Schedulability Analysis for Directed Acyclic Graphs on Multiprocessor Systems at a Subtask Level

Manar Qamhieh, Serge Midonnet

Schedulability Analysis for Directed Acyclic Graphs on Multiprocessor Systems at a Subtask Level

Manar Qamhieh & Serge Midonnet
{manar.qamhieh,serge.midonnet}@univ-paris-est.fr

Université Paris-Est, France

Abstract. This paper addresses the problem of scheduling parallel real-time tasks of Directed Acyclic Graph (DAG) model on multiprocessor systems. We propose a new scheduling method based on a subtask-level, which means that the schedulability decisions are taken based on the local temporal parameters of subtasks. This method requires modifying the subtasks to add more parameters which are necessary for the analysis, such as local offsets, deadlines and release jitters. Then we provide interference and workload analyses of DAG tasks, and we provide a schedulability test for any work conserving scheduling algorithm.

1 Introduction

Recently, the performance of systems has been increased using multiprocessors instead of uniprocessors to overcome processor physical limitations and to produce faster and smaller processors. The use of parallelism in software makes them compatible with multiprocessor hardware, because the calculations of parallel applications are performed on multiple processors simultaneously.
In real-time systems, scheduling parallel real-time tasks on multiprocessor systems is a challenging problem, and the extension of uniprocessor schedulability conditions to parallel multiprocessor systems is not trivial. The need of synchronization between parallel tasks and processors makes the scheduling process more complicated.

In this paper, we are interested in scheduling Directed Acyclic Graph (DAG) tasks on multiprocessor systems. The contribution is to provide schedulability conditions for DAG tasks that take into account subtasks' parameters instead of DAG-level parameters (a solution commonly used in previous researches found in literature).

The remainder of this paper is organized as follows. In Section 2, we present a state-of-the-art of methods relative to real-time parallel task scheduling on multiprocessor systems especially for the DAG model. The considered model and the used terminology are described in Section 3. In Section 4, we explain the subtask-level scheduling process, and we define additional parameters to subtasks in order to be scheduled individually using any work conserving algorithm. A workload analysis is given in Section 5. This workload analysis is used to derive

a schedulability test for any work conserving scheduling algorithms. Finally, we conclude this study and show future work in Section 6.

2 Related work

Many hard real-time scheduling algorithms and schedulability analyses on homogeneous multiprocessor systems have been proposed in the literature [1]. They mostly focus on the traditional sequential independent real-time task model.

Regarding parallel tasks, there are different models and each has its own advantages and limitations. First there is the Fork-join model, in which a parallel task is an alternating sequence of parallel and sequential segments, a stretching algorithm to execute parallel segments as sequential as possible was proposed in [2].

A more general model of parallel tasks, called the segment model, has been studied in the literature. The multiprocessor scheduling of periodic tasksets with implicit deadlines with this model has been addressed in [3]. This model represent a task as a sequence of segments, each segment consists of a number of identical threads.

The analysis has been extended to the DAG model and the same results can be applied. Another scheduling approach based on the response time analysis for the multi-threaded segment model has been provided in [4] for soft real-time multi-core systems.

The DAG model has been studied in [5] in the uniprocessor case. The authors considered a hybrid task set of periodic independent tasks and dependent sporadic graph tasks that execute only once. A graph task in this model consists of a set of tasks with precedence constraints and each task has a release time and deadline. They proposed an algorithm based on a modification of task parameters in order to remove the dependencies between the tasks in the analysis. We use a similar technique in Section 4 to modify subtasks with few differences due to the characteristics of the model.

A capacity augmentation bound of $4-\frac{2}{m}$ and a resource augmentation bound of $2-\frac{1}{m}$ have been proposed recently for GEDF scheduling of periodic implicit-deadline DAG tasksets in [6], where m is the number of processors in the system. Also, Bonifaci et al. [7] studied the schedulability of a DAG set on multiprocessor systems. They proved that GEDF has a speedup bound of $2-1/m$, and Deadline monotonic a speedup bound equal to $3-1/m$. It is worth noticing that the above described scheduling methods do not consider the internal structure of DAGs in the analysis. And the parallel DAG tasks are either transformed into a collection of independent sequential tasks or they are scheduled directly while considering the global parameters of the DAGs, such as their total worst-case execution time and the critical path length.

More recently, we proposed in [8] a schedulability test of periodic implicit-deadline DAG tasks when GEDF is used. The schedulability decisions are based on a DAG-level (the global deadline of the DAGs), while the workload analysis of the test considered the internal structure of DAGs. We proved by experimental

results that this method reduces analysis pessimism and enhances scheduling performance for DAGs. This paper is an extension of this work, in which we aim at enhancing the scheduling analysis by proposing a DAG scheduling on a subtask-level. To our best knowledge, no similar research exists using the method to address the problem of scheduling parallel DAG tasks.

3 System model

In this paper, we consider a taskset τ of n real-time Directed Acyclic Graph (DAG) tasks scheduled on m identical processors. Each DAG task τ_i is a sporadic constrained-deadline graph composed of n_i subtasks under precedence constraints.

A DAG task τ_i is characterized by $(n_i, \{1 \leq j \leq n_i | \tau_{i,j}\}, G_i, D_i, T_i)$, where n_i is the number of its subtasks, the second parameter is the set of subtasks, G_i is the set of directed relations between the subtasks, D_i is the relative deadline of τ_i and T_i is the minimum inter-arrival time between the successive jobs.

Let $\tau_{i,j}$ denote the j^{th} subtask of the set of subtasks forming DAG task τ_i, where $1 \leq j \leq n_i$. Each subtask $\tau_{i,j}$ is a single-threaded task that has a single timing parameter which is its worst case execution time (WCET) $C_{i,j}$. The subtasks of a DAG inherit the period and deadline of their DAG.

Let $g_{i,k}^{i,j} \in G_i$ represent a directed link from subtask $\tau_{i,j}$ to $\tau_{i,k}$. A direct link between subtask $\tau_{i,j}$ and $\tau_{i,k}$ means that subtask $\tau_{i,k}$ cannot start its execution unless subtask $\tau_{i,j}$ completes its own. In this case, subtask $\tau_{i,j}$ is called a ***parent*** subtask of $\tau_{i,k}$ where $\tau_{i,j} \in parents(\tau_{i,k}) \in pred(\tau_{i,k})$, where $pred(\tau_{i,k})$ is the set of all predecessors of subtask $\tau_{i,k}$ from the source of the DAG which have to execute indirectly before $\tau_{i,k}$ (such as parents of $\tau_{i,k}$'s parents). Likewise, subtask $\tau_{i,k} \in children(\tau_{i,j})$ is called a ***child*** subtask of $\tau_{i,j}$, and the set of all successors of $\tau_{i,k}$ is denoted by $succ(\tau_{i,k})$. Subtask $\tau_{i,j}$ may have zero or more parent/children subtasks. A source subtask has no parent subtasks and a sink subtask is the one without any successors.

Let C_i denote the total WCET of DAG task τ_i, where $C_i = \sum_{k=1}^{n_i} C_{i,k}$. Let L_i denote the length of the critical path of DAG task τ_i, which is defined as the longest execution path in τ_i when it executes on a platform of infinite number of processors.

We assume that each DAG task τ_i generates an infinite sequence of jobs. Let J_i^k be the k^{th} job of DAG task τ_i which is characterized by $(r_i,\ d_i)$, where r_i is the release time of the job, and d_i is its absolute deadline. Each DAG job J_i^k consists of a collection of subtask jobs each is denoted by $J_{i,j}^k$, $j \in 1 \ldots n_i$. In the remainder of this paper, the numeration of jobs is removed when it is unnecessary for the clarity of the discussion.

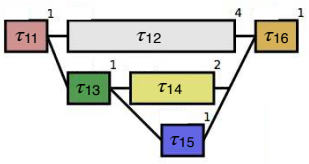
Figure 1(a) shows an example of a DAG task τ_1 which consists of 6 subtasks. Subtask $\tau_{1,1}$ is the source of the DAG and $\tau_{1,6}$ is its sink. The lines in the figure represent the directed precedence constraints between the subtasks. The critical path of τ_1 is $\{\tau_{1,1}, \tau_{1,2}, \tau_{1,6}\}$ and its length is $L_i = 6$. For subtask $\tau_{1,5}$, its parent subtask is $\tau_{1,3}$ while $\tau_{1,1}$ is one of its predecessor.

For any DAG taskset, there are two necessary basic conditions, if at least one of them is false, the taskset is not feasible:

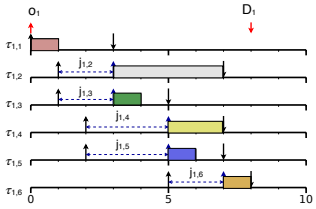
$$\sum_{\tau_i \in \tau} \frac{C_i}{T_i} \leq m$$

$$\forall \{\tau_i \in \tau\} : L_i \leq D_i$$

In the following sections, we provide a schedulability analysis for DAG tasks using any global work conserving scheduling algorithm. A global algorithm allows job migration and preemptions between processors (migration costs and preemption costs are not taken into account in this work), while a work conserving algorithm does not authorize delaying the execution of an active job if there is an idle processor in the system.



(a) A DAG task τ_1 consists of 6 subtasks with parameters: $D_1 = 8$, $C_1 = 10$, $L_1 = 6$.



(b) Time diagram of each subtask of DAG task τ_1

Fig. 1. Example of DAG model.

4 DAG task Scheduling

When scheduling DAG sets on multiprocessor systems, the interference on each DAG task has the following two sources:

- an external interference from jobs of higher priority DAG tasks, in which some or all of the interfering subtasks can contribute to the interference,
- an internal interference from the subtasks of the same DAG on each other.

For parallel DAG tasks, a DAG-level scheduling algorithm means that the scheduling decisions are based on global parameters of the DAG tasks. According to this scheduling, the priorities are assigned to DAG tasks which are applied then to their respective subtasks. The DAG-level schedulability analysis depends on the global parameters of the DAGs, such as their deadline, period, total WCET and the length of their critical path. Usually, the resulted schedulability tests are pessimistic, and the internal structure of the DAGs is not considered in the performed analysis.

Hence, the interference analysis of DAG-level scheduling is difficult to be calculated and it is harder to identify the exact sources of interference. However, if the scheduling algorithm uses extra knowledge about subtasks and their execution flow, then the interference analysis can be more accurate and precise. In this case the scheduling process is said to be done at a subtask-level. According to this, subtasks will be assigned priorities based on the scheduling algorithm. However, the schedulability analysis requires extra temporal parameters for each subtask other than its WCET provided by the DAG model.

Further in this paper, we will propose a technique to add local temporal parameters to the subtasks based on their dependencies and the precedence constraints between them. As a result, the problem of scheduling parallel DAG tasks on multiprocessor systems will be simplified to scheduling a set of independent sequential subtasks on multiprocessor systems, which is widely studied in the literature.

4.1 Subtask analysis and modification

In a previous work [8], we provided algorithms to add two temporal parameters, in addition to the WCET parameter (provided by the model), to each subtask in the DAG set. This was done to improve the interference analysis of DAG scheduling in the case of GEDF scheduling. These two parameters are the local offset and deadline of each subtask, and they were derived from the internal structure of the DAG task and the execution flow of the subtasks in the best case scenario. We consider that this scenario happens when a DAG set executes on an infinite number of processors, all of its subtasks execute in parallel as soon as possible without being delayed due to interference.

Definition 1. *A local offset $O_{i,j}$ of a subtask $\tau_{i,j}$ is defined as the earliest possible activation time of the subtask w.r.t. the release of its DAG task τ_i.*

Definition 2. *A local deadline $D_{i,j}$ of a subtask $\tau_{i,j}$ is the latest time for subtask $\tau_{i,j}$ to complete its execution so as to leave enough time for its successors to execute within their local deadlines.*

For each subtask, the local offset and deadline are calculated using straightforward depth-first search algorithms (a detailed description can be found in [8]). The local offset of a subtask takes into consideration the time needed for its predecessor subtasks to execute in the best case scenario. Respectively, the local deadline of a subtask leaves enough time for its successor subtasks to execute.

Observation 1 *If a subtask $\tau_{i,j} \in \tau_i$ misses its local deadline $D_{i,j}$, then its DAG task τ_i will definitely miss its deadline D_i.*

Based on the definition of the local deadline, when a subtask misses its local deadline, the time remaining until the global deadline of the DAG task is not enough for the successors to execute in the best case. Therefore, an early schedulability failure can be announced based on the subtask's deadline instead of waiting for the DAG's deadline miss.

Figure 1 shows the mapping between the DAG task τ_1 and its subtasks after the modification. The timing diagram of each subtask $\tau_{1,j}$, where $1 \leq j \leq 6$, of the DAG is shown in Figure 1(b). The local offset $O_{1,1}$ of the source subtask $\tau_{1,1}$ equals to 0 because this subtask has no predecessors and it is released at the release time of the DAG task τ_1. Its local deadline $D_{1,1} = 3$ because its successors need at least 5 time units to execute in the best case, which is the longest path from $\tau_{1,1}$ (excluded) to a sink subtask ($\{\tau_{1,2}, \tau_{1,6}\}$ in the example). Then, both subtasks $\tau_{1,2}$ and $\tau_{1,3}$ are released after the completion of $\tau_{1,1}$. As a result, $O_{1,2} = O_{1,3} = 1$. For the rest of the subtasks, their local offsets and deadlines are shown in Figure 1(b).

The local offsets and deadlines of subtasks are calculated based on their best-case activation scenario, in which all the subtasks execute as soon as they are released with no interference or delays, These parameters helps in identifying the longest execution window of each subtask. However, the activation of a subtask job can be delayed due to the interference of higher priority subtasks. Respectively, the latest possible activation of a subtask job occurs when all of its predecessors execute as late as possible (just before their local deadlines). According to this, the activation of a subtask job can happen at any time within this interval, and this can be considered as the maximum release jitter of the subtask.

Definition 3. *A maximum release jitter $\widehat{j}_{i,j}$ of a subtask $\tau_{i,j}$ is defined as the difference between the earliest and latest release time of the subtask with respect to the activation of the DAG task.*

$$\widehat{j}_{i,j} = \max_{\forall \tau_{i,k} \in Parents(\tau_{i,j})} (D_{i,k} - (O_{i,j} - O_{i,k})) \tag{1}$$

Based on the definition of the release jitter of a subtask, all of its jobs are released within the jitter interval, which is shown in the following equation:

$$\forall J^j_{i,k}\ ,\ j \in \mathbb{N}, \tau_{i,k} \in \tau_i\ :\ r^j_{i,k} \in [O_{i,k}, O_{i,k} + \widehat{j}_{i,j}]$$

Figure 1(b) shows the maximum release jitter values for the subtasks in DAG τ_1 from Figure 1(a). It's worth noticing that the source subtask $\tau_{1,1}$ has no jitter since it has no predecessor subtasks.

From Equation 4.1 and the example in Figure 1(b), we can notice that the calculation of the release jitter of the subtasks is pessimistic, and it considers always that the predecessor subtasks execute as late as possible. According to this, the critical subtasks of a DAG task (subtasks forming its critical path) will have no slack time[1] if they are activated at their maximum release jitter. In the following sections, we provide an optimization to the release jitter of subtasks based on the interference analysis.

Regarding the period of the subtasks (minimum inter-arrival time for sporadic tasks), each subtask inherits the period of its DAG task, where $\forall \tau_{i,j} \in \tau_i,\ T_{i,j} = T_i$.

[1] Slack time is the time difference between the deadline of a task and its WCET.

As a result, the subtasks of a given DAG task are characterized now by a local offset, a WCET, a local deadline and a release jitter. These parameters will allow us to treat subtasks individually and independently in order to provide a schedulability analysis at a subtask-level.

4.2 Interference Analysis

For the subtask-level scheduling process using any work conserving algorithm, the execution of a subtask can be blocked by higher priority subtasks. The interference on a subtask $\tau_{k,h} \in \tau_k$ is defined as follows:

Definition 4. *$I_{k,h}(a,b)$ is the length of all intervals where subtask $\tau_{k,h}$ is ready to execute but blocked by higher priority subtasks in an interval $[a,b)$.*

Definition 5. *$I^{i,j}_{k,h}(a,b)$ is the length of all intervals where subtask $\tau_{k,h}$ is ready to execute but blocked by subtask $\tau_{i,j}$ which has higher priority in an interval $[a,b)$.*

Since the subtasks are single-threaded sequential real-time tasks, the relation between $I_{k,h}(a,b)$ and $I^{i,j}_{k,h}(a,b)$ is denoted by the following equation:

$$I_{k,h}(a,b) = \frac{1}{m} * \sum_{\forall \tau_{i,j} \in \tau_i \in \tau} I^{i,j}_{k,h}(a,b) \tag{2}$$

Due to the characteristics of the DAG tasks and the precedence constraints between the subtasks, the interference on a subtask $\tau_{k,h}$ is divided into two sources, external and internal interference. Let $Ie_{k,h}(a,b)$ denote the interference from higher priority subtasks of DAG tasks other than τ_k in the set, which is defined as follows:

$$Ie_{k,h}(a,b) = \frac{1}{m} * \sum_{i \neq k, \forall \tau_{i,j} \in \tau_i} I^{i,j}_{k,h}(a,b) \tag{3}$$

Where some or all of the subtasks of DAG task τ_i $(i \neq k)$ can interfere with $\tau_{k,h}$ based on their priorities.

Furthermore, subtasks of DAG τ_k can block the execution of $\tau_{k,h}$ which is defined as the internal interference $Ii_{k,h}(a,b)$. Since we consider constrained deadline DAG tasks, for a given job of subtask $\tau_{k,h}$, one job at most from each subtask contributes to the interference. The internal interference depends on the type of interfering subtasks which are divided into the following categories:

- a predecessor subtask $\tau_{k,x} \in pred(\tau_{k,h})$ of subtask $\tau_{k,h}$: this subtask will delay the activation of $\tau_{k,h}$, but once subtask $\tau_{k,x}$ completes its execution, subtask $\tau_{k,h}$ will start its own and there will be no further effect of $\tau_{k,x}$ on $\tau_{k,h}$,
- a sibling subtask $\tau_{k,x} \in sibling(\tau_{k,h})$ is the subtask that executes in parallel with no dependencies with subtask $\tau_{k,h}$. The $sibling(\tau_{k,h})$ is the set of subtasks that are not predecessors or successors of subtask $\tau_{k,h}$,

– a successor subtask $\tau_{k,x} \in succ(\tau_{k,h})$ has no interference with subtask $\tau_{k,h}$, because both subtasks cannot execute in parallel and subtask $\tau_{k,x}$ starts its execution after $\tau_{k,h}$ completes its own. According to this, $I_{k,h}^{k,x}(a,b) = 0$,
– subtask $\tau_{k,h}$ has no interference on itself since we consider constrained deadline DAG tasks, in which only one job of each DAG task is activated at any time t. Hence, $I_{k,h}^{k,h}(a,b) = 0$.

Based on the above definitions, the internal interference $Ii_{k,h}(a,b)$ on subtask $\tau_{k,h}$ in the interval $[a,b)$ is defined as:

$$Ii_{k,h}(a,b) = \frac{1}{m} * \sum_{\forall \tau_{k,i} \notin succ(\tau_{k,h}); i \neq h} I_{k,h}^{k,i}(a,b) \tag{4}$$

Let $J_{k,h}^*$ be the job of subtask $\tau_{k,h}$ which has maximum interference, and let $I_{k,h}(r_{k,h}^*, d_{k,h}^*)$ denote the worst-case interference for subtask job $J_{k,h}^*$ of $\tau_{k,h}$ in the interval $[r_{k,h}^*, d_{k,h}^*)$. For the sake of clarity, we will use $\widehat{I}_{k,h} = I_{k,h}(r_{k,h}^*, d_{k,h}^*)$ in this document.

Lemma 1. *A taskset τ, of sporadic constrained deadline DAG tasks, is schedulable on m identical processors, for any work conserving algorithm if:*

$$\forall \tau_{k,h} \in \tau_k \in \tau$$
$$\widehat{I}_{k,h} = \widehat{Ie}_{k,h} + \widehat{Ii}_{k,h} \leq (D_{k,h} - C_{k,h}) \tag{5}$$

Proof. The proof of this lemma is straight-forward. The interference on a subtask job has two main sources; internal and external. In order for any subtask to be schedulable, its execution window (between its activation and deadline) should be enough to execute its execution time plus the interference workload which is identified above.

4.3 Interference from Predecessor subtasks

As described earlier, a predecessor subtask should complete its execution before its successors are activated. Hence, a successor subtask can only be delayed by its predecessors. In Section 4.1, we assigned a maximum release jitter for each subtask in the DAG. This parameter represents the interval in which the subtask job can be activated and it replaces the dependencies between the subtasks. If we consider that all the predecessors of $\tau_{k,h}$ have executed as late as possible, then the subtask job $J_{k,h}^*$ will be delayed to the end of this interval ($\widehat{j}_{k,h}$ time units after its offset), then the condition in Lemma 1 will be modified as follows:

Lemma 2. *A taskset τ of DAG tasks is schedulable on m identical processors, using any work conserving algorithm, if:*

$$\forall \tau_{k,h} \in \tau_k \in \tau$$
$$\widehat{Ie}_{k,h} + \widehat{Ii}_{k,h} \leq (D_{k,h} - C_{k,h} - \widehat{j}_{k,h}) \leq (D_{k,h} - C_{k,h} - j_{k,h}) \tag{6}$$

where

$$\widehat{Ii}_{k,h} = \frac{1}{m} * \sum_{\forall \tau_{k,i} \in sibling(\tau_{k,h})} I^{k,i}_{k,h}(a,b)$$

Proof. As shown in Figure 2 and based on the definition of the release jitter in Equation 4.1, if all predecessors respected their local deadlines, then subtask $\tau_{k,h}$ will be activated no later than $t = (r_k + O_{k,h} + \widehat{j}_{k,h})$. In the interval $[t, d_{k,h})$, predecessors will have no further interference, and only sibling subtasks of $\tau_{k,h}$ will interfere with $\tau_{k,h}$, which should be less than the available slack time $(D_{k,h} - C_{k,h} - j_{k,h})$.

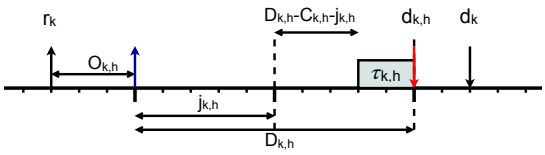


Fig. 2. The interference window excluding interference from predecessor subtasks.

We have mentioned earlier that considering the maximum release jitter $\widehat{j}_{k,h}$ of subtask $\tau_{k,h}$ in the interference analysis is too pessimistic, because it considers that the predecessors will execute as late as possible. As a result, the upper bound of interference used in Lemma 2 will be always zero for any critical subtask $\tau_{k,h}$ since their release jitter $\widehat{j}_{k,h} = D_{k,h} - C_{k,h}$.

As shown in Figure 3, it is possible to optimize the jitter of each subtask by knowing that a parent subtask $\tau_{k,i}$ has a response time equal to $(\widehat{I}_{k,i} + C_{k,i})$ when a work conserving algorithm is used. Its latest finish time $f^*_{k,i}$ can be defined as:

$$f^*_{k,i} = C_{k,i} + \widehat{I}_{k,i} \leq D_{k,i}$$

The finish time $f_{k,i}$ of any job of a schedulable subtask $\tau_{k,i}$ should not be greater than its local deadline $D_{k,i}$, or a deadline miss will occur.

By using the finish time of each parent subtask of $\tau_{k,h}$, we can calculate an optimized release jitter $j'_{k,h}$ defined by the following equation:

$$\begin{aligned} j'_{k,h} &= \max_{\forall \tau_{k,i} \in parents(\tau_{k,h})} (f^*_{k,i} - (O_{k,h} - O_{k,i})) \\ &= \max_{\forall \tau_{k,i} \in parents(\tau_{k,h})} (C_{k,i} + \widehat{I}_{k,i} - (O_{k,h} - O_{k,i})) \\ &\leq \widehat{j}_{k,h} \end{aligned} \tag{7}$$

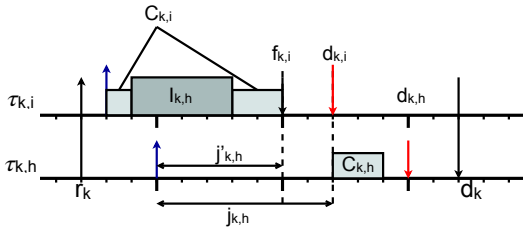


Fig. 3. The optimized release jitter of subtask $\tau_{k,h}$ from its sole parent $\tau_{k,i}$.

Corollary 1. *A taskset τ of DAG tasks is schedulable on m identical processors, using any work conserving algorithm, if:*

$$\forall \tau_{k,h} \in \tau_k \in \tau$$

$$\widehat{Ie}_{k,h} + \widehat{Ii}_{k,h} \leq (D_{k,h} - C_{k,h} - j'_{k,h}) \quad (8)$$

where

$$\widehat{Ii}_{k,h} = \frac{1}{m} * \sum_{\forall \tau_{k,i} \in sibling(\tau_{k,h})} I^{k,i}_{k,h}(a,b) \quad (9)$$

The use of the optimized release jitter of a subtask instead of its maximum release jitter improves the schedulability test by considering a more accurate upper bound on interference.

5 Workload Analysis

It is difficult to identify the actual interference from external and sibling subtasks required for the schedulability test in Corollary 1. However, we can use an upper bound on the interference based on the workload computation of an interfering subtask, knowing that the interference of a subtask on another one in a fixed interval cannot exceed the workload of the interfering subtask during the same interval. Let $W_{i,j}(a,b)$ be the amount of work done by the jobs of subtask $\tau_{i,j}$ in the interval $[a,b)$. Then:

$$I^{i,j}_{k,h}(a,b) \leq W_{i,j}(a,b)$$

Within the interference interval $[a,b)$, let a carry-in job of an interfering subtask be defined as the job that is released before the start of the interval and has a deadline within the interval. While a body job is the job that is released within the interval $[a,b)$ and its deadline can be within or after the end of the interval.

5.1 Workload from sibling subtasks

A sibling subtask $\tau_{k,i}$ of $\tau_{k,h}$ is the subtask from the same DAG task τ_k that can execute in parallel with $\tau_{k,h}$. Moreover, subtask $\tau_{k,i}$ has no precedence relations with $\tau_{k,h}$ and it cannot be among its predecessors or successors.

For a given subtask job, one job at most from each sibling subtask will interfere with it, because the jobs of sibling subtasks belong to the same DAG job, and the release of their DAG job is considered as their activation reference. In other words, one job from each sibling subtask $\tau_{k,h}$ is released in the interval $[r_k + O_{k,h}, r_k + O_{k,h} + \widehat{j}_{k,h}]$. For any work conserving algorithm, the interference $\widehat{Ii}_{k,h}^{k,i}$ of a subtask $\tau_{k,i}$ on its sibling $\tau_{k,h}$ is calculated by identifying the maximum interfering interval $L_{k,h}^{k,i}$ of $\tau_{k,i}$ on $\tau_{k,h}$. This interval is defined as the longest interval in which subtasks $\tau_{k,h}$ and $\tau_{k,i}$ can execute in parallel. It is calculated as follows:

$$L_{k,h}^{k,i} = min(D'_{k,i}, D'_{k,h}) - max(O_{k,i}, O_{k,h}) \tag{10}$$

For the sake of clarity, we considered $D'_{k,h}$ to be the relative deadline of subtask $\tau_{k,h}$ from the release of the DAG task, where $D'_{k,h} = O_{k,h} + D_{k,h}$.

Lemma 3. *The maximum internal interference $\widehat{Ii}_{k,h}^{k,i}$ of subtask $\tau_{k,i}$ on a job of its sibling subtask $\tau_{k,h}$ is*

$$\widehat{Ii}_{k,h}^{k,i} \leq min(C_{k,i}, L_{k,h}^{k,i}) = \widehat{Wi}_{k,i} \tag{11}$$

Proof. Based on the definition of the interference interval $L_{k,h}^{k,i}$, the maximum possible workload of subtask $\tau_{k,i}$ in the interval happens when $\tau_{k,i}$ executes as long as possible in this interval. From here comes the *min* in the Equation 11.

5.2 Workload Analysis for External subtasks

For any work conserving algorithm, Bertogna et al. identified, in their paper [9], the worst-case activation scenario of jobs of an interfering task in a fixed interval (a, b) which generates the maximum possible workload. They considered a task model of independent sequential single-threaded tasks. As shown in Figure 4, this scenario happens when the carry-in job of the interfering task starts its execution at the beginning of the interference window and executes as late as possible. The following body jobs then execute as soon as possible until the end of the window. This scenario is proved in [9] to generate the maximum workload in the interval.

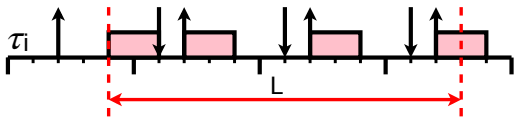
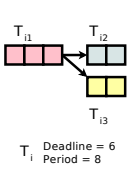


Fig. 4. The densest possible packing of jobs in interval pf length L for traditional task using any work conserving algorithm.

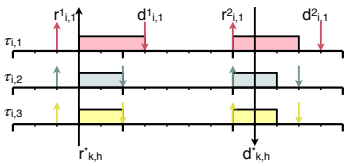
We use this scenario to calculate the workload of each external subtask in order to be used as an upper bound of its interference on a given subtask in the

system. Since each external subtask $\tau_{i,j}$ has no precedence constraints with $\tau_{k,h}$ (where $k \neq i$), then this scenario can be applied to each subtask independently. As shown in Figure 5, the subtasks of an interfering DAG task τ_i will interfere on subtask job $J^*_{k,h}$. Assume that subtask job $J^*_{k,h}$ has an activation window $[r^*_{k,h}, d^*_{k,h})$ as shown in Figure 5(b), while Figure 5(a) shows the DAG task τ_i and its internal structure. In order to calculate the worst-case workload of its subtasks on $j_{k,h}$, the worst-case activation scenario is applied to each subtask $\tau_{i,j}$. As shown in Figure 5(b), the first job of each subtask starts its execution as late as possible at the beginning of the interference interval, and the following job executes as soon as possible. Based on this scenario, the maximum workload done in the interference interval is 10.

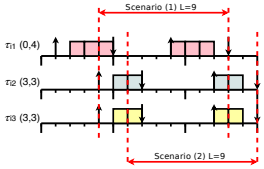
However, applying this scenario on each subtask of the same DAG task independently is pessimistic. Because in reality, these interfering subtasks have precedence constraints that define their execution flow. For example, subtask $\tau_{i,1}$ in Figure 5(b) cannot execute in parallel with its children subtasks $\tau_{i,2}$ and $\tau_{i,3}$. But still, the workload in this activation scenario can be used as an upper bound for workload. Using the following example, we will show that it is not trivial to find a worst-case activation scenario of jobs adapted to DAG tasks that generates the maximum workload.



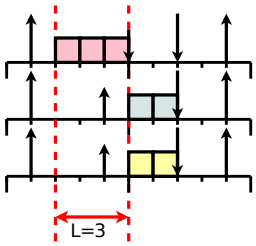
(a) Example of a DAG task τ_i.



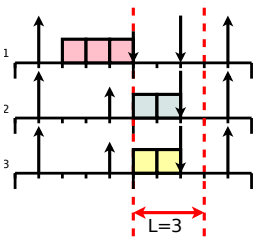
(b) The worst workload activation scenario for subtask jobs of τ_i for any work conserving algorithm.



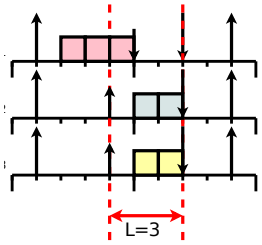
(c) The worst workload activation scenario for subtask jobs of τ_i for Global EDF.



(d) First possible activation scenario of subtasks of τ_i. Total workload is 3.



(e) Second possible activation scenario of subtasks of τ_i. Total workload is 4.



(f) Third possible activation scenario of subtasks of τ_i. Total workload is 5.

Fig. 5. Workload analysis for external subtasks.

Example
Back to the DAG task τ_i from Figure 5(a). According to the precedence constraints between its subtasks, the activation scenario of its subtask jobs is shown in Figures 5(d)-5(f). We consider an interference interval of length $L = 3$. Each Figure in 5(d)-5(f) shows a possible position of the interference interval w.r.t. to the interfering subtasks. For example, Figure 5(d) considers that subtask $\tau_{i,1}$ starts at the beginning of the interfering interval, and its total workload is 3. While Figure 5(e) considers that subtasks $\tau_{i,2}$ and $\tau_{i,3}$ start at the beginning of the interval and the total workload is 4. However, the maximum workload happens in Figure 5(f), in which the interference interval starts within subtask $\tau_{i,1}$ and it ends at the deadline of $\tau_{i,2}$ and $\tau_{i,3}$. In this case the total workload is 5.

Based on this example, we conclude that in order to calculate the maximum workload of external subtasks of the same DAG, we have to analyze all the possible positions of interference interval w.r.t. the activation of subtasks, and this is done at each time instant in the interfering interval.

Lemma 4. *The external interference $\widehat{Ie}_{k,h}^{i,j}$ of subtask $\tau_{i,j}$ on subtask $\tau_{k,h}$ in an interval, whose length is equal to the absolute deadline $D_{k,h}$ of $\tau_{k,h}$, is bounded by:*

$$\begin{aligned}\widehat{Ie}_{k,h}^{i,j} \leq\ & N_{i,j}(D_{k,h})C_{i,j}+ \\ & min(C_{i,j}, D_{k,h} + D_{i,j} - C_{i,j} - N_{i,j}(D_{k,h})T_{i,j})\end{aligned} \tag{12}$$

where

$$N_{i,j}(D_{k,h}) = \lfloor \frac{D_{k,h} + D_{i,j} - C_{i,j}}{T_{i,j}} \rfloor$$

Proof. The maximum interference workload from the external subtask $\tau_{i,j}$ on subtask $\tau_{k,h}$ happens based on the execution scenario described in [9] and shown in Figure 4. The calculations of workload is based on number of interfering jobs which lie completely within the interfering window plus the last job in the interval which may contribute partially in the interference. More details about these equations can be found in [9].

A schedulability test for DAG tasks using any work conserving algorithm on m identical processors is provided as follows:

Theorem 1. *A DAG set τ is schedulable on m identical processors using any work conserving algorithm if:*

$$\begin{aligned}&\forall \tau_{k,h} \in \tau_k \in \tau \\ &\sum_{\tau_{k,i} \in sibling(k,h)} min(\widehat{Ii}_{k,h}^{k,i}, D_{k,h} - C_{k,h} - j'_{k,h}) + \\ &\sum_{\tau_{i,j}\ ;\ i \neq k} min(\widehat{Ie}_{k,h}^{i,j}, D_{k,h} - C_{k,h} - j'_{k,h}) \\ &\leq m(D_{k,h} - C_{k,h} - j'_{k,h})\end{aligned}$$

Proof. Knowing that the interference of a subtask in a given interval can never exceed the workload of this subtask in the same interval, we can transform the interference schedulability bound on subtask $\tau_{k,h}$ described in Lemma 3 into a workload bound of schedulability of the same subtask. However, the internal and the external interference are based on their respective workload calculations shown in Equations 11 and 12,

The schedulability test described in the above theorem can be used to optimize the release jitter value of each subtask. Based on the test, the optimized release jitter can be calculated for each successor of the subtask. If the calculated release jitter is more than the actual release jitter (the maximum release jitter by default), then the calculated value is discarded and the actual release jitter will not be modified.

In order to analyze the performance of our schedulability test, we compare it with another test found in literature. Bonifaci et al. in [7] provided a GEDF schedulability test for DAG set on m identical processors. The test depends on the global parameters of the DAG tasks without considering the internal structure and the execution flow of the subtasks. Our schedulability test provided in Theorem 1 is provided for any work conserving algorithm. For the sake of simulation, we derived a special case of this test for GEDF scheduling algorithm and the results are shown in Figure 6.

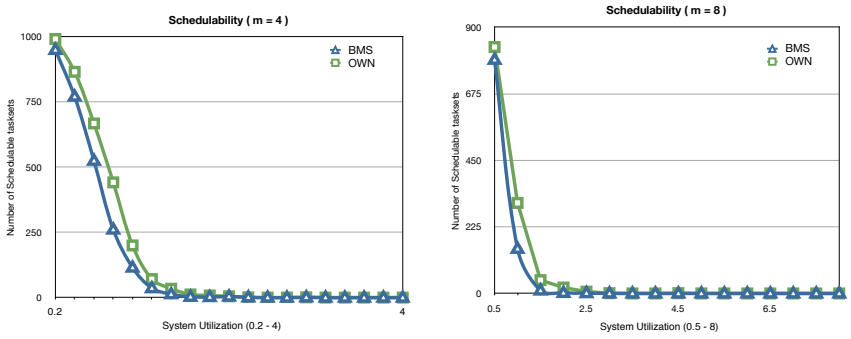


Fig. 6. Simulation results.

We generated a large number of random DAG tasksets, and we applied both GEDF-schedulability tests on the sets of utilization that range from 0 to 8. As shown in Figure 6, our own scheduling test (denoted by *OWN*) performs better than the test from [7] (denoted by *BMS*). For each system utilization, our test schedules more DAG sets than the *BMS* test.

The simulation results provided in this section proves the importance of the internal structure of DAG tasks in the schedulability analysis.

6 Conclusion

In this paper, we were interested in the scheduling of parallel real-time DAG tasks on multiprocessor systems. Our motivation was to show that the scheduling of real-time DAG tasks is affected by the internal structure of the DAG and the execution flow of its subtasks. Hence, we applied the scheduling algorithms at subtask-level instead of DAG-level. This means that the scheduling decisions are based on local parameters of subtasks instead of the global parameters of DAGs.

We modified the subtasks by adding local parameters such as local offset, deadline and release jitter for each subtask. Then we provided interference and workload analyses for any work conserving scheduling algorithm.

As a future perspective, we aim at extending our work to analyze common scheduling algorithms such as EDF and DM, so as to provide precise schedulability test for each algorithm. Also, we aim at providing further analysis for subtask-level schedulers including performance metrics such as speedup factor and approximations ratio. These metrics can be used as an indication of the performance of our proposed scheduling method.

References

1. R. I. Davis and A. Burns, "A survey of hard real-time scheduling algorithms and schedulability analysis techniques for multiprocessor systems," *ACM Computing surveys*, 2011.
2. K. Lakshmanan, S. Kato, and R. (Raj) Rajkumar, "Scheduling Parallel Real-Time Tasks on Multi-core Processors," in *Proceedings of RTSS*, 2010.
3. A. Saifullah, K. Agrawal, C. Lu, and C. Gill, "Multi-core Real-Time Scheduling for Generalized Parallel Task Models," in *Proceedings of RTSS*, 2011.
4. C. Liu and J. H. Anderson, "Supporting Soft Real-Time Parallel Applications on Multicore Processors," in *In proceedings of RTCSA*, 2012.
5. H. Chetto, M. Silly, and T. Bouchentouf, "Dynamic scheduling of real-time tasks under precedence constraints," *Real-Time Systems*, 1990.
6. A. J. Li, K. Agrawal, C. Lu, and C. Gill, "Analysis of Global EDF for Parallel Tasks," in *In proceedings of ECRTS*, 2013.
7. V. Bonifaci, A. Marchetti-spaccamela, S. Stiller, and A. Wiese, "Feasibility Analysis in the Sporadic DAG Task Model," in *In proceedings of ECRTS*, 2013.
8. M. Qamhieh, F. Fauberteau, L. George, and S. Midonnet, "Global EDF Scheduling of Directed Acyclic Graphs on Multiprocessor Systems," in *In proceedings of RTNS*, 2013.
9. M. Bertogna, M. Cirinei, and G. Lipari, "Schedulability Analysis of Global Scheduling Algorithms on Multiprocessor Platforms," *IEEE Transactions on Parallel and Distributed Systems*, 2009.