A Stretching Algorithm for Parallel Real-time DAG Tasks on Multiprocessor Systems

Manar Qamhieh, Laurent George, Serge Midonnet

A Stretching Algorithm for Parallel Real-time DAG Tasks on Multiprocessor Systems

Manar Qamhieh
Université Paris-Est
LIGM
manar.qamhieh@univ-paris-est.fr

Laurent George
Université Paris-Est
LIGM/ESIEE
laurent.george@univ-mlv.fr

Serge Midonnet
Université Paris-Est
LIGM
serge.midonnet@univ-paris-est.fr

ABSTRACT

Parallelism is becoming more important nowadays due to the increasing use of multiprocessor systems. In this paper, we study the problem of scheduling periodic parallel real-time Directed Acyclic graph (DAG) tasks on m homogeneous multiprocessor systems. A DAG task is an example of inter-subtask parallelism. It consists of a collection of dependent subtasks under precedence constraints. The dependencies between subtasks make scheduling process more challenging. We propose a stretching algorithm applied on each DAG tasks to transform them into a set of independent sequential threads with intermediate offsets and deadlines. The threads obtained with the transformation are two types, (i) fully-stretched master threads with utilization equal to 1 and (ii) constrained-deadline independent threads. The fully-stretched master threads are assigned to dedicated processors and the remaining processors $m' \leq m$, are scheduled using global EDF scheduling algorithm. Then, we prove that preemptive global EDF scheduling of stretched threads has a resource augmentation bound equal to $\frac{3+\sqrt{5}}{2}$ for all tasksets with $n < \varphi \cdot m'$, where n is the number of tasks in the taskset and φ is the golden ratio[1].

Keywords

Real-time scheduling, parallel tasks, Directed Acyclic Graphs, global EDF scheduling, speedup factor

1. INTRODUCTION

Increasing the performance of execution platforms has been done by increasing the speed of uniprocessor systems associated to the reduction of the size of chips leading to heating problems. Multiprocessor systems have been seen as one solution to overcome these physical limitations, by increasing execution capabilities with processor parallelism. Many practical examples of shifting towards multiprocessors can be found, such as the *Intel Xeon*, *ARM* and *Cavium* processor families.

However, current real-time software APIs are not able to efficiently take advantage of multiprocessor platforms especially when parallelism is considered. In the industry, the majority of designed applications are targeting uniprocessor systems. But this is expected to change in the coming few years which will focus on parallel programming to take advantage of multiprocessor architectures. There are many models of parallel application, but in this work, we are interested in a particular family of parallelism called inter-subtask parallelism, in which a parallel task consists of a collection of subtasks under precedence constraints and dependencies between subtasks. There are many parallel task models based on this family, such as the Fork-join model which is the base of the famous parallel programming API *OpenMP*, and the multi-threaded Segment model. But the most general model is the Directed Acyclic Graph (DAG) model, which we consider as our task model in this paper.

Real-time scheduling of DAG tasks in particular and parallel tasks in general is a challenging problem. In hard real-time systems, the correctness of a system does not only depend on the correctness of the results, but on the respect of tasks timing parameters. Real-time systems on both uniprocessor and multiprocessor systems have been studied in the last decade, and many researches and scheduling algorithms have been proposed for such platforms [6]. However, the extension of real-time scheduling w.r.t. parallel tasks with dependencies is not trivial. The scheduling analysis of such systems can be divided into two main categories: either by transforming task precedence constraints into independent sequential tasks that execute on multiprocessor systems, or by scheduling parallel tasks directly using adapted scheduling algorithms. The first method simplifies scheduling at the price of losing some of the characteristics of parallel tasks, as it removes subtasks dependencies such that classical scheduling algorithms can then be used. In this work, we study the scheduling of DAG tasks using DAG transformation with a stretching approach.

It is worth mentioning here that this work is a generalization of the stretching algorithm proposed by Lakshmanan *et. al.* in [9]. Their work targeted Fork-join parallel tasks and it aims at scheduling tasks as sequentially as possible by stretching their master threads[2] up to their deadline. They proposed to schedule stretched tasks using partitioned FBB-

[1]The value of the golden ratio is $\frac{1+\sqrt{5}}{2}$

RTNS 2014 , October 8 - 10 2014, Versailles, France

http://dx.doi.org/10.1145/2659787.2659818.

[2]A master thread of a parallel task is defined as the sequential path in the task with the longest execution time.

FFD DM-decreasing scheduling algorithm[3]. In our work, we consider a more general task model which is the DAG task and we provide a resource augmentation bound for global Earliest Deadline First (EDF) scheduling of stretched tasks.

The remainder of this paper is organized as follows. Section 2 presents related works w.r.t. the problem of scheduling real-time tasks on multiprocessor systems especially for parallel DAG model. Then we present in Section 3 the considered task model. In Sections 4 and 5, we explain the DAG stretching algorithm in detail and we provide a resource augmentation bound[4] (speedup factor) analysis. Simulation results are provided in Section 6 to study the performance of our stretching algorithm. Finally, Section 7 concludes this work.

2. RELATED WORK

The scheduling of parallel real-time tasks of different models has been studied on both uniprocessor and multiprocessor systems. In the case of uniprocessor systems, a classical approach in the state-of-the-art is to transform a parallel task into a chain and to assign each subtask of the chain extra timing parameters used for their scheduling. For example, in [4], the authors considered a hybrid taskset of periodic independent tasks and sporadic dependent graph tasks with a DAG model. They proposed an algorithm aiming at modifying the DAG timing parameters (by adding intermediate offsets and deadlines) in order to remove the dependencies between the tasks in the analysis.

In the case of multiprocessor systems, most research has been done regarding scheduling hard real-time tasks on homogeneous multiprocessor systems, especially for independent sequential tasks [6]. As mentioned above, there are mainly two methods for scheduling parallel DAGs in hard real-time systems. The first method schedules DAG tasks by using directly common scheduling algorithms and adapting the performance analysis and the scheduling conditions to take into consideration the particular characteristics of DAGs and parallel tasks in general. This technique is introduced in [2], which considers a taskset of a single sporadic DAG. They also provided polynomial and pseudo-polynomial schedulability tests for EDF scheduling algorithm.

The problem of scheduling multiple DAG tasks on multiprocessors have been more studied in [3, 10]. The authors considered general timing parameters of DAG tasks without taking into account their internal structure. They proved that global EDF has a resource augmentation bound of $2 - 1/m$, where m is the number of processors in the system. In [11], the internal structure and the dependencies between subtasks are considered in the analysis of global EDF scheduling of DAG tasks.

The second method for DAG scheduling is based on DAG transformation. Dependencies of inter-subtask parallelism are removed and a DAG task is transformed into a collection of independent sequential threads. A decomposition algorithm is provided in [12] to distribute the slack time of the DAG task on its subtasks. The slack time is defined as the difference between the deadline of the graph and its minimum sequential execution time. The subtasks are assigned intermediate offsets and deadlines. The authors proved that preemptive global EDF scheduling algorithm has a resource augmentation bound equal to 4, and 4 plus a constant non-preemption overhead for non-preemptive global EDF.

In [9], the authors considered fork-join model of parallel tasks and they proposed a stretching algorithm to execute them as sequentially as possible. A fork-join model is represented as an alternative sequence of sequential and parallel segment. All parallel segments of the same task have the same number of threads, and the threads of each segment have the same sequential execution length. The authors proposed to stretch a fork-join task into a master thread with utilization equal to 1, the remaining parallel threads are forced to execute in parallel with the master thread within a fixed activation interval. Hence, dependencies are no longer considered in the scheduling process. A resource augmentation bound of 3.42 for partitioned preemptive FBB-FFD using Deadline Monotonic (DM) scheduling is given.

Fork-join model is a special case of DAG task model, our paper is an extension of the stretching algorithm of fork-join tasks to DAG tasks. The concept of our stretching algorithm is the same as the one proposed in [9], but the steps and the analysis are adapted to a general DAG task model. The contributions of this paper are as follows:

- we propose a stretching algorithm adapted to DAG task model. This algorithm is a pre-step of the scheduling process of the DAGs, which transforms parallel DAGs into independent sequential threads to be scheduled on m identical processors,
- for a taskset of n DAG tasks that execute on a platform of m identical processors, we prove that global EDF scheduling of stretched tasks has a resource augmentation bound equal to $\frac{3+\sqrt{5}}{2}$ for all tasks if $n < \varphi \cdot m'$, where $m' \leq m$ is the number of processors not running master threads and φ is the golden ratio, and equal to 4 else.

3. TASK MODEL

We consider a taskset τ of n periodic parallel real-time Directed Acyclic Graph (DAG) tasks run on a system of m identical processors. The taskset τ is represented by $\{\tau_1, ..., \tau_n\}$. Each DAG task τ_i, where $1 \leq i \leq n$, is a periodic implicit-deadline graph which consists of a set of subtasks under precedence constraints that determine their execution flow. A DAG task τ_i is characterized by $(\{\tau_{i,j} | 1 \leq j \leq n_i\}, G_i, O_i, D_i)$, where the first parameter represents the set of subtasks of τ_i and n_i is their number, G_i is the set of directed relations between these subtasks, O_i is the offset of the DAG and D_i is τ_i's relative deadline. Since each DAG task has an implicit deadline, its period T_i (interval time between its successive jobs) is the same as its deadline $T_i = D_i$.

Let $\tau_{i,j}$ denotes the j^{th} subtask of the set of subtasks forming the DAG task τ_i, where $1 \leq j \leq n_i$. Each subtask $\tau_{i,j}$ is a single-threaded sequential task which is characterized by a worst-case execution time (WCET) $C_{i,j}$. All subtasks of a DAG share the same deadline and period of the DAG. The total WCET C_i of DAG τ_i is defined as the sum of WCETs

[3]Partitioned scheduling algorithm FBB-FFD stands for Fisher Baruah Baker-First Fit Decreasing, then Deadline Monotonic priority assignment is used.

[4]A resource augmentation bound ν for scheduling algorithm A is the processor speed up factor, such that any feasible taskset on unit-speed processor, is guaranteed to be schedulable with A on processor of speed at least ν.

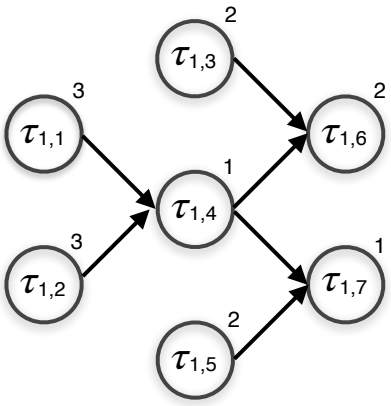


Figure 1: An example of a DAG task τ_1 which consists of 7 subtasks. The number on the upper right corner of each subtask represents its WCET and the arrows represent their precedence constraints.

of its subtasks, where $C_i = \sum_{j=1}^{n_i} C_{i,j}$. Let U_i denote the utilization of τ_i where $U_i = \frac{C_i}{T_i}$, and its density is denoted by $\delta_i = \frac{C_i}{min(D_i, T_i)}$. For an implicit-deadline task, whose deadline is equal to its period, the density is the same as the utilization, while the density of a constrained-deadline task becomes $\delta_i = \frac{C_i}{D_i}$.

The set of directed relations between the subtasks of DAG τ_i is denoted by G_i and it defines their dependencies. A directed relation $G_i(j,k)$ between subtasks $\tau_{i,j}$ and $\tau_{i,k}$ means that $\tau_{i,j}$ is a predecessor of $\tau_{i,k}$, and the latter subtask have to wait for all of its predecessors to complete their execution before it can start its own. An example of a DAG task is shown in Figure 1, in which τ_1 consists of 7 subtasks. Precedence constraints are represented by directed arrows between subtasks. A source subtask is a subtask with no predecessors such as $\tau_{1,1}$. Respectively, an ending subtask is the one without any successors such as $\tau_{1,7}$.

Based on the structure of DAG tasks, the critical path of DAG τ_i is defined as the longest sequential execution path in the DAG when it executes on a virtual platform composed of an infinite number of processors. Its length L_i is the minimum response time of the DAG. A subtask that is part of the critical path is referred to as a critical subtask, while non-critical subtasks are executed in parallel with the critical ones.

A DAG task is said to be feasible if the subtasks of all of its jobs respect its deadline. A taskset τ is deemed unfeasible when scheduled using any scheduling algorithm on m unit-speed processors if, at least, one of the following conditions is false:

$$\forall \tau_i \in \tau, \quad L_i \leq D_i$$
$$U(\tau) = \sum_{i=1}^{n} U_i = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq m$$

4. DAG STRETCHING ALGORITHM

Based on our task model, subtasks of a DAG inherit the absolute deadline of the DAG task. However, only source subtasks are activated by the activation of the DAG task, while their successors are activated dynamically based on the completion time of their predecessors. During scheduling process, the subtasks of a DAG have to execute within the activation interval of the DAG (between the release time and the absolute deadline of the DAG), while the exact activation time of subtasks is unknown prior to scheduling process. As a result, multiprocessor scheduling of DAG tasks is more complicated than the scheduling of independent sequential tasks. Therefore, scheduling process of DAG tasks can be simplified by avoiding the dynamic activation of subtasks by assigning them intermediate offsets and deadlines. Hereby, we propose a stretching algorithm for parallel DAG tasks which transforms each DAG task into a set of independent periodic constrained-deadline threads, that can be scheduled using any scheduling algorithm of multiprocessors. With our DAG stretching algorithm, the scheduling of a DAG task is done based on the timing parameters of the threads, and a task is deemed feasible if all of its jobs respect their assigned intermediate deadlines. Our DAG stretching algorithm chooses intermediate offset and deadline for each subtask in a DAG task. Before explaining the concept of our algorithm, we start by analyzing the DAG model and identifying its characteristics that led to the stretching algorithm.

Assume that the critical subtasks of DAG τ_i are combined together in a sequential master thread τ_i^{master} whose execution time is equal to L_i. Based on the task model, τ_i can be seen as a master thread with the longest sequential execution time among all paths in the DAG from a source to an ending subtask, while the remaining paths of the DAG execute in parallel with it. For example, DAG task τ_1 from Figure 1 consists of 7 subtasks and 6 execution paths: $\{\{\tau_{1,1}, \tau_{1,4}, \tau_{1,6}\}\ \{\tau_{1,1}, \tau_{1,4}, \tau_{1,7}\}\ \{\tau_{1,2}, \tau_{1,4}, \tau_{1,6}\}\ \{\tau_{1,2}, \tau_{1,4}, \tau_{1,7}\}\ \{\tau_{1,3}, \tau_{1,6}\}\ \{\tau_{1,5}, \tau_{1,7}\}\}$. The master thread of τ_1 is either $\{\tau_{1,1}, \tau_{1,4}, \tau_{1,6}\}$ or $\{\tau_{1,2}, \tau_{1,4}, \tau_{1,6}\}$ with a length $L_1 = 6$. Since both paths are identical w.r.t. their length, we consider the former to be the master thread of the DAG arbitrarily. Thus, DAG task τ_1 needs at least 6 time units in order to execute on unit-speed processors. Let Sl_i denote the positive slack available to DAG τ_i when it is scheduled exclusively without interference from other tasks. Sl_i is given by:

$$Sl_i = D_i - L_i \tag{1}$$

The slack time of the DAG can be seen as a slack time of its master thread since it is the DAG's longest path. Back to the previous example, if we assume that τ_1 had a deadline $D_1 = 10$, then its slack time would be $Sl_1 = 4$. In order to avoid dependencies between the subtasks of a DAG task, we propose to use the stretching algorithm which is summarized as follows:

DAG Stretching Algorithm.

Our stretching algorithm fills the slack time of a DAG task by adding fractions of non-critical subtasks to its master thread until it is stretched up to DAG's deadline. As a result, the remaining non-critical subtasks (if any) will be forced to execute in parallel with the stretched master thread within a fixed activation interval.

The objective of our stretching algorithm is to avoid the

parallel structure of DAG tasks by executing their subtasks as sequentially as possible. Hence, for each DAG task $\tau_i \in \tau$, if τ_i fits completely on a single processor ($C_i \leq D_i$), then the stretching algorithm transforms it into a single thread which contains all of its subtasks and forces them to execute sequentially. Otherwise, the algorithm fully stretches the master thread of τ_i up to its deadline D_i. As a result, the stretching algorithm generates a fully-stretched master thread and a collection of independent threads with intermediate offsets and deadlines that execute in parallel with the master thread. The offsets and deadlines of threads are important for the scheduling process and also to maintain the precedence constraints of DAG τ_i. It is worth noting that the stretching algorithm is a pre-step to the scheduling process, and any master thread with utilization equal to 1, is assigned a dedicated processor by the scheduler. Other parallel threads are scheduled on the remaining processors of the system using any multiprocessor scheduling algorithm. In this paper, we choose global EDF scheduling algorithm.

For clarity reasons, the reader is advised to refer to the example in Figures 2 and 3 so as to have a better understanding of our DAG stretching algorithm. More details about the example are provided at the end this section.

In order to perform our stretching algorithm, we propose to transform a DAG task into Multi-Threaded Segment (MTS) representation. This is a basic transformation for DAGs that maintains precedence constraints between their subtasks and leads to an easier scheduling analysis.

The Multi-Threaded Segment (MTS) Representation

A multi-threaded Segment (MTS) task τ'_i consists of a sequence of segments. Each segment is composed of a number of parallel threads with the same WCET. Let S_i denote the set of parallel segments of τ'_i, and s_i denote the total number of segments in S_i. Each segment $S_{i,j}$, $1 \leq j \leq s_i$, consists of $m_{i,j}$ parallel threads. All threads of segment $S_{i,j}$ have the same WCET which is denoted by $e_{i,j}$. The number of threads and their WCETs vary from one segment to another, and each segment contains at least one thread.

In this work, MTS task τ'_i is not a new task but a representation of DAG task $\tau_i \in \tau$. This is done by considering that all subtasks of τ_i are executed as soon as possible on a virtual platform having an infinite number of processors. As a result, each subtask executes under precedence constraints and it does not suffer from interference of other subtasks in the system. For a DAG task τ_i, its source subtasks are activated at the activation time of DAG τ_i. Then, successor subtasks are activated as soon as their predecessors have completed their execution. A segment in the MTS task is defined whenever a subtask completes its execution. Thus, threads of the same segment have the same WCET. It is worth noticing that a subtask in DAG τ_i may be divided into two or more threads executing in successive segments from its MTS representation τ'_i.

Figure 2 shows the MTS representation τ'_1 of DAG τ_1 from Figure 1. Knowing that DAG τ_1 is released at time $t = 0$ and has a deadline $D_1 = 10$, source subtasks $\{\tau_{1,1}, \tau_{1,2}, \tau_{1,3}, \tau_{1,5}\}$ are activated at $t = 0$. At $t = 2$, both subtasks $\tau_{1,3}$ and $\tau_{1,5}$ complete their execution and the first segment $S_{1,1}$ is defined. Their successors (subtasks $\tau_{1,6}$ & $\tau_{1,7}$) have to wait for all of their predecessors to complete their execution before they can start theirs. At time $t = 3$, subtasks $\tau_{1,1}$ and $\tau_{1,2}$ complete their execution, subtask $\tau_{1,4}$ starts its own

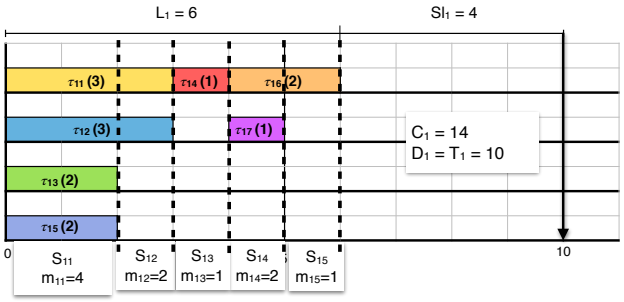


Figure 2: The Multi-Threaded Segment (MTS) representation of DAG task τ_1 from Figure 1.

and segment $S_{1,2}$ is defined. Finally, at time $t = 4$, subtasks $\tau_{1,6}$ and $\tau_{1,7}$ are activated, and segments $S_{1,3}$, $S_{1,4}$ and $S_{1,5}$ are defined at times 4, 5 and 6 respectively. The resulting MTS task τ'_1 is represented as a sequence of 5 segments. The number and the WCET of threads of each segment are identified. Subtask $\tau_{1,1}$ is an example of a subtask that is spread on multiple segments. It is divided into two threads executing in segments $S_{1,1}$ and $S_{1,2}$.

Finally, the MTS task τ'_i shares the same deadline D_i and period T_i of DAG task τ_i. Hence, Equation 1 regarding the slack Sl_i of τ_i remains correct for the MTS task τ'_i. Also, the critical path length L_i and the total WCET C_i of the original DAG task τ_i is the same for τ'_i, but they are calculated differently based on the parameters of the MTS task τ'_i:

$$L_i = \sum_{j=1}^{s_i} e_{i,j}$$

$$C_i = \sum_{j=1}^{s_i} m_{i,j} * e_{i,j}$$

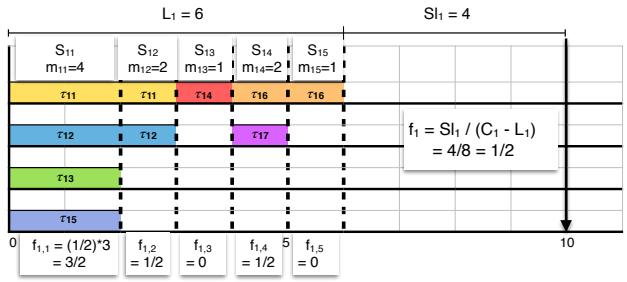
For a given DAG set τ, the stretching algorithm is applied to τ'_i for each DAG task $\tau_i \in \tau$. The algorithm is explained in detail in the remainder of this section.

The DAG Stretching Algorithm

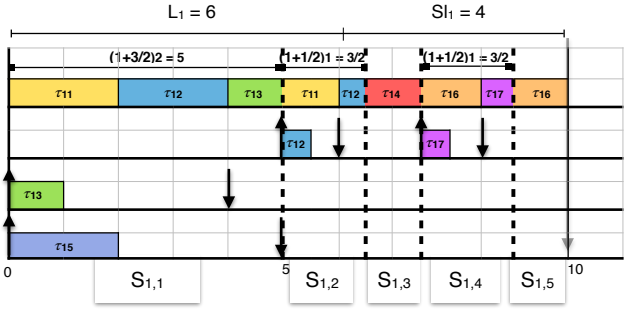
As stated earlier, the DAG stretching technique executes the DAGs as sequentially as possible. By doing so, parallel structure of DAGs is removed and dependencies between the subtasks are replaced by intermediate offsets and deadlines. The stretching algorithm is done based on the MTS representation τ'_i rather than the DAG structure τ_i. Hence, threads of parallel segments of τ'_i are used to fill its slack Sl_i and intermediate offsets and deadlines are assigned to the segments. As a result, a subtask $\tau_{i,j}$ of DAG τ_i is said to be feasible if all of its threads in its respective τ'_i have respected their intermediate deadlines.

The stretching algorithm is divided into two main cases, based on the timing parameters of the parallel DAG τ_i and the relation between its total WCET C_i and its deadline D_i:

- if $U_i \leq 1$, then DAG τ_i is considered as a sequential task because its total WCET C_i can be contained entirely within its deadline D_i. As a result, the stretching algorithm transforms its subtasks into a single master thread τ_i^{master} in which all of the subtasks execute sequentially. If U_i is equal to 1, then the master thread



(a) The MTS representation of task τ_1 from Figure 1 showing the slack factors (f_i&$f_{i,j}$) derived using the stretching algorithm.



(b) The Stretch algorithm is applied on the MTS task. The result of the stretching are the master thread and the constrained-deadline parallel threads.

Figure 3: Example of the DAG stretching algorithm applied on DAG τ_1 from Figure 1.

will be fully-stretched ($C_i^{master} = D_i^{master}$) and the scheduler dedicates an entire processor for it. Otherwise, the master thread is scheduled as a sequential implicit-deadline periodic task using any multiprocessor scheduling algorithm. In this work, we use global EDF scheduling algorithm.

In this case, the stretching is done directly on the original DAG task τ_i and it is not necessary to use its MTS representation τ_i'.

- if $U_i > 1$, then it is impossible for DAG τ_i to be transformed completely into a single master thread, since its total WCET C_i is larger than its deadline D_i. Hence, the stretching algorithm cannot avoid parallelism in τ_i. Applying the stretching algorithm on τ_i' generates a fully-stretched master thread τ_i^{master} as a result of stretching the critical path of τ_i' up to its deadline, in addition to a set of constrained-deadline threads $\{\tau_i^{cd}\}$ with intermediate offsets, deadlines and period equal to the original T_i. The importance of intermediate offsets and deadlines of each thread is to keep the dependencies between the subtasks of the original task, and prevent threads of the same subtask from executing in parallel. As in the previous case, the master thread, which has a utilization $U_i^{master} = 1$, will be assigned its own processor, while threads of the latter set will be scheduled independently on the remaining processors of the system.

As stated earlier, the stretching algorithm aims at stretching the critical path of task τ_i' up to its deadline and create a fully-stretched master thread. In order to do so, parallel non-critical threads of τ_i' are used to fill the slack Sl_i of the task uniformly, and all segments of τ_i' with more than one thread participate. To achieve a uniform filling of the slack, equal fractions from these threads are added to the master thread. In this case, where the stretching algorithm targets only parallel tasks with $C_i > D_i$, the total WCET C_i of τ_i' without its critical path length L_i definitely exceeds its deadline D_i. We define a distribution factor f_i of each execution unit in $(C_i - L_i)$ that has to be added to the slack Sl_i as follows:

$$f_i = \frac{Sl_i}{C_i - L_i} = \frac{D_i - L_i}{C_i - L_i} \leq \frac{D_i}{C_i} < 1 \quad (2)$$

In order to clarify the meaning of f_i, let us suppose that each non-critical execution unit in task τ_i' is divided into two parts, first part of length equal to f_i is added to the master thread and $(1 - f_i)$ execute in parallel as a constrained-deadline thread. The total execution time added from non-critical threads to the master thread is equal to Sl_i. However, it is unpractical to use f_i in this way, and force each execution unit to be divided into two parts. So, the filling of the slack is based on the execution requirement of each segment in the task and not on the threads directly. We fill the slack with the maximum number of entire threads from each segment. Thus, at most one thread from each segment will be used to fill the slack partially. From the MTS representation τ_i', each segment $S_{i,j} \in S_i \in \tau_i'$ has $m_{i,j}$ parallel threads, among them there is a critical thread. Hence, the total non-critical worst-case execution time of segment is equal to $(e_{i,j} * (m_{i,j} - 1))$. Based on the definition of f_i, we conclude that a total of $(f_i * e_{i,j}(m_{i,j} - 1))$ time units from segment $S_{i,j}$ will be added to the master thread.

Now, we want to identify how many threads of each segment $S_{i,j}$ are added to the master thread based on the total execution time of the segment. Knowing that the threads of segment $S_{i,j}$ have equal WCETs denoted by $e_{i,j}$, which is equivalent to the shortest sequential execution length of the segment, we can identify how many entire threads are added to the master thread. Let $f_{i,j}$ denote the number of threads from segment $S_{i,j}$ to be added to the master thread:

$$f_{i,j} = \frac{f_i * e_{i,j}(m_{i,j} - 1)}{e_{i,j}} = f_i * (m_{i,j} - 1) \quad (3)$$

According to this, each segment $S_{i,j}$ adds $\lfloor f_{i,j} \rfloor$ entire threads and a fraction of a thread of length $(f_{i,j} - \lfloor f_{i,j} \rfloor)e_{i,j}$ to the master thread. As a result, the slack Sl_i of task τ_i' is filled completely and a fully-stretch master thread τ_i' with U_i^{master} equal to 1 is generated. We conclude that each segment $S_{i,j}$ adds in total $(1+f_{i,j})$ threads to the master thread (including the critical thread), while the remaining threads of the segment execute in parallel with the master thread. Hence, each segment $S_{i,j}$ has an intermediate deadline $D_{i,j}$ calculated as follows:

$$D_{i,j} = (1 + f_{i,j}) * e_{i,j} \quad (4)$$

From the definition of the MTS model, segments of a task τ_i' execute sequentially and when one segment completes its execution, its successor starts its own. Hence, at any time

$t \geq 0$, there is only one active segment from each task τ'_i. According to this, we can define an intermediate offset $O_{i,j}$ for each segment $S_{i,j} \in \tau'_i$ based on the intermediate deadlines of the segments, where:

$$\forall S_{i,j} : j > 1 \;\; \rightarrow \;\; O_{i,j} = \sum_{k=1}^{j-1} D_{i,k}$$

and $O_{i,1} = 0$ (since τ_i has no offset).

After applying the stretching algorithm, a segment $S_{i,j}$ of τ'_i comprises of:

- a thread $\tau_{i,j}^{master}$ which is part of the master thread τ_i^{master} of τ'_i. It has a WCET of $D_{i,j}$ and a deadline of $D_{i,j}$.
- $(m_{i,j} - \lfloor f_{i,j} \rfloor - 2)$ parallel constrained-deadline threads with a WCET of $e_{i,j}$ and a deadline $D_{i,j}$.
- one remaining thread with WCET of $(1 + \lfloor f_{i,j} \rfloor - f_{i,j})e_{i,j}$ and a constrained-deadline $(1+\lfloor f_{i,j} \rfloor)e_{i,j}$. The remaining WCET is added to the master thread so as to fill its slack. We can notice here that the remaining thread has a shorter deadline than the other entire remaining threads from the same segment. This is done so as to force the first fraction of the thread to finish its execution before the second fraction in the master thread starts its own. As a result, both fractions of the same threads are forced to execute sequentially.

For each segment $S_{i,j}$, the total number of threads in a segment (including the partial thread), which are not added to the master thread, is given by:

$$q_{i,j} = m_{i,j} - \lfloor f_{i,j} \rfloor - 1 \tag{5}$$

According to this, $q_{i,j}$ independent threads from segments of each task $\tau'_i \in \tau$ are scheduled using any multiprocessor scheduling algorithm, while the fully-stretched master threads are assigned their own processors. After applying the stretching algorithm on each task $\tau'_i \in \tau$, each task generates at most one fully-stretched master thread. Hence, the total number of these master threads, respectively the number of their dedicated processors, cannot exceed n, which is the total number of tasks in the original DAG set τ. As a result, the constrained-deadline tasks are scheduled on the remaining processors of the system which are referred to as m'. The relation between the remaining processors m' and the total number of processors in the system m is given as follows:

$$m' \geq m - n \tag{6}$$

Algorithm 1 shows the DAG stretching algorithm. It shows its steps and the generated threads after the stretching. Now, we present an example of the stretching algorithm applied on the DAG task τ_1 from Figure 1.

Algorithm 1 DAG Stretching Algorithm

Require: $\tau_i(n_i, \{1 \leq j \leq n_i | \tau_{i,j}\}, G_i, D_i)$
Ensure: $\tau_i^{master}, \{\tau_i^{cd}\}$
  if $C_i \leq T_i$ **then** ▷ Execute τ'_i sequential as a single task
    for $S_{i,j} \in S_i$ **do**
      for $k = 1$ to $m_{i,j}$ **do**
        $\tau_i^{master} \leftarrow \tau_i^{master} \cup \tau_{i,j}^k : e_{i,j}$
      end for
    end for
  else
    $\tau'_i \leftarrow DagToMTS(\tau_i)$
    ▷ Represent a DAG task as a multi-threaded segment task.
    $f_i \leftarrow \frac{Sl_i}{C_i - L_i}$
    for $S_{i,j} \in S_i$ **do**
      $f_{i,j} \leftarrow f_i * (m_{i,j} - 1)$
      $q_{i,j} \leftarrow \lfloor f_{i,j} \rfloor + 1$
      for $k = 1$ to $q_{i,j}$ **do** ▷ $q_{i,j}$ threads of segment $S_{i,j}$ are added to the master thread
        $\tau_i^{master} \leftarrow \tau_i^{master} \cup \tau_{i,j}^k : e_{i,j}$
      end for
      $\tau_i^{master} \leftarrow \tau_i^{master} \cup \tau_{i,j}^{q_{i,j}+1} : (f_{i,j} - \lfloor f_{i,j} \rfloor) * e_{i,j}$
      $\tau_{i,j}^{tmp} \leftarrow \tau_{i,j}^{q_{i,j}+1} : (1 + \lfloor f_{i,j} \rfloor - f_{i,j}) * e_{i,j}$
      $D_{i,j}^{tmp} \leftarrow (1 + \lfloor f_{i,j} \rfloor) * e_{i,j}$
      if $j == 1$ **then**
        $O_{i,j}^{tmp} \leftarrow O_i$
      else
        $O_{i,j}^{tmp} \leftarrow O_{i,(j-1)} + D_{i,(j-1)}$
      end if
      $\{\tau_i^{cd}\} \leftarrow \{\tau_i^{cd}\} \cup \tau_{i,j}^{tmp}$
      for $k = (q_{i,j} + 2)$ to $m_{i,j}$ **do**
        $\tau_{i,j}^{tmp} \leftarrow \tau_{i,j}^k : e_{i,j}$
        $D_{i,j}^{tmp} \leftarrow (1 + f_{i,j}) * e_{i,j}$
        if $j == 1$ **then**
          $O_{i,j}^{tmp} \leftarrow O_i$
        else
          $O_{i,j}^{tmp} \leftarrow O_{i,(j-1)} + D_{i,(j-1)}$
        end if
        $\{\tau_i^{cd}\} \leftarrow \{\tau_i^{cd}\} \cup \tau_{i,j}^{tmp}$
      end for
    end for
  end if

  $(O_i^{master} \leftarrow O_i$
  $(D_i^{master} \leftarrow D_i$
  return $(\tau_i^{master}, \{\tau_i^{cd}\})$

Example: Stretching Algorithm

We present in this section an example of the stretching algorithm applied to the DAG task of Figure 1. The algorithm is summarized in three main steps which are shown in Figures 2 & 3. We assume that task τ_1 is a periodic implicit-deadline DAG with $C_1 = 14$ and a deadline $D_1 = 10$. Since its utilization $U_1 = 1.4 > 1$, then τ_1 has to be represented by τ'_1 which is shown in Figure 2. The subtasks of τ_1 execute as soon as possible while considering an execution platform of infinite number of processors, and the only blocking effect on a subtask is due to its predecessors.

In Figure 2, the MTS task τ'_1 is identified. It consists of 5 segments. Segment $S_{1,1}$ has 4 threads with $e_{1,1} = 2$, while the remaining threads have WCET equal to 1. Segments $S_{1,2}$ & $S_{1,4}$ have two threads and segments $S_{1,3}$ & $S_{1,5}$ have a single thread. The length of the critical path is $L_1 = 6$ and its deadline $D_1 = 10$. Based on Equation 1, the slack Sl_1 is equal to 4. The total WCET of τ_1 is equal to 14 and the non-critical execution time (while excluding the critical path) is equal to 8. According to Equation 2, the distribution factor of τ'_1 is equal to $f_1 = \frac{4}{8} = \frac{1}{2}$. For each segment $S_{1,j}$, its distribution factor $f_{1,j}$ depends on the number of its parallel

threads, where $f_{1,1} = \frac{3}{2}$, $f_{1,2} = f_{1,4} = \frac{1}{2}$ and $f_{1,3} = f_{1,5} = 0$, as shown in Figure 3(a). It is worth noticing that segments with a single thread have a distribution factor equal to zero, because their single thread is a critical one and it is already included in the master thread.

Based on the values of the segment distribution factor of each segment, we can identify how many threads to be added to the master thread from each segment. As described earlier, $(1 + f_{i,j})$ threads from segment $S_{i,j}$ are used to fill the master thread. Figure 3(b) shows the stretching of task τ_1 and its final result. We can notice that a total of $\frac{5}{2}$ threads from segment $S_{1,1}$ are added to the master thread where each thread has a length of 2. This is equivalent to 2 entire threads and a half. Knowing that the original DAG task τ_1 has no offset, then the first segment $S_{1,1}$ has no offset and its execution interval length is defined as $\frac{5}{2} * 2 = 5$. The third thread of segment $S_{1,1}$ (denoted by $\tau_{1,3}$ in Figure 3(b)) is divided into two halves, one is added to the master thread and the other is an independent constrained-deadline thread with a deadline $D_{1,3} = 4$. The fourth remaining thread of the segment has a deadline equal to the deadline of the segment which is 5. The difference between both threads is that thread $\tau_{1,3}$ has to finish earlier than the deadline of the segment so as to give its other part (the one added to the master thread) enough time to execute. Both parts have to execute sequentially and never in parallel, and the intermediate deadline of the remaining thread forces the sequential execution of the entire subtask. By applying the same calculations on all of the segments, we obtain the result in Figure 3(b). Regarding segments $S_{1,2}$ & $S_{1,4}$, each one adds $\frac{1}{2}$ thread to the master thread. This means that there is only one partial thread left to execute in parallel with the master thread from each segment.

As shown in the example in Figure 3(b), the stretching algorithm generates a master thread τ_1^{master} with WCET and deadline equal to 10, and 4 constrained-deadline threads identified as follows: $\{\tau_1^{cd}\} = \{\{\tau_{1,3} : (0, 1, 4, 10)\}, \{\tau_{1,5} : (0, 2, 5, 10)\}, \{\tau_{1,2} : (5, 0.5, 1, 10)\}, \{\tau_{1,7} : (7.5, 0.5, 1, 10)\}\}$, where each thread is identified by its intermediate offset, WCET, intermediate deadline and period. These threads are scheduled as independent constrained-deadline threads on m' processors. In this example , we consider that taskset τ contains a single DAG τ_1 which execute on a system of 2 identical processors. Then the master thread occupies a processor for itself, and the other processor is used for the scheduling of the parallel threads, which means $m' = 1$.

LEMMA 1. *A DAG set τ that is schedulable using algorithm $\mathcal{A}$ on a system of m processors, remains schedulable after stretching, i.e., the DAG stretching algorithm does not affect its schedulability.*

PROOF. As described above, the stretching algorithm assigns intermediate offsets and deadlines for threads of each DAG in τ, without changing their original offsets and deadlines. Intermediate offsets and deadlines maintain precedence constraints of the original DAG, which means that if a τ'_{str} is schedulable while respecting the intermediate deadlines of the threads, then the original DAG taskset τ must be schedulable while respecting the DAGs' deadlines. □

5. RESOURCE AUGMENTATION BOUND ANALYSIS

In this section, we analyze the performance of DAG stretching algorithm of parallel tasks by calculating its resource augmentation bound (speedup factor) when global EDF is used to schedule the non fully-stretched threads generated from the stretching algorithm. Speedup factor is a common performance metric for real-time scheduling algorithms, that gives indications about their performance w.r.t. the performance of an optimal algorithm.

DEFINITION 1 (FROM [8]). *For a given taskset τ that is feasible on m unit-speed processors using an optimal scheduler, it is schedulable using $\mathcal{A}$ scheduling algorithm on m processors that are ν times faster. The minimum speedup factor of processors speed is the resource augmentation bound (or simply the speedup factor) of scheduler $\mathcal{A}$.*

Let τ'' be the thread set generated after applying our stretching algorithm on every DAG task in τ. It contains all threads generated from the stretching algorithm except for the fully-stretched master threads (including the constrained-deadline parallel threads and the implicit-deadline master threads). We prove that global EDF scheduling of τ'' when executing on m' processors, has a speedup factor of $\frac{3+\sqrt{5}}{2}$ for all tasks with $n < \varphi \cdot m'$, where $\varphi = \frac{1+\sqrt{5}}{2}$, the remaining number of processors is denoted by $m' \geq m - n$, where m is the original number of processors in the system and n is the number of tasks in the set τ This implies that if a taskset τ'' is feasible on m' unit-speed processors, then it is schedulable using global EDF on m' processors with speed of $\frac{3+\sqrt{5}}{2}$ as fast as the original. Otherwise, the speedup factor is equal to 4 if we use the decomposition algorithm from [12]. The fully-stretched master threads are assigned their own processors, while the remaining parallel threads are scheduled using global EDF on the remaining processors.

Our speedup factor analysis is based on the following sufficient scheduling condition of global EDF scheduling algorithm.

THEOREM 1 (FROM [1]). *Any constrained-deadline sporadic sequential task set τ with total density $\delta^{sum}(\tau)$ and maximum density $\delta^{max}(\tau)$ is schedulable using preemptive global EDF policy on m unit-speed processors if*

$$\delta^{sum}(\tau) \leq m - (m-1)\delta^{max}(\tau)$$

THEOREM 2. *If constrained-deadline taskset τ'' of parallel threads, with $n < \varphi \cdot m'$, where $\varphi = \frac{1+\sqrt{5}}{2}$ (the golden ratio) and n is the number of tasks in DAG taskset τ, is feasible on m' unit-speed processors, then τ'' is schedulable using global EDF on m' processors with speed at least $\frac{3+\sqrt{5}}{2}$.*

PROOF. Based on Theorem 1, a taskset of constrained-deadline parallel threads τ'' that is generated after applying the DAG stretching algorithm is schedulable on the remaining processors m' of the system if:

$$\delta^{sum}(\tau'') \leq m' - (m'-1)\delta^{max}(\tau'')$$

We start by calculating the maximum density $\delta^{max}(\tau'')$ of the threads of τ''. As stated earlier, τ'' consists of two types of threads: implicit-deadline master threads (not fully-stretched) from DAG tasks with utilization less than 1, and a collection of constrained-deadline threads $\{\tau^{cd}\}$ from each DAG task with a utilization higher than 1. We will consider both cases while calculating $\delta^{max}(\tau'')$:

- case 1: for an implicit-deadline DAG task τ_i with $U_i < 1$, it is completely transformed into a single master thread τ_i^{master} with the same timing parameters of the original DAG τ_i. Based on the assumption that τ_i has implicit deadline, then its deadline D_i^{master} is equal to its period T_i^{master}, and respectively, $\delta_i^{master} = U_i^{master} < 1$.

- case 2: for an implicit-deadline DAG task τ_i with $U_i > 1$, it is transformed into a set of constrained-deadline parallel threads with intermediate offsets and deadlines. From the structure of multi-threaded segment tasks, only one segment is active at every time instant t. Each segment $S_{i,j} \in S_i \in \tau'$, with $m_{i,j} > 1$, has at most two types of parallel threads, entire and partial threads. Parallel threads of segment $S_{i,j}$ (other than the one in the master thread) have a maximum density $\delta_{i,j}^{max}$ of:

$$\begin{aligned}\delta_{i,j}^{max} &= \max\{\frac{e_{i,j}}{(1+f_{i,j})e_{i,j}}, \frac{(1+\lfloor f_{i,j}\rfloor - f_{i,j})e_{i,j}}{(1+\lfloor f_{i,j}\rfloor)e_{i,j}}\} \\ &= \max\{\frac{1}{1+f_{i,j}}, \frac{1-(f_{i,j}-\lfloor f_{i,j}\rfloor)}{(1+f_{i,j})-(f_{i,j}-\lfloor f_{i,j}\rfloor)}\} \\ \forall\ m_{i,j} > 1 \ &\Rightarrow\ \delta_{i,j}^{max} = \frac{1}{1+f_{i,j}} \end{aligned} \tag{7}$$

From the above equation, the maximum density of threads of segment $S_{i,j}$ in τ_i depends on the segment distribution factor $f_{i,j}$. From Equation 3, $f_{i,j}$ depends on the number of threads in this segment. Hence, the segment with a small number of threads has a higher density. We can conclude that the highest density of a thread in task τ_i occurs when the segment has the smallest possible number of threads ($m_{i,j} \geq 2$).

$$\begin{aligned}\delta_i^{max} &= \frac{1}{1+(f_i*(2-1))} \quad \longrightarrow \quad \text{from 3} \\ &= \frac{1}{1+f_i}\end{aligned} \tag{8}$$

From Equation 2, we can derive the following relation:

$$\begin{aligned}&\forall f_i \geq 0 \rightarrow \delta_i^{max} = \frac{1}{1+f_i} \text{ and } f_i \geq \frac{D_i - L_i}{C_i} \\ &\rightarrow \frac{1}{1+f_i} \leq \frac{1}{1+\frac{D_i-L_i}{C_i}} \leq \frac{C_i}{C_i+D_i-L_i} \leq 1 \\ &\text{(Since } L_i \leq D_i\text{)}.\end{aligned}$$

From both cases, the maximum density $\delta^{max}(\tau'')$ of all threads in τ'' is:

$$\begin{aligned}\delta^{max}(\tau'') &= \max_{\tau_i'' \in \tau''}\{\delta_i^{master}, \delta_i^{max}\} \\ &\leq 1\end{aligned} \tag{9}$$

It is worth noticing here that DAG tasks with utilization equal to one are not considered in the previous cases, because the DAG stretching algorithm transforms them into fully-stretched master threads that are assigned their own executing processors. Hence, they are not included in the resource augmentation bound of global EDF scheduling.

In order to calculate the total density of thread set τ'', we consider that at time t, only a single segment from each task with utilization higher than 1 can be active in addition to the master threads of tasks with utilization < 1. We calculate the density of a task τ_i'', with $U_i > 1$, by considering the threads of the segment with the highest density. Let δ_i^{sum} be the sum of densities of such threads in a certain segment in a task τ_i:

$$\begin{aligned}\delta_i^{sum} &\leq \frac{1}{1+f_{i,j}} * (m_{i,j} - 1 - \lfloor f_{i,j}\rfloor) \quad \longrightarrow \quad \lfloor f_{i,j}\rfloor \geq 0 \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{and from Eq. (5)} \\ &\leq \frac{m_{i,j}-1}{f_i*(m_{i,j}-1)} \\ &\leq \frac{1}{f_i} \\ &\leq \frac{C_i - L_i}{D_i - L_i} \leq \frac{C_i}{D_i - L_i} \quad \longrightarrow \quad \text{from Eq. (2)}\end{aligned} \tag{10}$$

Now, we consider the second case where task τ_i'' has a utilization less than 1 and is transformed into an implicit-deadline non-fully stretched master thread. The total density of the task is denoted $\delta_{max}^{master} = \frac{C_i}{D_i} \leq \delta_i^{sum}$.

For taskset τ'', let $\delta^{sum}(\tau'')$ be the sum of densities of every DAG task in the original taskset τ:

$$\delta^{sum}(\tau'') \leq \sum_{\tau_i \in \tau} \frac{C_i}{D_i - L_i} \tag{11}$$

In order to calculate the speedup factor, we consider that taskset τ'' executes on m' processors with a minimum speed ν, where $\nu > 1$. Increasing the speed of processors affects the execution parameters of task τ_i such as C_i and L_i.

The critical path length L_i of any task τ_i has a necessary feasibility condition:

$$\forall \tau_i \ :\ 1 \leq i \leq n \quad \longrightarrow \quad L_i \leq D_i \tag{12}$$

Otherwise, task τ_i is not feasible on a unit-speed processors.

On a processor that is ν times faster, the critical path length L_i^ν is given by:

$$\forall \tau_i \ :\ 1 \leq i \leq n \quad \longrightarrow \quad L_i^\nu = \frac{L_i}{\nu} \leq \frac{D_i}{\nu} \tag{13}$$

The same notation is applied for the total execution time C_i^ν of task τ_i when it is run on processors ν times faster:

$$\forall \tau_i \ :\ 1 \leq i \leq n \quad \longrightarrow \quad C_i^\nu = \frac{C_i}{\nu} \tag{14}$$

The total density of taskset $\delta^{sum,\nu}$ on speed-ν processors is:

$$\delta^{sum,\nu}(\tau'') \leq \sum_{\tau_i \in \tau} \frac{C_i^\nu}{D_i - L_i^\nu} \tag{15}$$

From Equation 13,

$$\forall 1 \leq i \leq n \quad \longrightarrow \quad L_i^\nu \leq \frac{D_i}{\nu} \Rightarrow (D_i - L_i^\nu) \geq D_i(1 - \frac{1}{\nu})$$

Using this result,

$$\begin{aligned}\delta^{sum,\nu}(\tau'') &\leq \sum_{\tau_i \in \tau} \frac{C_i/\nu}{D_i(1-\frac{1}{\nu})} \\ &\leq \frac{1}{\nu-1}\sum_{\tau_i \in \tau}\frac{C_i}{D_i} \\ &\leq \frac{m}{\nu-1}\end{aligned}$$

From Equation 9, the maximum density $\delta^{max,\nu}(\tau'')$ of taskset τ'' on speed-ν processors is:

$$\delta^{max,\nu}(\tau'') = \frac{\delta^{max}}{\nu} \leq \frac{1}{\nu} \tag{16}$$

Therefore, taskset τ'' is schedulable under preemptive global EDF on m' speed-ν processors if:

$$\frac{m}{\nu - 1} \leq \frac{m' + n}{\nu - 1} \leq m' - (m' - 1)\frac{1}{\nu}$$

$$\frac{m'}{\nu - 1} + \frac{n}{\nu - 1} \leq m' - \frac{m'}{\nu} + \frac{1}{\nu}$$

$$\frac{1}{\nu - 1} + \frac{1}{\nu} - \frac{1}{m'\nu} \leq \frac{1}{\nu - 1} + \frac{1}{\nu} \leq 1 - \frac{n}{(\nu - 1)m'}$$

$$\frac{2\nu - 1}{\nu^2 - \nu} \leq 1 - \frac{n}{m'(\nu - 1)}$$

In order to have a positive value for the right hand side of the inequality, we consider the following:

$$0 < \frac{n}{m'(\nu - 1)} < 1 \Rightarrow m' > \frac{n}{\nu - 1}$$

$$\frac{2\nu - 1}{\nu^2 - \nu} < 1$$

$$0 < \nu^2 - 3\nu + 1$$

$$\rightarrow \nu = \frac{3 + \sqrt{5}}{2} \quad \text{for } n < \varphi \cdot m' \text{ , where } \varphi = \frac{1 + \sqrt{5}}{2}$$

where φ is the golden ratio.

This concludes the proof of the speedup factor. □

6. SIMULATION

In this section, we provide simulation results for global EDF scheduling with stretched tasksets. The objective of simulation experiments is to test the performance of our DAG stretching algorithm when used prior to the scheduling process of DAG tasks. Also, we aim to compare it with another existing DAG scheduling technique found in the state-of-the-art [12]. To our knowledge, the decomposition algorithm proposed in [12] is the only other DAG scheduling algorithm using a DAG transformation technique. Therefore, we simulated the process of DAG scheduling for both algorithms, stretching and decomposition.

In all experiments, we generated 100000 synchronous tasksets. We used a random generator of DAG tasksets which uses UUnifast-Discard method from [5]. In order to generate a taskset, the generator requires two inputs, the total utilization of the taskset and the number of tasks. Then it distributes the utilization on its tasks uniformly. Based on the assigned utilization of each task, we derive its remaining timing parameters (WCET and deadline). In order to limit the hyperperiod (least common multiple of periods) of tasksets, we used the limitation technique from [7] in which we define a matrix containing all possibilities of period values of tasks. We defined the simulation interval of tasksets to be as large as the hyperperiod. Also, the number of subtasks of each DAG task and their dependencies are generated randomly.

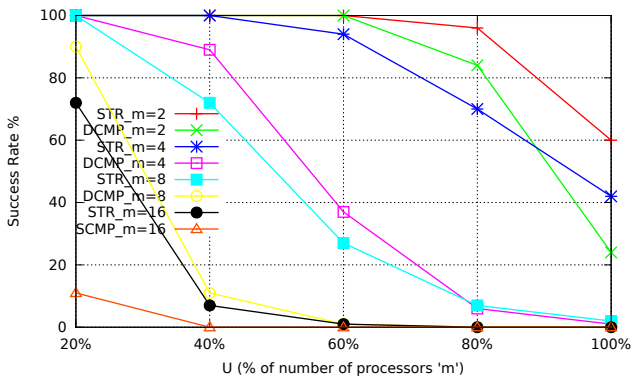


Figure 4: Comparison results of global EDF scheduling simulation between the DAG stretching algorithm and decomposition algorithm from [12]

Comparison between DAG stretching and decomposition algorithms

The first results of simulation are provided in Figure 4. They are obtained by varying two parameters, the number of unit-speed processors m on which tasksets are executing, and the percentage of taskset utilization, where the maximum utilization of a taskset is equal to m. Figure 4 shows the comparison results between our DAG stretching algorithm (denoted 'STR' in the figure) and the DAG decomposition algorithm[5] (denoted by 'DCMP' in the figure). Similarly, The decomposition algorithm transforms each dependent DAG task into a set of independent sequential threads with intermediate offsets and deadlines by distributing the slack of the DAG on its segments. Figure 4 shows the percentage of successfully schedulable tasksets when global EDF is used for m processors (from 2 to 16). The x-axis of the figure represents the percentage of taskset utilization w.r.t. the maximum utilization which is equal to m. The percentage values are between 20% and 100%. For each taskset, the scheduling process is simulated for both cases, when the taskset is stretched (STR) or decomposed (DCMP).

From our simulations, we notice that the performances of our stretching algorithm is better than the decomposition algorithm for the scheduling of the DAG tasks. In all the experiments, the percentage of schedulable tasksets when stretched are higher than the decomposed ones. Also, we can notice that the percentage of the schedulable tasksets decreases when the number of processors increases.

Varying the speed of processors

Here, we analyze the effect of increasing the speed of processors w.r.t. the schedulability of stretched DAG tasks. Based on the speedup factor of our stretching approach, the schedulability of stretched DAG tasks increases when the speed of processors increases and feasible tasksets become schedulable using global EDF when the speed $\nu > \frac{3+\sqrt{5}}{2}$. In these experiments, we increased the speed of processors from $\nu = 1$ up to 4 by steps of 1, while varying the number of processor and taskset utilization as in the above simulations.

The results shown in Figure 5 are done by varying the

[5]For more details about the decomposition algorithm, please refer to the original paper in [12].

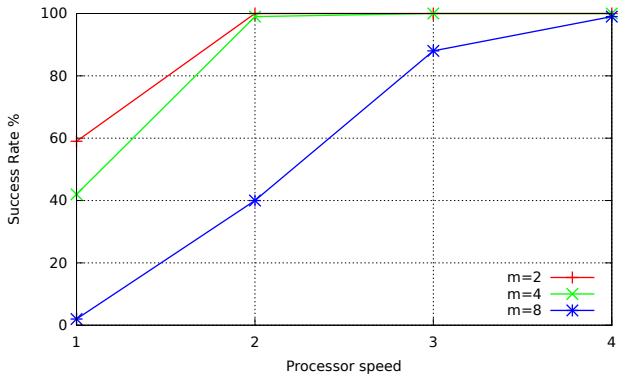


Figure 5: Simulation results show the effect of processor speed on the schedulability of the stretched tasksets.

speed of processors on which tasksets are executed. For each m from 2 to 8, we simulate global EDF scheduling of tasksets of maximum utilization (equal to m). The choice of taskset utilization is done because these tasksets have the smallest success ratio in the results from Figure 4. As shown in Figure 5, schedulability of stretched tasksets increases when the speed of processors increases. In the case of $m = 2$ and $m = 4$, a success ratio of almost 100% is achieved when the speed of processors $\nu \geq 2$, while it needs processors speed at least equal to 4 for $m = 8$.

The simulation results provided in this section show the performances of our DAG stretching approach when compared to the decomposition scheduling algorithm of DAGs. Although simulation experiments based on random generation of tasksets cannot be used as an accurate performance metric of algorithms in real-time systems, the results are considered as an indication regarding the average behavior of studied scheduling algorithms.

7. CONCLUSIONS

In this paper, we presented a stretching algorithm for parallel periodic real-time DAG tasks on homogeneous multiprocessor systems. The stretching algorithm is an extension to the one in [9] which targeted Fork-join task model, a special case of DAG model. Similarly, our DAG stretching algorithm aimed at removing dependencies between subtasks of DAG tasks and stretching their critical path (master thread) up to its deadline. In addition to the master thread which is assigned its own executing processor, a set of independent threads with intermediate offsets and deadlines are generated and scheduled using global EDF scheduling algorithm.

Then, we proved a speedup factor of $\frac{3+\sqrt{5}}{2}$ for global EDF scheduling algorithm for stretched threads if $n < \varphi \cdot m'$, where n is the total number of tasks in the taskset, m' is the remaining number of processors after taking off the processors dedicated for the fully-stretched master threads and φ is the golden ratio whose value is $\frac{1+\sqrt{5}}{2}$ and 4 else.

Finally, we compared the performance of our stretching algorithm with another scheduling technique for DAGs from the state-of-art using DAG decomposition. The simulation results show that our DAG stretching algorithm has better average performances.

8. REFERENCES

[1] S. Baruah. Techniques for Multiprocessor Global Schedulability Analysis. *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*, pages 119–128, Dec. 2007.

[2] S. K. Baruah, V. Bonifaciy, A. Marchetti-Spaccamela, L. Stougie, and A. Wiese. A Generalized Parallel Task Model for Recurrent Real-time Processes. In *Proceedings of the 33rd IEEE Real-Time Systems Symposium (RTSS)*, pages 63–72. IEEE Computer Society, Dec. 2012.

[3] V. Bonifaci, A. Marchetti-spaccamela, S. Stiller, and A. Wiese. Feasibility Analysis in the Sporadic DAG Task Model. In *25th euromicro Conference on Real-Time Systems (ECRTS'13)*, 2013.

[4] H. Chetto, M. Silly, and T. Bouchentouf. Dynamic scheduling of real-time tasks under precedence constraints. *Real-Time Systems*, 2(3):181–194, 1990.

[5] R. Davis and A. Burns. Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. *Real-Time Systems*, 47(1):1–40, 2011.

[6] R. I. Davis and A. Burns. A survey of hard real-time scheduling algorithms and schedulability analysis techniques for multiprocessor systems. *ACM Computing surveys*, pages 1 – 44, 2011.

[7] J. Goossens and C. Macq. Limitation of the Hyper-Period in Real-Time Periodic Task Set Generation. In *Proceedings of the 9th International Conference on Real-Time Systems (RTS)*, pages 133–148, Mar. 2001.

[8] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance [scheduling problems]. In *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*, pages 214–221, 1995.

[9] K. Lakshmanan, S. Kato, and R. (Raj) Rajkumar. Scheduling Parallel Real-Time Tasks on Multi-core Processors. In *Proceedings of the 31st IEEE Real-Time Systems Symposium (RTSS)*, pages 259–268. IEEE Computer Society, 2010.

[10] A. J. Li, K. Agrawal, C. Lu, and C. Gill. Analysis of Global EDF for Parallel Tasks. In *Euromicro Conference on Real-Time Systems ECRTS*, number 314, 2013.

[11] M. Qamhieh, F. Fauberteau, L. George, and S. Midonnet. Global edf scheduling of directed acyclic graphs on multiprocessor systems. In *Proceedings of the 21st International Conference on Real-Time Networks and Systems*, RTNS '13, pages 287–296, New York, NY, USA, 2013. ACM.

[12] A. Saifullah, D. Ferry, K. Agrawal, C. Lu, and C. Gill. Parallel real-time scheduling of DAGs. In *IEEE Transactions on Parallel and Distributed Systems*, 2014. accepted.