Techniques Optimizing the Number of Processors to Schedule Multi-Threaded Tasks

Geoffrey Nelissen[1], Vandy Berten, Joël Goossens, Dragomir Milojevic
Parallel Architectures for Real-Time Systems (PARTS) Research Center
Université Libre de Bruxelles (ULB)
Brussels, Belgium

***Abstract*—These last years, we have witnessed a dramatic increase in the number of cores available in computational platforms. Concurrently, a new coding paradigm dividing tasks into smaller execution instances called *threads*, was developed to take advantage of the inherent parallelism of multiprocessor platforms. However, only few methods were proposed to efficiently schedule hard real-time multi-threaded tasks on multiprocessor.**

In this paper, we propose techniques optimizing the number of processors needed to schedule such sporadic *parallel* tasks with constrained deadlines. We first define an optimization problem determining, for each thread, an intermediate (artificial) deadline minimizing the number of processors needed to schedule the whole task set. The scheduling algorithm can then schedule threads as if they were *independent sequential* sporadic tasks. The second contribution is an *efficient* and nevertheless *optimal* algorithm that can be executed online to determine the thread's deadlines. Hence, it can be used in dynamic systems were all tasks and their characteristics are not known *a priori*. We finally prove that our techniques achieve a resource augmentation bound of 2 when the threads are scheduled with algorithms such as U-EDF, PD^2, LLREF, DP-Wrap, *etc*.

I. Introduction

These last years, we have witnessed a dramatic increase in the number of cores available in computational platforms. For instance, Intel designed a 80 cores platform [1] and Tilera sells processors with up to 100 identical cores [2]. To take advantage of this high degree of parallelism, a new coding paradigm has been introduced using programming languages and API such as OpenMP [3] or CilkPlus [4]. In this work we consider multi-threaded tasks, i.e., each task is a sequence of segments, each segment is a collection of threads and threads of a same segment can be scheduled *simultaneously* (see Fig. 1).

Multi-threaded parallel tasks have been studied in only few previous works. One may cite [5] in which the authors propose a sufficient (but pessimistic) schedulability test for such systems scheduled with global EDF. Interesting approaches were also proposed in [6] and [7] to optimize the utilization of the platform when tasks follow the *fork-join* model.

In this paper, we propose techniques optimizing the number of processors needed in the computational platform to ensure that all parallel tasks respect their deadlines. To reach this goal, we propose two different approaches: an optimization problem which must be solved offline; and an efficient (optimal) algorithm which can be executed online. Both techniques compute, for each thread, an intermediate (artificial) deadline. Consequently, the online scheduler can manage the execution of each thread as an independent *sequential* sporadic task with constrained deadline.

While being less efficient, the offline approach has the interest of being easily extensible to more general models of tasks and is therefore perfectly suited to systems where task sets are completely defined at design time. On the other hand, the online algorithm targets dynamic systems where all tasks and their characteristics are not known *a priori*. Furthermore, we prove the optimality of the online algorithm (amongst the algorithms adding intermediate deadlines) when the schedulability test is based on task densities, and show that both approaches achieves a resource augmentation bound[2] of 2 when the threads are scheduled using algorithms such as U-EDF [8], PD^2 [9], LLREF [10] or DP-Wrap [11].

II. Related Works

There are two main models of parallel tasks (i.e., tasks that may use several processors *simultaneously*): the *Gang* [12]–[15] and the *Thread* model [6], [7], [16]. With the Gang model, all parallel instances of a same task start and stop using the processors synchronously. For instance, if a task is composed of four parallel instances, then, it needs exactly four available processors to start its execution. On the other hand, with the Thread model, there is no such constraint. Hence, once a thread has been released, it can be executed on the processing platform independently of the execution of the other threads. Since our research studies multi-threaded tasks, the following state-of-the-art review is focused to this particular model of parallel tasks.

In [16], the authors consider that a task is a succession of instances composed of a set of (independent) threads. They define several kinds of real-time schedulers and their associated schedulability tests. In [6], Lakshmanan *et al.* propose a scheduling algorithm for the *fork-join* model. This model can be seen as a generalization of the previous work. Indeed, it considers each task instance as a sequence of segments, alternating between sequential and parallel phases.

[1]Supported by the Belgian National Science Foundation (F.N.R.S.) under a F.R.I.A. grant.

[2]An algorithm A has a resource augmentation bound of v if any task set schedulable on a platform with m processors of speed 1 is also schedulable with algorithm A on m processors of speed v.

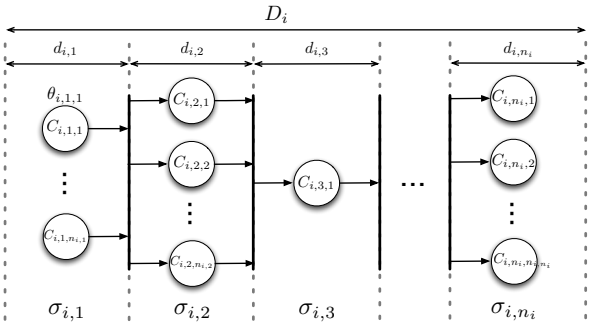


Fig. 1: Parallel task τ_i composed of n_i segments $\sigma_{i,1}$ to σ_{i,n_i}.

A sequential phase executes only one master thread which is split in multiple parallel threads during the parallel phases. In [6], all the parallel segments are assumed to have the same number of parallel threads. This alternation between parallel and sequential segments has been raised in [7] where each segment is composed of an arbitrary number of threads. In this last model, tasks have relative deadlines, but segments do not. In order to provide a schedulability test, the authors proposed to create an artificial deadline for each segment. Once those deadlines have been defined, the scheduling of the task system is equivalent to the scheduling of a set of sequential (sporadic) tasks, each thread being considered as an independent job [7][3].

The research presented in our paper strictly dominates the work in [7] in the sense that if the method in [7] needs m' processors to schedule a task set τ, then, we need $m \leq m'$ processors to schedule the same set τ. Actually, we prove that, compared to any solution imposing intermediate deadlines for the segment executions, our technique is optimal when the schedulability test is based on the task densities. During our experiments we even found examples where the method of [7] requires up to three times more processors than our approach.

III. Model

We assume a set $\tau \stackrel{\text{def}}{=} \{\tau_1, ..., \tau_n\}$ of n sporadic tasks with constrained deadlines. Each parallel task τ_i is a sequence of n_i segments. Each segment $\sigma_{i,j}$ is a set of $n_{i,j}$ threads and each thread $\theta_{i,j,k}$ has a worst case execution time (WCET) of $C_{i,j,k}$ (see Fig. 1).

A segment $\sigma_{i,j}$ can be characterized by the two quantities $C^{\min}_{i,j} \stackrel{\text{def}}{=} \max_k \{C_{i,j,k}\}$ and $C_{i,j} \stackrel{\text{def}}{=} \sum_{k=1}^{n_{i,j}} C_{i,j,k}$. On the one hand, $C^{\min}_{i,j}$ represents the minimum amount of time needed to complete the execution of all threads belonging to the segment $\sigma_{i,j}$ assuming that it can use $n_{i,j}$ processors for its execution. On the other hand, $C_{i,j}$ is the maximum amount of time needed to execute all threads in $\sigma_{i,j}$ on a single processor.

The total WCET C_i of any task τ_i is equal to the sum of the WCET of its constituting segments, i.e., $C_i \stackrel{\text{def}}{=} \sum_{j=1}^{n_i} C_{i,j}$. Hence, C_i is the maximum amount of time to complete the execution of the entire task τ_i on a single core.

[3]Note that adding artificial deadlines is not a new idea: it was already proposed in [17] to synchronize the execution of different parts of strictly periodic tasks running on different processors.

TABLE I: System notations

System characteristics	
τ	Set of tasks
n	Number of tasks in τ
m	Number of processors
Task characteristics	
τ_i	Task number i
n_i	Number of segments in τ_i
D_i	Relative deadline of τ_i
T_i	Minimal inter-arrival time
C_i	Worst-case execution time of τ_i ($C_i \stackrel{\text{def}}{=} \sum_{j=1}^{n_i} C_{i,j}$)
δ_i	Density of τ_i ($\delta_i \stackrel{\text{def}}{=} \frac{C_i}{D_i}$)
Segment characteristics	
$\sigma_{i,j}$	j^{th} segment of τ_i
$n_{i,j}$	Number of threads in $\sigma_{i,j}$
$C_{i,j}$	Worst-case execution time of $\sigma_{i,j}$ ($C_{i,j} \stackrel{\text{def}}{=} \sum_{k=1}^{n_{i,j}} C_{i,j,k}$)
$C^{\min}_{i,j}$	Minimum execution time of $\sigma_{i,j}$ ($C^{\min}_{i,j} \stackrel{\text{def}}{=} \max_k \{C_{i,j,k}\}$)
Thread characteristics	
$\theta_{i,j,k}$	k^{th} thread of $\sigma_{i,j}$
$C_{i,j,k}$	Worst case execution time of $\theta_{i,j,k}$

In this work, we assume that all threads of a same segment $\sigma_{i,j}$ are synchronous. That is, all threads of $\sigma_{i,j}$ are released simultaneously. Furthermore, all threads of a segment $\sigma_{i,j}$ must have completed their execution before starting to execute the next segment $\sigma_{i,j+1}$.

Each task τ_i is a sporadic task with constrained deadline, meaning that τ_i has a relative deadline D_i not larger than its minimal inter-arrival time T_i. This implies that τ_i has never more than one active instance at any time t. Therefore, we define an *active job* at time t (if any) as an instance of a task which has been released before or at time t and which has not reached its deadline yet.

The density of a task τ_i given by $\delta_i \stackrel{\text{def}}{=} \frac{C_i}{D_i}$, represents the average computing capacity needed for the execution of a job of τ_i between its arrival and its deadline occurring D_i time units later.

The notations used in this paper are summarized in Table I.

IV. Problem Statement

As previously explained, our goal is to transform the threads composing the parallel tasks in a set of sporadic *sequential* tasks with constrained deadlines. This approach, already described in [6], [7], implies that for each thread, an arrival time and a relative deadline must be defined.

We assume in our model that all threads of a same segment $\sigma_{i,j}$ are released simultaneously and the threads of $\sigma_{i,j+1}$ can only be released when the execution of threads of $\sigma_{i,j}$ is completed. We therefore assume that all threads in a segment $\sigma_{i,j}$ have the same deadline $d_{i,j}$ and threads in $\sigma_{i,j+1}$ are released when the deadline $d_{i,j}$ has been reached. Hence, the minimum inter-arrival time of any thread of τ_i is equal to the minimum inter-arrival time T_i of τ_i. Consequently, our problem consists in determining the relative deadlines $d_{i,j}$ of

every segment $\sigma_{i,j}$ so that the number of required processors to respect all deadlines is minimized.

Let the instantaneous total density $\delta(t)$ be the sum of the densities of all the *active jobs* at time t.

Some existing scheduling algorithms such as U-EDF, PD^2, DP-Wrap or LLREF can schedule any set of sequential *sporadic* tasks as long as the instantaneous total density $\delta(t)$ does not exceed the number m of processors constituting the processing platform [8]–[11]. This property is expressed through the following *sufficient feasibility test*:

Property 1. *A task set τ is feasible on a platform of m identical processors if, at any time t, we have $\delta(t) \leq m$.*

According to Property 1, if we want to minimize the number of processors needed to schedule τ, then we must minimize the maximum value reachable by $\delta(t)$. Hence, we define the density of $\sigma_{i,j}$ as $\delta_{i,j} \stackrel{\text{def}}{=} \frac{C_{i,j}}{d_{i,j}}$ and since two segments $\sigma_{i,j}$ and $\sigma_{i,\ell}$ ($j \neq \ell$) of the same task τ_i cannot be active simultaneously, we have to minimize the following expression:

$$\delta^{\max} = \sum_{\tau_i \in \tau} \max_j \{\delta_{i,j}\} = \sum_{\tau_i \in \tau} \max_j \left\{ \frac{C_{i,j}}{d_{i,j}} \right\} \qquad (1)$$

Indeed, since the parallel tasks are sporadic, we do not know which segment of each task will be active at each instant t. Therefore, in the worst case, each task has its segment with the largest density active at time t.

If, after the optimization process, we have $m \geq \delta^{\max}$ then the task set τ is schedulable on the processing platform using a scheduling algorithm such as those previously cited (i.e., U-EDF, PD^2, LLREF, DP-Wrap).

V. Offline Approach: An Optimization Problem

As stated in the previous section, optimizing the number of processors m implies to minimize $\delta^{\max}$. However, some constraints on the relative deadlines of each segment must be respected:

- The total execution time granted to all segments composing τ_i must be smaller than the relative deadline of τ_i. That is, for each task τ_i, it must hold that
$$\sum_{j=1}^{n_i} d_{i,j} \leq D_i$$
In practice, we will impose that $\sum_{j=1}^{n_i} d_{i,j} = D_i$ since the segment densities decrease if their deadlines increase (remember that $\delta_{i,j} \stackrel{\text{def}}{=} \frac{C_{i,j}}{d_{i,j}}$).
- The relative deadline of any segment $\sigma_{i,j}$ cannot be smaller than its minimum execution time $C_{i,j}^{\min}$. That is,
$$d_{i,j} \geq C_{i,j}^{\min} = \max_k \{C_{i,j,k}\}$$
Indeed, since a thread cannot be executed in parallel on two (or more) processors, the relative deadline of the associated segment cannot be smaller than its largest thread WCET.

TABLE II: Problem notations

Variables	
$d_{i,j}$	Relative deadline of segment $\sigma_{i,j}$
$\delta_{i,j}$	Density of segment $\sigma_{i,j}$ ($\delta_{i,j} \stackrel{\text{def}}{=} \frac{C_{i,j}}{d_{i,j}}$)
$\delta_i^{\max}$	Maximum instantaneous density of τ_i ($\delta_i^{\max} \stackrel{\text{def}}{=} \max_{\sigma_{i,j} \in \tau_i} \{\delta_{i,j}\}$)
$\delta^{\max}$	Maximum instantaneous density of τ ($\delta^{\max} \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau} \delta_i^{\max}$)
Properties	
$\widehat{\delta_{i,j}}$	Upper bound on $\delta_{i,j}$ ($\widehat{\delta_{i,j}} \stackrel{\text{def}}{=} \frac{C_{i,j}}{C_{i,j}^{\min}}$)
Optimisation problem	
S_i^ℓ	Set of segments of τ_i that remain to consider at the ℓ^{th} step of the algorithm
L_i^ℓ	Amount of time that has still to be dispatched among the segments of S_i^ℓ
C_i^ℓ	Worst-case execution time of S_i^ℓ ($C_i^\ell \stackrel{\text{def}}{=} \sum_{\sigma_{i,j} \in S_i^\ell} C_{i,j}$)
δ_i^ℓ	Average density of S_i^ℓ on L_i^ℓ time units ($\delta_i^\ell \stackrel{\text{def}}{=} \frac{C_i^\ell}{L_i^\ell}$)
$d_{i,j}^*$	Optimal value of $d_{i,j}$
$\delta_{i,j}^*$	Density of $\sigma_{i,j}$ in the optimal solution ($\delta_{i,j}^* \stackrel{\text{def}}{=} \frac{C_{i,j}}{d_{i,j}^*}$)
$\delta_i^{\ell\,\max}$	Maximum density of the segments in S_i^ℓ ($\delta_i^{\ell\,\max} \stackrel{\text{def}}{=} \max_{\sigma_{i,j} \in S_i^\ell} \{\delta_{i,j}\}$)
$\delta_i^{\ell\,\max *}$	Maximum density of the segments in S_i^ℓ in the optimal solution $\delta_i^{\ell\,\max *} \stackrel{\text{def}}{=} \max_{\sigma_{i,j} \in S} \left\{\delta_{i,j}^*\right\}$

Hence, the optimization problem can be written as

$$\begin{aligned} \text{Minimize:} \quad & \delta^{\max} = \sum_{\tau_i \in \tau} \max_j \left\{ \frac{C_{i,j}}{d_{i,j}} \right\} \\ \text{Subject to:} \quad & \forall \tau_i \in \tau \ , \ \sum_{j=1}^{n_i} d_{i,j} = D_i \\ & \forall \sigma_{i,j} \ , \ d_{i,j} \geq C_{i,j}^{\min} \end{aligned}$$

Nevertheless, since the constraints on the relative deadlines of any segment $\sigma_{i,j}$ of a task τ_i are independent of the constraints on any other segment $\sigma_{a,b}$ ($a \neq i$) of another task τ_a, we can subdivide this optimization problem in n subproblems consisting, for every task τ_i, in the minimization of $\max_j \left\{ \frac{C_{i,j}}{d_{i,j}} \right\}$. Indeed, the minimization of a sum with independent terms, is equivalent to the minimization of every term. We therefore get $\forall \tau_i \in \tau$,

$$\text{Minimize:} \quad \max_j \left\{ \frac{C_{i,j}}{d_{i,j}} \right\} = \max_j \{\delta_{i,j}\} \qquad (2)$$

$$\text{Subject to:} \quad \sum_{j=1}^{n_i} d_{i,j} = D_i \qquad (3)$$

$$\forall \sigma_{i,j} \in \tau_i \ , \ d_{i,j} \geq C_{i,j}^{\min} \qquad (4)$$

This new formulation has three different interests:

1) We now see that this optimization problem finds the optimal solution for any schedulability test that can be expressed in terms of system and/or task densities.

Indeed, in addition to minimize the maximum instantaneous density of the system, it also minimizes the maximum density of each task independently. Hence, we also find an optimal solution for the schedulability tests of many algorithms proposed in [18] for instance.

2) It allows to determine the relative deadlines of each task *independently*. Therefore, if the task set is modified, we do not have to recompute the properties of every task, but only for the new and altered tasks in τ.
3) It highly reduces the solution research space visited by the optimization algorithm. Notice that for each task τ_i, both the number of variables and the number of constraints in the optimization problem, increases linearly with the number of segments n_i in τ_i.

This optimization problem can be linearized as shown in [19], and can therefore be solved with linear programming techniques such as those presented in [20].

Notice that by Equations 3 and 4, a solution exists if and only if $\sum_{j=1}^{n_i} C_{i,j}^{\min} \leq D_i$. This condition is generalized in the following lemma:

Lemma 1. *Let S be a set of segments. There exists a solution to the problem of dispatching L time units between segments in S, under the constraints $\sum_{\sigma_{i,j} \in S} d_{i,j} \leq L$ and $\forall \sigma_{i,j} \in S$, $d_{i,j} \geq C_{i,j}^{\min}$, if and only if*

$$\sum_{\sigma_{i,j} \in S} C_{i,j}^{\min} \leq L$$

VI. Online Approach: An Efficient Algorithm

We just showed that, if the condition expressed by Lemma 1 is respected, a linear programming technique can be used to optimally solve the problem of finding intermediate deadlines for a parallel task τ_i. Usually, a scheduler cannot afford to solve a linear optimization problem online. However, in this specific case, the problem studied allows to provide a solution with a low time complexity (see Theorem 2 for more details). Due to the low complexity of this algorithm, segment deadlines can now be determined online. Hence, this algorithm can be used in dynamic systems where tasks may arrive or leave the application at any moment, and where the number of threads in each segment is not always defined at design time. Therefore, when a new task τ_i releases a job in the system, we can compute the segment intermediate deadlines minimizing the number of processors needed to the execution of τ_i. Furthermore, we can verify at any new task arrival if the system is overloaded. In the case of an affirmative answer, the operating system can take appropriate decisions such as stopping some tasks, re-dispatching the workload between different clusters, switching a processor on, *etc.*

In this section, we explain how we can derive from the optimization problem presented in the previous section, an algorithm that optimally determines the segment intermediate deadlines. This algorithm is *greedy*. That is, it iteratively builds the optimal solution by fixing one intermediate deadline at each step. Whenever an intermediate deadline is determined, it will never be updated. By smartly choosing the order in which segments are considered, we can guarantee that the optimality of the global solution will not be affected by the local decision taken for a particular segment.

Our algorithm must therefore determine the optimal intermediate deadlines $d_{i,j}^*$ for the set of segments $\sigma_{i,j} \in \tau_i$ (i.e., the intermediate deadlines minimizing the maximal segment density), while respecting the constraints (3) and (4) described in the previous section.

Similarly to the definition of $d_{i,j}^*$, the optimal density $\delta_{i,j}^*$ of a segment $\sigma_{i,j}$ is defined as $\dfrac{C_{i,j}}{d_{i,j}^*}$.

In the following, we denote by S_i^ℓ the set of segments we still need to consider at the ℓ^{th} step of the greedy algorithm (i.e., the segments $\sigma_{i,j}$ that do not have an associated deadline yet). Hence, it initially holds that $S_i^1 \stackrel{\text{def}}{=} \{\sigma_{i,j} \in \tau_i\}$. Similarly, L_i^ℓ denotes the amount of time that may be distributed among the segments in S_i^ℓ, assuming that $L_i^1 \stackrel{\text{def}}{=} D_i$. Therefore, at the ℓ^{th} iteration, there is a segment $\sigma_{i,j}$ such that $S_i^{\ell+1} = S_i^\ell \backslash \sigma_{i,j}$, and $L_i^{\ell+1} = L_i^\ell - d_{i,j}^*$.

We define the WCET of the remaining segments at iteration ℓ as

$$C_i^\ell \stackrel{\text{def}}{=} \sum_{\sigma_{i,j} \in S_i^\ell} C_{i,j} \tag{5}$$

and the average density at iteration ℓ as $\delta_i^\ell \stackrel{\text{def}}{=} \dfrac{C_i^\ell}{L_i^\ell}$.

Similarly, the maximum instantaneous density of the set of segments S_i^ℓ is defined as

$$\delta_i^{\ell\,\max} \stackrel{\text{def}}{=} \max_{\sigma_{i,j} \in S_i^\ell} \{\delta_{i,j}\}$$

Notice that $C_i^1 = C_i$, $\delta_i^1 = \delta_i$ and $\delta_i^{1\,\max} = \delta_i^{\max}$, where $\delta_i^{\max} \stackrel{\text{def}}{=} \max_{\sigma_{i,j} \in \tau_i} \{\delta_{i,j}\}$. Table II summarizes the notations used in the problem presented in this section.

The algorithm presented in this section is based on a simple property that can be expressed as follows:

Property 2. *Let L_i^ℓ be the time that must be distributed among the segments belonging to S_i^ℓ. The maximum instantaneous density $\delta_i^{\ell\,\max}$ of any solution (i.e., even suboptimal) cannot be smaller than the average density δ_i^ℓ. (i.e., $\delta_i^{\ell\,\max} \geq \delta_i^\ell$, $\forall \ell$).*

This property is directly derived from the fact that the maximum value of a set is never lower than the average value of the same set. Another interesting property is presented through Lemma 2 and its corollary.

Lemma 2. *If the time L_i^ℓ that must be distributed among the segments belonging to a set S_i^ℓ increases, then, the optimal maximum instantaneous density $\delta_i^{\ell\,\max *}$ of S_i^ℓ can only decrease.*

Proof: If the time L_i^ℓ increases by Δ time units, we can distribute Δ among the segments with the largest densities $\delta_{i,j}^*$. The densities $\delta_{i,j}^*$ of these segments are then reduced which in turn implies that $\delta_i^{\ell\,\max *} \stackrel{\text{def}}{=} \max_{\sigma_{i,j} \in S_i^\ell} \left\{\delta_{i,j}^*\right\}$ decreases. ■

Corollary 1. *If the time L_i^ℓ that must be distributed among the segments belonging to a set S_i^ℓ decreases, then, the optimal maximum instantaneous density $\delta_i^{\ell\max *}$ of S_i^ℓ can only increase.*

Thanks to these properties, we can now construct our algorithm.

According to Equation 4, the relative deadline $d_{i,j}$ of a segment $\sigma_{i,j}$ cannot be smaller than $C_{i,j}^{\min}$. Since $C_{i,j}^{\min}$ is a lower bound on the relative deadline of $\sigma_{i,j}$, we define the density upper bound of $\sigma_{i,j}$ as

$$\widehat{\delta_{i,j}} \stackrel{\text{def}}{=} \frac{C_{i,j}}{C_{i,j}^{\min}} \tag{6}$$

Two situations can be encountered at any step ℓ of our algorithm. Either the density upper bound $\widehat{\delta_{i,j}}$ of every segment $\sigma_{i,j} \in S_i^\ell$ is not smaller than the average density $\delta_i^\ell \stackrel{\text{def}}{=} \frac{C_i^\ell}{L_i^\ell}$, or there is a segment $\sigma_{i,j} \in S_i^\ell$ such that $\widehat{\delta_{i,j}} < \delta_i^\ell$. These two cases must be considered independently.

Lemma 3. *If, for every segment $\sigma_{i,j} \in S_i^\ell$, we have $\widehat{\delta_{i,j}} \geq \delta_i^\ell$, then, in order to minimize the maximum instantaneous density $\delta_i^{\ell\max}$ of S_i^ℓ, we must impose $\delta_{i,j} = \delta_i^\ell$ to every segment $\sigma_{i,j} \in S_i^\ell$ (i.e., $\delta_{i,j}^* = \delta_i^\ell$, $\forall \sigma_{i,j} \in S_i^\ell$). Furthermore, we get $\delta_i^{\ell\max *} = \delta_i^\ell$.*

Proof: Let us assume that every segment $\sigma_{i,j}$ in S_i^ℓ has a density $\delta_{i,j}$ equal to δ_i^ℓ. Since $\delta_i^{\ell\max} = \max_{\sigma_{i,j} \in S_i^\ell}\{\delta_{i,j}\}$, we get that $\delta_i^{\ell\max} = \delta_i^\ell$.

Furthermore, by Property 2, the maximum instantaneous density $\delta_i^{\ell\max}$ cannot be smaller than δ_i^ℓ. Therefore, $\delta_i^{\ell\max}$ is minimum when $\delta_{i,j} = \delta_i^\ell$ for every segment $\sigma_{i,j}$ in S_i^ℓ. ■

Lemma 4. *Let S_i^ℓ be a set of segments and δ_i^ℓ be its average density on L_i^ℓ. If there is a segment $\sigma_{i,j} \in S_i^\ell$ such that $\widehat{\delta_{i,j}} < \delta_i^\ell$, then, in order to minimize the maximum density $\delta_i^{\ell\max}$ of S_i^ℓ on L_i^ℓ, we must impose $\delta_{i,j} = \widehat{\delta_{i,j}}$ (i.e., $\delta_{i,j}^* = \widehat{\delta_{i,j}}$).*

Proof: Let C_i^ℓ be the total worst-case execution time of S_i^ℓ. Let us assume that $\sigma_{i,j}$ has a worst-case execution time $C_{i,j}$. We must determine the relative-deadline $d_{i,j}$ for $\sigma_{i,j}$ so that the maximum density of S_i^ℓ is minimized on L_i^ℓ. Let $S_i^{\ell+1}$ denote the set $S_i^\ell \setminus \sigma_{i,j}$. We have $C_i^{\ell+1} \stackrel{\text{def}}{=} C_i^\ell - C_{i,j}$ and the resolution of the optimization problem implies the resolution of the subproblem of dispatching $L_i^{\ell+1}$ time units on the set of segments $S_i^{\ell+1}$.

The density of $\sigma_{i,j}$ can be expressed as $\delta_{i,j} \stackrel{\text{def}}{=} \frac{C_{i,j}}{d_{i,j}}$. Therefore, it holds that $\delta_{i,j} \times d_{i,j} = C_{i,j}$. Since $C_{i,j}$ is constant, if $\delta_{i,j}$ decreases then $d_{i,j}$ increases. Hence, the time $L_i^\ell - d_{i,j}$ that must be distributed among the segments belonging to $S_i^{\ell+1}$ decreases.

Consequently, by Corollary 1, $\delta_i^{\ell+1\max *}$, the maximum density of $S_i^{\ell+1}$ can only increase.

Furthermore, by Property 2, $\delta_i^{\ell\max}$ cannot be smaller than δ_i^ℓ. Since, by assumption, the maximum density $\widehat{\delta_{i,j}}$ reachable by $\sigma_{i,j}$ is smaller than δ_i^ℓ, then, $\delta_i^{\ell\max} = \delta_i^{\ell+1\max *}$.

Algorithm 1: Deadlines assignment algorithm.

Input: τ_i;

1 $S_i^1 := \{\sigma_{i,j} \in \tau_i\}$; // Ordered by $\widehat{\delta_{i,j}}$
2 $C_i^1 := C_i$;
3 $L_i^1 := D_i$;
4 **for** $\ell : 1 \rightarrow n_i$ **do**
5 $\quad \delta_i^\ell := \frac{C_i^\ell}{L_i^\ell}$;
6 $\quad \sigma_{i,j}$:= first segment of S_i^ℓ (i.e., with the minimal $\widehat{\delta_{i,j}}$) ;
7 $\quad$ **if** $\widehat{\delta_{i,j}} < \delta_i^\ell$ **then**
8 $\quad\quad d_{i,j}^* := C_{i,j}^{\min}$; // Rule 2
9 $\quad$ **else**
10 $\quad\quad \forall \sigma_{i,j} \in S_i^\ell : d_{i,j}^* := \frac{C_{i,j}}{\delta_i^\ell}$; // Rule 1
11 $\quad\quad$ break;
12 $\quad$ **end**
13 $\quad S_i^{\ell+1} := S_i^\ell \setminus \sigma_{i,j}$;
14 $\quad C_i^{\ell+1} := C_i^\ell - C_{i,j}$;
15 $\quad L_i^{\ell+1} := L_i^\ell - d_{i,j}^*$;
16 **end**

Hence, $\delta_{i,j}$ must be maximized to minimize $\delta_i^{\ell\max}$. Therefore, we have to impose $\delta_{i,j} = \widehat{\delta_{i,j}}$ which states the lemma. ■

Since $\delta_{i,j} = \frac{C_{i,j}}{d_{i,j}}$, we derive from Lemmas 3 and 4, the two following rules to minimize the maximum instantaneous density $\delta_i^{\ell\max}$ of any problem of dispatching L_i^ℓ time units among the set of segments S_i^ℓ.

Rule 1. *If, for every segment $\sigma_{i,j} \in S_i^\ell$ we have $\widehat{\delta_{i,j}} \geq \delta_i^\ell$ then $d_{i,j}^* = \frac{C_{i,j}}{\delta_i^\ell}$ for all segments in S_i^ℓ.*

Rule 2. *Let $\sigma_{i,j}$ be a segment in S_i^ℓ. If $\widehat{\delta_{i,j}} < \delta_i^\ell$ then we have $d_{i,j}^* = C_{i,j}^{\min}$.*

These two rules lead to Algorithm 1 which is used to determine the segment relative deadlines of a task τ_i.

First, the algorithm computes the maximum reachable density $\widehat{\delta_{i,j}}$ of every segment $\sigma_{i,j}$ belonging to τ_i and sorts the segments in an increasing $\widehat{\delta_{i,j}}$ order (Line 1). Then, at the step ℓ of Algorithm 1, it selects the segment $\sigma_{i,j}$ with the smallest density upper bound among the remaining segments, and compares $\widehat{\delta_{i,j}}$ with the average density δ_i^ℓ of S_i^ℓ (Line 6). If $\widehat{\delta_{i,j}} < \delta_i^\ell$ then we apply Rule 2 (Line 8). Otherwise, we can use Rule 1 to compute the segment deadline (Line 10). Finally, whenever the optimal deadline $d_{i,j}^*$ of a segment $\sigma_{i,j}$ has been determined, $\sigma_{i,j}$ is removed from the problem, and the values $S_i^{\ell+1}$, $C_i^{\ell+1}$ and $L_i^{\ell+1}$ are computed accordingly (Lines 13 to 15).

Theorem 1. *If, $\forall i, \sum_j C_{i,j}^{\min} \leq D_i$, Algorithm 1 provides an optimal solution to the problem of dispatching D_i time units amongst the n_i segments of τ_i.*

Proof: To prove the optimality of the solution proposed by Algorithm 1, we must prove that $\delta_i^{\max}$ is minimum and all constraints expressed by Equations 3 and 4 are respected.

1) Since the segments are ordered in an increasing $\widehat{\delta_{i,j}}$

order, we first compute the relative deadlines of the segments with the smallest $\widehat{\delta_{i,j}}$ values. Therefore, if $\widehat{\delta_{i,j}} \geq \delta_i^\ell$, then all the remaining segments in S_i^ℓ have $\widehat{\delta_{i,j}} \geq \delta_i^\ell$. Hence, we can apply Lemma 3 to determine the optimal deadline of $\sigma_{i,j}$ while minimizing $\delta_i^{\ell\,\max}$. It is exactly what is done at line 10 of Algorithm 1. On the other hand, if $\widehat{\delta_{i,j}} < \delta_i^\ell$, then, according to Lemma 4 we must maximize the density of $\sigma_{i,j}$ in order to minimize $\delta_i^{\max}$. Consequently, Line 8 in Algorithm 1 applies Rule 2 which is the direct consequence of Lemma 4.
Hence, in both cases, Algorithm 1 applies the correct rule to minimize $\delta_i^{\max}$.

2) Proving that $\sum_j d_{i,j}^* = D_i$ is straightforward if the algorithms enters the "else" bloc (Line 10). Indeed, by applying Rule 2, the algorithm dispatches the remaining L_i^ℓ times units between all the remaining segments in S_i^ℓ. Then, $\sum_{\sigma_{i,j} \in S_i^\ell} d_{i,j}^* = L_i^\ell$. As, by construction (see Line 15), $L_i^\ell = D_i - \sum_{\sigma_{i,j} \notin S_i^\ell} d_{i,j}^*$, we have that $\sum_{\sigma_{i,j} \in \tau_i} d_{i,j}^* = D_i$.
We still need to prove that Algorithm 1 eventually enters the "else" bloc. By contradiction, let us assume that we always have $\widehat{\delta_{i,j}} < \delta_i^\ell$. It means that for the very last segment in S_i^ℓ (say, $\sigma_{i,a}$; then $C_i^\ell = C_{i,a}$), we have $\widehat{\delta_{i,a}} = \frac{C_{i,a}}{C_{i,a}^{\min}} < \delta_i^\ell = \frac{C_i^\ell}{L_i^\ell} = \frac{C_{i,a}}{L_i^\ell}$. Then, $C_{i,a}^{\min} > L_i^\ell$. As, by Lines 8 and 15 (and we never enter the "else"), $L_i^\ell = D_i - \sum_{j \neq a} C_{i,j}^{\min}$, we have that $\sum_j C_{i,j}^{\min} > D_i$, which contradicts with the hypothesis of the theorem.

3) For every segment $\sigma_{i,j}$ belonging to τ_i, Algorithm 1 determines $d_{i,j}^*$ according to Rules 1 and 2. These rules are based on Lemmas 3 and 4, respectively. Since Lemmas 3 and 4 impose that $\delta_{i,j} \leq \widehat{\delta_{i,j}}$, it yields $d_{i,j}^* \geq C_{i,j}^{\min}$ from Equation 6. Hence, the constraint formulated by Equation 4 is respected.

Consequently, Algorithm 1 optimally resolves the optimization problem described in Section V. ∎

Theorem 2. *The computational complexity to determine the optimal segment deadlines for a task τ_i with Algorithm 1 is $O(n_i \times \log n_i)$.*

Proof: Algorithm 1 starts by sorting the n_i segments belonging to τ_i in an increasing $\widehat{\delta_{i,j}}$ order (Line 1). This manipulation has a computational complexity of $O(n_i \times \log n_i)$. Then, the *for* loop executed at Lines 6 to 15, iterates at most n_i times (once for each segment of τ_i). Since the computation time of $d_{i,j}^*$ is constant, the computational complexity of the loop is $O(n_i)$. Therefore, the total computational complexity of Algorithm 1 is dominated by the sorting algorithm, and is then $O(n_i \times \log n_i)$. ∎

VII. Resource Augmentation Bound

Let τ^* denote the task set τ when intermediate deadlines have been allocated to segments using one of the two techniques proposed in Sections V and VI. We first prove in Lemma 5 that every segment $\sigma_{i,j}$ of any task in τ^* has a density $\delta_{i,j} < \frac{C_i}{D_i - \sum_{k=1}^{n_i} C_{i,k}^{\min}}$. This property is then used to prove that both the offline and the online technique presented in Sections V and VI, achieve a resource augmentation bound of 2 when threads in τ^* are scheduled using algorithms such as U-EDF, PD^2, LLREF or DP-Wrap.

Lemma 5. *Every segment $\sigma_{i,j}$ of every task in τ^* has a density*

$$\delta_{i,j} < \frac{C_i}{D_i - \sum_{k=1}^{n_i} C_{i,k}^{\min}}$$

Proof: According to Algorithm 1, we have $\delta_i^\ell = \frac{C_i^\ell}{L_i^\ell}$. Furthermore, Algorithm 1 implies that $L_i^\ell = D_i - \sum_{\sigma_{i,k} \notin S_i^\ell} C_{i,k}^{\min}$ (as, as soon as the condition of line 7 is not valid anymore, the loop ends) and using Expression 5, we get

$$\delta_i^\ell = \frac{\sum_{\sigma_{i,q} \in S_i^\ell} C_{i,q}}{D_i - \sum_{\sigma_{i,k} \notin S_i^\ell} C_{i,k}^{\min}}$$

Furthermore, since $\sum_{\sigma_{i,q} \in S_i^\ell} C_{i,q} \leq \sum_{\sigma_{i,q} \in \tau_i} C_{i,q} = C_i$ and because, $\sum_{\sigma_{i,k} \notin S_i^\ell} C_{i,k}^{\min} < \sum_{\sigma_{i,k} \in \tau_i} C_{i,k}^{\min}$, it holds that

$$\delta_i^\ell < \frac{C_i}{D_i - \sum_{k=1}^{n_i} C_{i,k}^{\min}}$$

Because at every step ℓ, Algorithm 1 imposes:

- $\delta_{i,j} = \widehat{\delta_{i,j}}$ if $\widehat{\delta_{i,j}} < \delta_i^\ell$
- $\delta_{i,j} = \delta_i^\ell$ otherwise.

We have in both cases: $\delta_{i,j} \leq \delta_i^\ell < \frac{C_i}{D_i - \sum_{k=1}^{n_i} C_{i,k}^{\min}}$.

Hence, the lemma is proven for Algorithm 1.

Since the resolution of the optimization problem presented in Section V, minimizes the maximum density reachable by every segment in τ_i, and because Algorithm 1 get $\delta_{i,j} < \frac{C_i}{D_i - \sum_{k=1}^{n_i} C_{i,k}^{\min}}$ for every segment $\sigma_{i,j}$, it must hold that $\delta_{i,j} < \frac{C_i}{D_i - \sum_{k=1}^{n_i} C_{i,k}^{\min}}$ with the optimization algorithm.

This states the lemma. ∎

Theorem 3. *If a periodic multi-threaded task set τ is schedulable on m unit-speed identical processors using any optimal algorithm, then the task set τ^* is schedulable on m identical processors of speed 2, using any algorithm respecting Property 1.*

Proof: According to Lemma 5 any segment of any task in τ^* has a density $\delta_{i,j} < \frac{C_i}{D_i - \sum_{k=1}^{n_i} C_{i,k}^{\min}}$ on a unit-speed processor. Hence, the maximum instantaneous density achievable in τ^* is given by

$$\delta^{\max} = \sum_{\tau_i \in \tau} \max_j \{\delta_{i,j}\} < \sum_{\tau_i \in \tau} \left\{ \frac{C_i}{D_i - \sum_{k=1}^{n_i} C_{i,k}^{\min}} \right\}$$

Furthermore, if the processors in the platform are v times faster, then all the worst-case execution times are divided by v. Hence, the maximum instantaneous density becomes

$$\delta^{\max,v} < \sum_{\tau_i \in \tau} \left\{ \frac{\frac{C_i}{v}}{D_i - \frac{\sum_{k=1}^{n_i} C_{i,k}^{\min}}{v}} \right\}$$

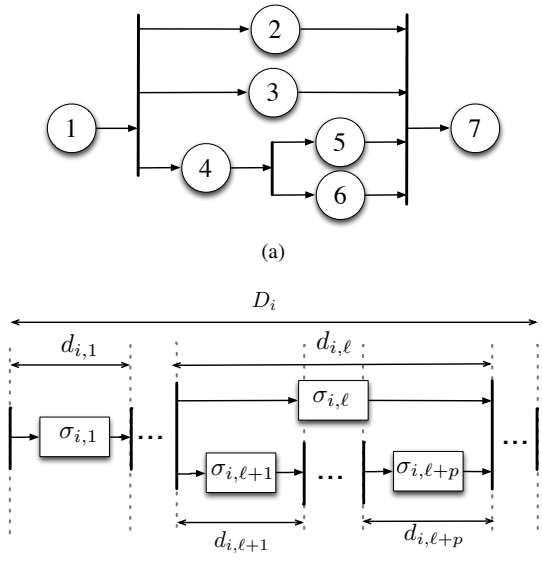


Fig. 2: (a) Generalized task model where threads can run in parallel with several successive segments. (b) General representation of the same problem where a segment $\sigma_{i,\ell}$ is simultaneously active with segments $\sigma_{i,\ell+1}$ to $\sigma_{i,\ell+p}$.

Applying Lemma 1 to this expression (with L, the available time, being replaced by D_i), implies that

$$\delta^{\max,v} < \sum_{\tau_i \in \tau} \left\{ \frac{\frac{C_i}{v}}{D_i - \frac{D_i}{v}} \right\} = \frac{1}{v-1} \sum_{\tau_i \in \tau} \frac{C_i}{D_i}$$

A periodic task set is schedulable by any optimal algorithm on m unit-speed processors if $\sum_{\tau_i \in \tau} \frac{C_i}{D_i} \leq m$. Hence,

$$\delta^{\max,v} < \frac{1}{v-1} m$$

Furthermore, any algorithm respecting Property 1 imposes that $\delta^{\max,v} \leq m$. This condition is therefore respected if

$$\frac{1}{v-1} m \leq m$$

thereby, by imposing a processor speed v of 2, we guarantee no task will ever miss any deadline. ■

Note that the same approach can be used to derive an augmentation bound for other algorithms such as global EDF, as long as the schedulability test is based on densities. Hence, it can be proven that global EDF has an augmentation bound of 2.62 using the schedulability test (Test 10) proposed in [18].

VIII. Task Model Generalization

In the model considered so far, we assumed that threads of a same segment can be executed in parallel, but segments of a same task cannot. Although this task representation is realistic for many applications, it is also simplistic. For instance, let us consider the task τ_i represented on Figure 2(a). Its execution starts with a single thread (numbered 1). Then, the first thread forks in three threads indexed 2, 3 and 4. However, contrarily to threads 2 and 3, at a certain point of its execution the thread 4 can be parallelized in two other threads numbered 5 and 6. Finally, all active threads must complete their execution before executing the last thread 7. In this example, we have one segment composed of threads 2 and 3 which is running concurrently with two other segments containing thread 4 and threads 5 and 6, respectively.

Therefore, in this section we generalize the task model, allowing several segments of the same task to run in parallel. Then, we propose some variations of the offline approach to minimize the number of processors needed to schedule such parallel tasks.

A. One Parallel Segment

Let us first consider a task τ_i (pictured on Fig. 2(b)) containing one (and only one) segment $\sigma_{i,\ell}$ running in parallel with p consecutive segments $\sigma_{i,\ell+1}$ to $\sigma_{i,\ell+p}$. We remind the reader that each segment is composed of several threads (as illustrated in the example of Fig. 2(a)).

From the segment point of view, we say that the segments $\sigma_{i,\ell}$ to $\sigma_{i,\ell+p}$ belong to a parallel phase. This parallel phase is composed of two branches: a first branch containing only one segment $\sigma_{i,\ell}$, and a second branch constituted of p successive segments $\sigma_{i,\ell+1}$ to $\sigma_{i,\ell+p}$. For the sake of clarity, we assume that there is only one parallel phase. However, the solution presented in this section is straightforwardly applicable to tasks containing several parallel phases.

The technique computing the segment intermediate deadlines is similar to the approach proposed in Section V. Since the instantaneous density of a task at time t is equal to the sum of the densities of its segments running at time t, we define an optimization problem considering that the densities of segments $\sigma_{i,\ell+1}$ to $\sigma_{i,\ell+p}$, are increased by the density $\delta_{i,\ell}$ of the segment $\sigma_{i,\ell}$. Furthermore, as we can see on Fig. 2(b), the deadline $d_{i,\ell}$ of the segment $\sigma_{i,\ell}$ must be equal the sum of the deadlines allocated to segments $\sigma_{i,\ell+1}$ to $\sigma_{i,\ell+p}$.

Therefore, the optimization problem can be expressed as follows: $\forall \tau_i \in \tau$,

$$\text{Minimize:} \quad \max_{j \neq \ell} \left\{ \delta'_{i,j} \right\} \tag{7}$$

$$\text{Subject to:} \quad \sum_{j \neq \ell} d_{i,j} = D_i \tag{8}$$

$$\forall \sigma_{i,j} \in \tau_i \ , \ d_{i,j} \geq C^{\min}_{i,j} \tag{9}$$

$$\sum_{q=1}^{p} d_{i,\ell+q} = d_{i,\ell} \tag{10}$$

$$\text{where } \delta'_{i,j} \stackrel{\text{def}}{=} \begin{cases} \delta_{i,j} + \delta_{i,\ell} & \text{if } j \in [\ell+1, \ldots, \ell+p] \\ \delta_{i,j} & \text{otherwise} \end{cases}$$

This new optimization problem is *non-linear*. However, the constraints and objective functions are convex. Hence, a non-linear programming technique such as [21] can be used to efficiently solve this new problem.

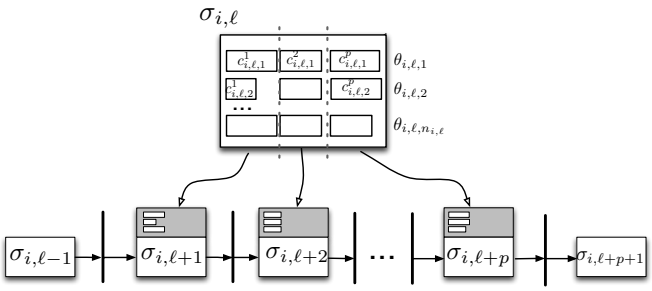


Fig. 3: Work of segment $\sigma_{i,\ell}$ being dispatched amongst the p successive segments $\sigma_{i,\ell+1}$ to $\sigma_{i,\ell+p}$. Each thread of $\sigma_{i,\ell}$ is split in p parts.

B. Splitting the Parallel Segment

The approach proposed above is rather pessimistic. Indeed, it assumes that the segment $\sigma_{i,\ell}$ is uniformly executed in parallel with the segments $\sigma_{i,\ell+1}$ to $\sigma_{i,\ell+p}$, meaning that it increases the densities of all segments $\sigma_{i,\ell+1}$ to $\sigma_{i,\ell+p}$ by the same value $\delta_{i,\ell}$. A better technique consists in dispatching $\sigma_{i,\ell}$'s workload so that more work is executed in parallel with segments with low densities and less work on densest segments. Hence, every thread $\theta_{i,\ell,k}$ belonging to $\sigma_{i,\ell}$ is divided in p parts associated with the p segments $\sigma_{i,\ell+1}$ to $\sigma_{i,\ell+p}$ (see Fig. 3). Each part has a worst-case execution time $c^q_{i,\ell,k}$ representing the portion of $C_{i,\ell,k}$ executed in parallel with segment $\sigma_{i,\ell+q}$ ($q \in [1,\ldots,p]$). This quantity is a new variable in our optimization problem. We therefore re-define the optimization problem as follows: $\forall \tau_i \in \tau$,

$$\text{Minimize:} \quad \max_{j \neq \ell} \left\{ \delta'_{i,j} \right\} \tag{11}$$

$$\text{Subject to:} \quad \sum_{j \neq \ell} d_{i,j} = D_i \tag{12}$$

$$\forall \sigma_{i,j} \in \tau_i \ , \ d_{i,j} \geq C^{\min}_{i,j} \tag{13}$$

$$\forall \theta_{i,\ell,k} \in \sigma_{i,\ell} \ , \ \sum_{q=1}^{p} c^q_{i,\ell,k} = C_{i,\ell,k} \tag{14}$$

$$\forall j \in [\ell+1,\ldots,\ell+p], \forall k, \ d_{i,j} \geq c^{j-\ell}_{i,\ell,k} \tag{15}$$

$$\text{where } \delta'_{i,j} \stackrel{\text{def}}{=} \begin{cases} \dfrac{C_{i,j} + \sum_{k=1}^{n_{i,\ell}} c^{j-\ell}_{i,\ell,k}}{d_{i,j}} & \text{if } j \in [\ell+1,\ldots,\ell+p] \\ \delta_{i,j} & \text{otherwise.} \end{cases}$$

Constraint (14) ensures that the entire workload of $\sigma_{i,\ell}$ is dispatched amongst the segments $\sigma_{i,\ell+1}$ to $\sigma_{i,\ell+p}$. On the other hand, constraint (15) makes sure that the work allocated in each segment is not greater than its deadline. As in the previous case, the problem is non linear, but is still convex, and can therefore be solved efficiently [21].

C. General Problem

The model defined in the previous section can still be further generalized. Instead of considering one branch containing only one segment and another branch containing several segments, we may consider that the parallel phase is composed of several branches with, respectively, $p_1, p_2, p_3, \ldots$ segments.

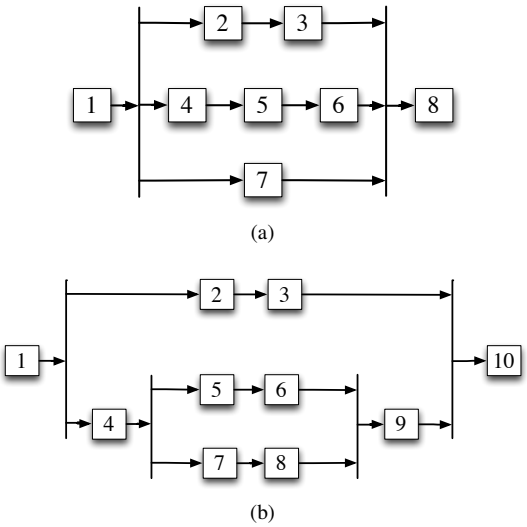


Fig. 4: Two examples of more general models. (a) A task containing a parallel phase composed of several branches of various size. (b) A task described by a DAG.

Figure 4(a) shows an example where $p_1 = 2$, $p_2 = 3$ and $p_3 = 1$. In this case, finding the maximal density of a task requires to consider all combinations of segments that may run concurrently. For instance, in Figure 4(a), segment number 1 will never run in parallel with any other segment, but segments $\{2,4,7\}$ may run in parallel, as well as $\{2,5,7\}, \{2,6,7\}, \{3,4,7\}, \{3,5,7\}$ or $\{3,6,7\}$. Then, in order to minimize the maximum density of task τ_i, we now have to minimize the following expression in place of (7):

$$\max\{\delta_{i,1}, \delta_{i,2} + \delta_{i,4} + \delta_{i,7}, \delta_{i,2} + \delta_{i,5} + \delta_{i,7}, \ldots, \delta_{i,8}\}$$

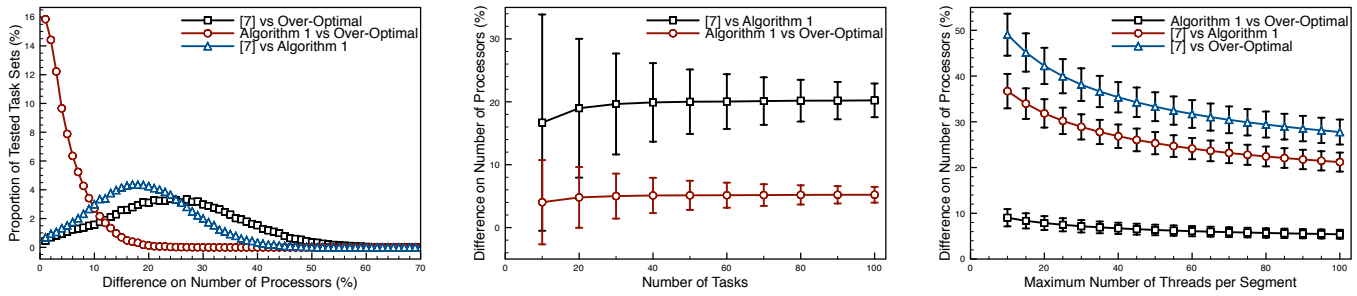
The number of terms in the objective function is then $p_1 \times p_2 \times p_3 \times \ldots$, plus the number of segments in the sequential phases. Furthermore, as we did with constraints (8) and (10), we need to make sure that

$$\begin{aligned} D_i &= d_{i,1} + d_{i,2} + d_{i,3} + d_{i,8} \\ &= d_{i,1} + d_{i,4} + d_{i,5} + d_{i,6} + d_{i,8} \\ &= d_{i,1} + d_{i,7} + d_{i,8} \end{aligned}$$

There is then as many such constraints as there are branches in task τ_i. Of course, we still need to make sure that $\forall \sigma_{i,j}, d_{i,j} \geq C^{\min}_{i,j}$ (constraint (9)). The method described in Section VIII-A can then be easily extended to this new problem.

More generally, we may consider any Directed Acyclic Graph (DAG) to represent the dependencies between segments, as in Figure 4(b). Our objective function needs now to consider any set of segments that could possibly run in parallel. Note that such segment combination is called a maximal anti-chain in the graph community[4]. In our example, we have $\{1\}$, $\{2,4\}$, $\{2,5,7\}$, $\{2,5,8\}$, $\{2,6,7\}$, $\ldots$, $\{3,9\}$, $\{10\}$.

[4]Algorithms enumerating all the maximal anti-chains in a graph can be found in [22]



(a) (b) (c)

Fig. 5: Simulation results comparing the number of processors needed to schedule a task set τ, when using Algorithm 1, the algorithm proposed in [7] and the "over-optimal" schedulability bound ($m = \sum_{\tau_i \in \tau} \delta_i$).

Therefore, the objective function that must be minimized is:

$$\max\{\delta_{i,1}, \delta_{i,2} + \delta_{i,4}, \delta_{i,2} + \delta_{i,5} + \delta_{i,7}, \ldots, \delta_{i,3} + \delta_{i,9}, \delta_{i,10}\}$$

In our example, there are 14 possible segment combinations. Note that, in some particular cases, the number of anti-chains could grow exponentially with the number of segments. Furthermore, the number of constraints depends on the structure of the DAG: for any possible "path" going through the graph (called a maximal chain in the literature), the sum of intermediate deadlines has to be equal to D_i. In our example, we have only 3 paths ($\{1, 2, 3, 10\}$, $\{1, 4, 5, 6, 9, 10\}$ or $\{1, 4, 7, 8, 9, 10\}$). Hence, for each path we get one more constraint. Note that the number of paths could also grow exponentially with the number of segments in some pathological cases.

IX. Simulation Results

We evaluated the performances of Algorithm 1 through three different experiments. Note that Algorithm 1 provides an optimal solution for the optimization problem presented in Section V. Hence, the simulation results discussed in this section also hold for the offline technique. Furthermore, the generalization of the offline technique presented in Section VIII cannot be compared with any other solution since such a solution did not yet exist. Usually, the offline approach finds a solution in a few seconds for the general task model. This is more than reasonable for an algorithm executed offline.

In the first experiment, we randomly generated $100,000$ task sets and computed the number of processors needed to ensure that all tasks respect their deadlines. We compared our method with the methodology proposed in [7]. We also compared both solutions with a lower bound on this number of processors, obtained by summing up the density δ_i of all tasks (i.e., $m = \sum_{\tau_i \in \tau} \delta_i$). We named this bound "over-optimal", as even an optimal algorithm cannot always reach this bound [6]. Nevertheless, it gives a good idea on the quality of the solution found with Algorithm 1.

In our tests, each task set is composed of 50 tasks. Each task has a number of segments n_i randomly chosen in a uniform distribution within $[1, 30]$. To be coherent with the model presented in [7], we assume that all threads in a segment $\sigma_{i,j}$ have an identical worst-case execution time $C^{\min}_{i,j}$ randomly chosen within $[1, 100]$[5]. The number of threads $n_{i,j}$ belonging to a segment $\sigma_{i,j}$ is randomly chosen within $[1, 50]$. The deadline D_i of the task τ_i belongs to the interval extending from $\sum_{\sigma_{i,j} \in \tau_i} C^{\min}_{i,j}$ to $\sum_{\sigma_{i,j} \in \tau_i} n_{i,j} \times C^{\min}_{i,j}$. Indeed, as stated in Lemma 1, the task set is not feasible if $D_i < \sum_{\sigma_{i,j} \in \tau_i} C^{\min}_{i,j}$. On the other hand, all threads in τ_i can be executed consecutively (without any kind of parallelism) and still respect the task deadline when $D_i > \sum_{\sigma_{i,j} \in \tau_i} n_{i,j} \times C^{\min}_{i,j}$. Hence the solution would have been trivial and identical for both Algorithm 1 and the method proposed in [7]. In all experiments, we assume that the tasks are scheduled with an optimal algorithm such as U-EDF, LLREF or PD^2.

The results are presented in Fig. 5(a). The curve *" [7] vs Algorithm 1"* in Fig. 5(a) for instance, represents the distribution of the task sets regarding the difference between the number of processors needed with Algorithm 1 and the solution provided by [7]. We can observe that our algorithm performs quite well against the "over-optimal" bound. Indeed, in average, it needs less than 5% more processors than the "over-optimal" bound (which may not be reachable). The median value is even under 4%. On the other hand, the algorithm proposed in [7] needs in average 25.5% processors more than the over-optimal bound. Furthermore, its distribution is widely spread leading to a standard deviation greater than 11.5%. Therefore, the algorithm in [7] logically needs almost 20% more processors than our methodology (in average). Even worst, we detected in some particular cases, a difference of 210% between the number of processors needed with our solution and the algorithm proposed in [7]. That is, the method presented in [7] may need three times more processors than ours for some specific task sets.

The second experiment gives the average value of the relative difference between the number of processors needed with the three techniques, when the number of tasks in τ varies within $[1, 100]$. Each point in Fig. 5(b) is the result of $10,000$ simulations. We can observe that the average value is

[5]Notice that the fact that all threads have the same worst-case execution time will not impact the quality of the simulation results. Indeed, the only parameters actually used in Algorithm 1 are $C_{i,j}$ and $C^{\min}_{i,j}$. The solution is therefore independent of the particular $C_{i,j,k}$ values.

small when there are few tasks in the system, but increases when the number of tasks grows. Inversely, the standard deviation is high for small task systems but decreases when the system grows. We note that the average difference between the number of processors needed with [7] and our solution seems to stabilize around 20% when the number of tasks is greater than 20. On the other hand, the average relative difference on m when we compare our algorithm with the over-optimal solution, never exceeds 6%. This means that even if we constrain the system by imposing deadlines to the task segments, the number of processors needed when using our methodology in conjunction with a scheduling algorithm such as U-EDF, LLREF or PD^2, remains close to the minimum number of processors which could ever be reached.

Finally, the third experiment shows the sensitivity of the solution produced by the algorithm proposed in [7], to the maximum number of threads $n_{i,j}$ composing each segment $\sigma_{i,j}$ (see Fig. 5(c)). Again, each point is the result of $10,000$ simulations. The variation of the average value of the relative difference between our solution and the algorithm in [7] is due to the fact that the methodology presented in [7] divides the segments in two groups: the light and the heavy segments. The distinction between these two kinds of segments is based on the number of threads $n_{i,j}$ composing each segment $\sigma_{i,j}$. When all segments are considered as being heavy, the solution proposed by [7] is optimal and identical to ours. On the other hand, when all segments are light, the solution proposed by [7] is almost systematically suboptimal. Since there are more and more task sets with only heavy segments when the maximum value of $n_{i,j}$ increases, and since there are more and more task sets with only light segments when the maximum value of $n_{i,j}$ decreases, the trend of Fig. 5(c) can easily be explained.

X. Conclusion and Future Works

In this paper, we propose two new techniques determining the relative deadlines that must be applied to each segment belonging to a parallel task τ_i in order to optimize the number of processors needed to schedule the task set. The first approach consists in an optimization problem that can be solved offline. The second approach is a greedy algorithm which optimally solves the optimization problem with a complexity of $O(n_i \times \log(n_i))$. The advantage of the first technique is that it can be easily extended to very general and realistic models of tasks, where segments, instead of being sequential, are organized in a DAG. However, this generalization is at the expense of an increasing complexity of the optimization problem. The advantage of the second model lies in his very low complexity, allowing to execute this algorithm online. Hence, this second solution is particularly well suited for dynamic task systems where all task informations are not known at design time.

We proved that both techniques achieve a resource augmentation bound of 2 when threads are scheduled with algorithms such as U-EDF, PD^2, LLREF or DP-Wrap, in comparison with algorithms that do not impose intermediate deadlines. We also showed through simulations that our methodology to compute the segment deadlines may drastically improve the number of processors needed in the processing platform compared to previous works (up to three times less).

As future works, we aim at proposing an algorithm with a low run-time complexity, determining the optimal segment deadlines for the generalized parallel task model presented in this paper. Furthermore, we will study how both approaches could be extended to ensure the schedulability of task sets constituted of parallel tasks with dependencies. Finally, we would like to extend this approach to partitioned and semi-partitioned algorithms.

References

[1] Intel, "Teraflops research chip." [Online]. Available: http://techresearch.intel.com/articles/Tera-Scale/1449.htm

[2] Tilera, "Tile-gx 3000 series overview," 2011. [Online]. Available: http://www.tilera.com/sites/default/files/productbriefs/TILE-Gx 3000 Series Brief.pdf

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

[4] Intel, "Cilkplus." [Online]. Available: http://software.intel.com/en-us/articles/intel-cilk-plus

[5] M. Korsgaard and S. Hendseth, "Schedulability analysis of malleable tasks with arbitrary parallel structure," in *RTCSA'11*, 2011, pp. 3–14.

[6] K. Lakshmanan, S. Kato, and R. Rajkumar, "Scheduling parallel real-time tasks on multi-core processors," in *RTSS'10*, 2010, pp. 259–268.

[7] A. Saifullah, K. Agrawal, C. Lu, and C. Gill, "Multi-core real-time scheduling for generalized parallel task models," in *RTSS'11*, November 2011, pp. 217–226.

[8] G. Nelissen, V. Berten, V. Nélis, J. Goossens, and D. Milojevic, "U-EDF: An unfair but optimal multiprocessor scheduling algorithm for sporadic tasks," in *ECRTS '12*, July 2012, to appear.

[9] A. Srinivasan and J. Anderson, "Fair scheduling of dynamic task systems on multiprocessors," *Journal of Systems and Software*, vol. 77, no. 1, pp. 67–80, Avril 2005.

[10] H. Cho, B. Ravindran, and E. D. Jensen, "An optimal real-time scheduling algorithm for multiprocessors," in *RTSS '06*, 2006, pp. 101–110.

[11] G. Levin, S. Funk, C. Sadowski, I. Pye, and S. Brandt, "DP-Fair: A simple model for understanding optimal multiprocessor scheduling," in *ECRTS '10*, July 2010, pp. 3–13.

[12] V. Berten, P. Courbin, and J. Goossens, "Gang fixed priority scheduling of periodic moldable real-time tasks," in *JRWRTC '11*, 2011, pp. 9–12.

[13] J. Goossens and V. Berten, "Gang FTP scheduling of periodic and parallel rigid real-time tasks," in *RTNS'10*, 2010, pp. 189–196.

[14] S. Kato and Y. Ishikawa, "Gang EDF scheduling of parallel task systems," in *RTSS'09*. IEEE Computer Society, 2009, pp. 459–468.

[15] S. Collette, L. Cucu, and J. Goossens, "Integrating job parallelism in real-time scheduling theory," *Information Processing Letters*, vol. 106, no. 5, pp. 180–187, May 2008, extended version.

[16] I. Lupu and J. Goossens, "Scheduling of hard real-time multi-thread periodic tasks," in *RTNS'11*, September 2011, pp. 35–44.

[17] J. Sun and J. Liu, "Synchronization protocols in distributed real-time systems," in *ICDCS*, 1996, pp. 38–45.

[18] T. P. Baker and S. K. Baruah, *Schedulability analysis of multiprocessor sporadic task systems*. CRC Press, 2007, ch. 3.

[19] G. Nelissen, V. Berten, J. Goossens, and D. Milojevic, "Optimizing the number of processors to schedule multi-threaded tasks," in *RTSS'11-WiP Session*, December 2011, pp. 5–8.

[20] G. B. Dantzig, A. Orden, and P. Wolfe, "The generalized simplex method for minimizing a linear form under linear inequality restraints," *Pacific Journal of Mathematics*, no. 5, pp. 183–195, 1955.

[21] S. J. Wright, *Primal-dual interior-point methods*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1997.

[22] S. Tsukiyama, M. Ide, H. Ariyoshi, and I. Shirakawa, "A new algorithm for generating all the maximal independent sets," *SIAM Journal on Computing*, vol. 6, no. 3, pp. 505–517, 1977.