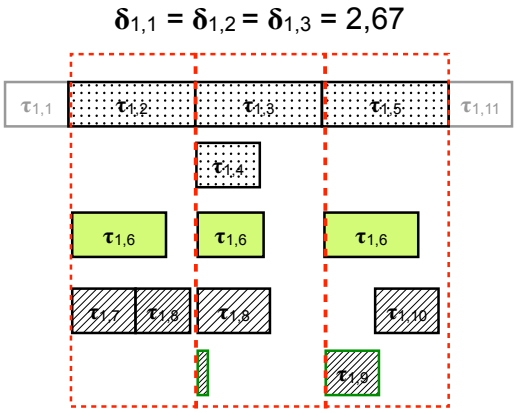

Figure 6. The second iteration of the transformation technique applied on graph τ_1.

applied on the generated multi-thread task generated by our transformation technique.

Theorem 1. *the graph-to-segment transformation algorithm will transoform a graph real-time task into a multi-thread segment real-time task. The output task has the optimal number of processors when compared with the rest of the possible segment tasks of the same graph.*

Proof: The optimality of the output multi-thread segment means that the task will the minimum possible number of processors when executed on multi-processor platform, while the other possible segment tasks of the same input graph will need more porcessors to execute.

In order to prove the optimality of our transformation, we will start by explaining the intermerdiate-deadline-assignment optimization technique proposed by [5]. The authors of this paper proposed an iterative algorithm that imposes the average denisty of the task to the maximum number of segments in the task. This is justified by the following property [5]:

Property 1. *The maximum instantaneous density of any solution cannot be smaller than the average density of the task.*

Based on this, imposing the average density on a set of segments guarantees minimizing the number of processors needed to schedule those segments. on another hand, the optimization technique is iterative algorithm. At each iteration, the algorithm compares the least upper bound density $\widehat{\delta_{i,j}}$ of the segment set with the average density of the set δ_i^*, if $\widehat{\delta_{i,j}} < \delta_i^*$, then $d_{i,k} = C_{i,k}^m$, and $\sigma_{i,k}$ will be omitted from the segment set in the next iteration. If $\widehat{\delta_{i,j}} \geq \delta_i^*$, then δ_i^* will be the density of the rest segments in the set, and the deadline of each segment will be $d_{i,j} = \frac{C_{i,j}}{\delta_i^*}$ (for more details, check Algorithm 1 in [5]).

It is proven as well in [5] that the average density of segments will increase by the increase of the number of iterations.

According to our transformation, the generated multi-thread segment task consists of a set of segments, each segment $\sigma_{i,j}$ (except the starting and the ending segment) has an upper bound density $\widehat{\delta_{i,j}} = \frac{C_i}{P_i}$. This density value is proven in Section V-B to be the upper bound density of the task. According to this, and when applying the transformation technique on this segment set, the second case of the algorithm will be applied always, since $\widehat{\delta_{i,j}} \geq \delta_i^*$ is always true, and the optimization technique will succeed in finding an intermediate-deadline assignment from the first iteration, which will guarantee the minimum possible denisty for all the segments of the set, and the minimum possible number of processors for the task to execute on as a result.

The generated segment structure of the graph will dominate the other possibilities regarding the number of processors, and it is well adapted to the optimization technique of [5]. In other words, the other segment structures will need more processors to be scheduled on if they use the same deadline algorithm to assign their intermediate deadlines.

VII. Conclusion

In this paper, we provided a transformation technique to transform real-time parallel graphs into a sequential multi-thread segment task. The segments of the generated task have average density segments, hence, we are sure to maintain the optimality of the optimization technique of [5], which minimizes the number of processors and thus reduces the energy consumption of the processors in the system with DPM.

In the future we aim to better study the scheduling of a task set on multiprocessor platforms, and provide more energy-aware analysis for the scheduling, and perform as well some simulation analysis to study the transformation effects and advantages.

References

[1] "OpenMP." [Online]. Available: http://www.openmp.org

[2] S. Irani, S. Shukla, and R. Gupta, "Algorithms for power savings," *ACM Trans. Algorithms*, vol. 3, no. 4, 2007.

[3] K. Lakshmanan, S. Kato, and R. (Raj) Rajkumar, "Scheduling Parallel Real-Time Tasks on Multi-core Processors," in *RTSS*, 2010.

[4] A. Saifullah, K. Agrawal, C. Lu, and C. Gill, "Multi-core Real-time Scheduling for Generalized Parallel Task Models," in *RTSS*, 2011.

[5] G. Nelissen, V. Berten, J. Goossens, and D. Milojevic, "Techniques Optimizing the Number of Processors to Schedule Multi-threaded Tasks," in *ECRTS*, 2012.