Simulation-Based Evaluations of DAG Scheduling in Hard Real-time Multiprocessor Systems

Manar Qamhieh, Serge Midonnet

Simulation-Based Evaluations of DAG Scheduling in Hard Real-time Multiprocessor Systems

Manar Qamhieh
Université Paris-Est
LIGM UMR CNRS 8049
UPEM, France
manar.qamhiehsalous@univ-mlv.fr

Serge Midonnet
Université Paris-Est
LIGM UMR CNRS 8049
UPEM, France
serge.midonnet@univ-mlv.fr

ABSTRACT

The scheduling of parallel real-time tasks on multiprocessor systems is more complicated than the one of independent sequential tasks, specially for the Directed Acyclic Graph (DAG) model. The complexity is due to the structure of DAG tasks and the precedence constraints between their subtasks. The trivial DAG scheduling approach is to directly apply common real-time scheduling algorithms on DAGs despite their lack of compatibility with the parallel model. Another scheduling approach, which is called the stretching method, aims at transforming each parallel DAG task in the set into a collection of independent sequential threads that are easier to be scheduled.

In this paper, we are interested in analyzing global preemptive scheduling of DAGs using both approaches by showing that they are not comparable when associated with Deadline Monotonic (DM) and Earliest Deadline First (EDF) scheduling algorithms. Then we use extensive simulations to evaluate their schedulability performance. To this end, we use our simulation tool *YARTISS* to generate random DAG tasks with many parameter variations so as to guarantee reliable experimental results.[1]

Categories and Subject Descriptors

J.7 [**Computers in Other Systems**]: Real time
F.1.2 [**Theory of Computation**]: Computation by Abstract Devices—*Parallelism and concurrency*
I.6.m [**Simulation and Modeling**]: Miscellaneous

General Terms

Experimentation

Keywords

Real-time systems, hard real-time scheduling, parallel tasks, Directed Acyclic Graphs, global preemptive scheduling, DM and EDF scheduling algorithms.

1. INTRODUCTION

Chip manufacturers are tending to build multi-processors and multi-core processors as a solution to overcome the physical constrains of the manufacturing process, such as chip's size and heating. Many practical examples of shifting towards multiprocessors can be found nowadays, such as the *Intel Xeon* processor with up to 18 cores and the 72-core processor from Tilera. Because of that, parallel programming has gained a higher importance although it has been used for many years. The concept of parallel programming is to write a code that can be executed simultaneously on different processors. Usually these programs are harder to be written than sequential ones, since it is necessary to keep the parallel partitions independent in order to execute them correctly on different processors at the same time.

From practical implementation's point of view, there exist certain libraries, APIs and models created specially for parallel programming like OpenMP[1] and POSIX threads[2]. In this paper, we are interested in a particular family of parallelism called the inter-subtask parallelism, in which a parallel task consists of a collection of subtasks under precedence constraints. The most general model is the Directed Acyclic Graph (DAG) model, which is our task model in this paper.

In hard real-time systems, the correctness of results depends on the respect of certain timing parameters assigned to tasks. We consider that each task generates an unlimited number of jobs (copies) based on its timing parameters. A scheduler is responsible of choosing which job to execute on which processor at all times. The problem of hard real-time scheduling on uniprocessor and multiprocessor systems have been studied thoroughly for many years, and many researches and scheduling algorithms have been proposed for such platforms [9].

The extension of real-time scheduling w.r.t. parallel dependent tasks is not trivial. The real-time scheduler has to take into consideration the internal dependencies of tasks when it schedules them. For a given DAG task, the execution order of its subtasks is not known prior to the scheduling process, i.e., a subtask can start its execution when all of its predecessors have finished their own, and a subtask can execute either sequentially or in parallel with its siblings based on the decisions of the scheduler. To solve this problem, there are two DAG scheduling approaches that are presented in the state-of-the-art: the Direct scheduling and the Model Transformation approaches. The Direct Scheduling approach represents the parallel execution form of DAG tasks in which the scheduling process is done based on the internal dependencies of DAGs. The Model Transformation approach aims at converting the dependent parallel model of DAG tasks into independent sequential model so as to simplify its scheduling. The latter approach represents the

sequential execution form of DAG tasks since it converts parallel tasks into sequential threads. Both approaches are used to determine the execution order to subtasks and they are associated with other real-time scheduling algorithms, such as Earliest Deadline First (EDF) and Deadline Monotonic (DM).

DAG scheduling approaches have been studied recently in many researches and schedulability analyses were provided. But they have never been compared to each other w.r.t. schedulability performance. In this paper, we show that both approaches are incomparable, i.e., there exist task sets that are schedulable using one scheduling approach while they are unschedulable when the other one is used, and vice versa. Due to this incomparability, we analyze their performance by performing extensive simulations. In this work, we consider global preemptive scheduling of periodic implicit-deadline DAGs on identical processors when EDF and DM algorithms are used. To the best of our knowledge, there is no similar analysis presented previously in the state-of-the-art.

The remainder of this paper is organized as follows. Section 2 presents related works w.r.t. to the problem of scheduling real-time DAG tasks on multiprocessor systems. In Section 3, we introduce our task model. In Section 4, we describe in details the two DAG scheduling approaches and we prove their incomparability. Simulation results are provided in Section 5 to evaluate the performance of these scheduling algorithms with brief description of our simulation tool *YARTISS*. Finally, Section 6 concludes this work.

2. RELATED WORK

The scheduling of dependent real-time tasks of different models has been studied on both uniprocessor and multiprocessor systems. In uniprocessor systems which consists of a single processing unit, a dependent parallel task is transformed into a sequential chain, and subtasks are assigned local timing parameters which are used in the scheduling process (e.g., [15, 25, 30, 16, 28]). The DAG model on uniprocessor systems was considered in [7]. The authors proposed an algorithm to modify the timing parameters of DAGs (by adding intermediate offsets and deadlines to subtasks) in order to get rid of their internal dependencies.

In the case of multiprocessor systems, preemptive scheduling of jobs with precedence constraints has been proved NP-Hard in the strong sense in [29]. However, many researches targeted the scheduling of parallel tasks of different models, as in [14, 10, 17, 11, 8]. Regarding the DAG model and as mentioned above, there are two approaches for its scheduling in hard real-time systems. The Direct Scheduling was introduced in [3], in which a taskset of a single sporadic DAG is scheduled on multiprocessor systems using EDF algorithm. The authors provided polynomial and pseudo-polynomial schedulability tests for EDF scheduling algorithm.

Later on, many works have considered the scheduling of multiple DAG tasks on multiprocessor systems, e.g., [5, 20, 21]. In these researches, the scheduling analyses are performed based on the general timing parameters of DAG tasks, such as their total Worst-Case Execution Time (WCET), deadline and critical path length. In [22], the internal dependencies of DAGs and the execution order of their subtasks were included in the analysis of global EDF scheduling. A Subtask-Level scheduling of DAG tasks was proposed in [24], in which scheduling decisions were taken based on the local timing parameters of subtasks rather than the timing parameters of DAGs.

The Model Transformation approach was introduced in [18], in which a stretching algorithm of Fork-Join (FJ) task model was proposed. The stretching algorithm avoided the parallel structure of FJ tasks by executing them as sequentially as possible. Then a DAG Stretching (DAG-Str) algorithm was proposed in [23] to consider the general model DAG tasks. In the stretching algorithms, the DAG tasks are stretched up to their deadline and their dependent subtasks are transformed into a set of independent sequential threads. Intermediate offsets and deadlines are assigned to threads so as to determine their execution interval. In this paper, we analyze the schedulability performance of the DAG-Str algorithm when compared to Direct Scheduling approach of DAGs.

The Decomposition algorithm[26, 27] is another example of the Model Transformation approach for DAG tasks. It aims at distributing the slack time of each DAG task, which is the difference between its relative deadline and its minimum sequential execution time, on its subtasks. Accordingly, the subtasks are assigned intermediate offsets and deadlines which guarantee their independent execution.

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. A DAG task τ_i is characterized by $(n_i, \{1 \leq j \leq n_i | \tau_{i,j}\}, G_i, D_i)$, where n_i is the number of its subtasks, the second parameter represents the set of subtasks of τ_i, G_i is the set of directed relations between these subtasks 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 $\tau_{i,j}$ denote 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 WCET $C_{i,j}$. All subtasks respect the absolute deadline and period of their respective DAG. The total WCET C_i of DAG τ_i is defined as the sum of WCETs of its subtasks, where $C_i = \sum_{j=1}^{n_i} C_{i,j}$. Let U_i denote the utilization of τ_i where $U_i = C_i/T_i$.

The directed relations G_i of DAG τ_i define the dependencies between its subtasks. A directed relation 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. Sibling subtasks refer to subtasks that have the same predecessor subtask and they can execute independently in parallel. Figure 1 shows an example of a DAG task τ_1 which consists of 7 subtasks. Precedence constraints are represented by directed arrows between subtasks. A source subtask is a subtask with no predecessors, e.g., subtasks $\tau_{1,1}$ and $\tau_{1,2}$. Respectively, a sink 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 com-

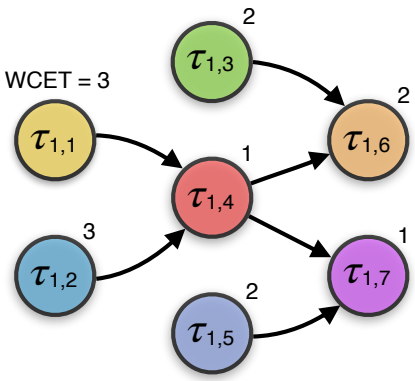


Figure 1. An example of a DAG task τ_1 which consists of 7 subtasks.

posed of unlimited 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 the remaining subtasks which execute in parallel with the critical ones. A slack time of a DAG τ_i is defined as the time difference between its relative deadline D_i and its critical path length L_i.

A DAG task is said to be feasible if subtasks of each job respect its absolute 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:

$$\forall \tau_i \in \tau, \quad L_i \leq D_i$$

$$U(\tau) = \sum_{i=1}^{n} U_i \leq m$$

4. DAG SCHEDULING APPROACHES: PARALLEL VS. STRETCHING

A real-time scheduler is responsible for choosing which job to execute on which processor at all times. The priorities of executed jobs are determined by using a scheduling algorithm. A scheduling algorithm is referred to as global if it allows job migrations between processors of the system, and an algorithm is referred to as preemptive if it allows higher priority jobs to interrupt the execution of lower priority ones. A taskset is said to be schedulable w.r.t. a specific scheduling algorithm if all jobs in the set complete their execution before their absolute deadlines when this scheduling algorithm is used.

Definition 1. *A scheduling algorithm is said to be optimal if it is able to schedule all possible feasible task sets.*

Despite that many optimal uniprocessor scheduling algorithms lose their optimality when applied on multiprocessor systems, these algorithms are widely used in many researches regarding the scheduling of parallel DAG tasks. In this paper, we consider two global preemptive scheduling algorithms, EDF from the fixed job priority assignment family and DM from fixed task family. Regarding EDF, it assigns priorities to jobs based on their absolute deadlines, i.e., the job with the earliest absolute deadline is assigned the highest priority. While DM assigns priorities to tasks based on their relative deadlines, in which the jobs with the earliest relative deadline, are assigned the highest priority. Hence, jobs of the same task are assigned the same priority.

In the case of independent sequential tasks, a scheduling algorithm assigns priorities to jobs based on the timing parameters of tasks, such as their deadline, period and slack time. However, the priority assignment of DAGs is more challenging. When a DAG job is assigned a priority by a scheduling algorithm, all of its subtasks inherit the same priority. Therefore, the scheduling algorithm has to determine the execution order of sibling subtasks of a DAG task, which are assigned the same priority and they execute independently. Hence, it is important to specify the default execution order of subtasks of DAGs before applying any scheduling algorithm by using the DAG scheduling approaches. In this section, we describe in details the Direct Scheduling approach (parallel execution form) and the DAG-Str algorithm (sequential execution form). Then we prove that both approaches are not comparable by providing scheduling examples.

4.1 Direct Scheduling Approach (Parallel Structure)

The Direct Scheduling approach defines the default parallel structure of DAG tasks. It supports the inter-subtask parallelism of DAGs in which subtasks are assumed to execute as soon as possible when they are activated. According to this approach, a scheduling algorithm does not permit there to be any time at which a processor is idle and there is a subtask ready to execute. Hence, if there are idle processors in the system, all ready sibling subtasks of a DAG task are allowed to execute in parallel.

According to the Direct Scheduling approach, each subtask $\tau_{i,j} \in \tau_i$ is characterized by an earliest offset $O_{i,j}$ and a relative deadline $D_{i,j}$. Both timing parameters determine maximum execution interval of the subtask and are calculated based on the precedence constraints of their DAG. The offset $O_{i,j}$ refers to the earliest activation time of any job of subtask $\tau_{i,j}$ when its DAG τ_i executes on unlimited number of processors. Hence, predecessor subtasks of $\tau_{i,j}$ execute without any delay or interruption. If a DAG task is activated at time t, then subtask $\tau_{i,j}$ cannot be activated before time instant $t + O_{i,j}$.

Similarly, a relative deadline $D_{i,j}$ refers to the latest finish time of subtask $\tau_{i,j}$ that guarantees no deadline miss of the DAG. If DAG task τ_i is released at time t and subtask $\tau_{i,j}$ fails in finishing its execution at most at $t + D_{i,j}$, then the remaining time before the deadline of the DAG at $t + D_i$ is not enough for the successors of $\tau_{i,j}$ to execute, even if unlimited number of processors is considered in the system[2].

Figure 2 shows an example of the parallel structure of DAG task τ_1 from Figure 1. We assume that DAG task τ_1 has a deadline equal to 10 and it consists of 7 subtasks. As shown in Figure 2, source subtasks $\{\tau_{1,1}, \tau_{1,2}, \tau_{1,3}, \tau_{1,5}\}$ are activated at time $t = 0$ and they are assumed to execute in parallel. Subtask $\tau_{1,4}$ is a successor of subtasks $\tau_{1,1}$ and $\tau_{1,2}$ and it has to wait for them both to finish execution before it starts its own, hence, its offset $O_{1,4}$ is equal to 3. Its successor subtasks $\tau_{1,6}$ and $\tau_{1,7}$ have offset equal to 4. When we consider that DAG task τ_1 has a deadline at $t = 10$, then sink subtasks $\tau_{1,6}$ and $\tau_{1,7}$ have to finish their execution at

[2]In both computations, we consider that subtask jobs execute up to their worst-case execution time.

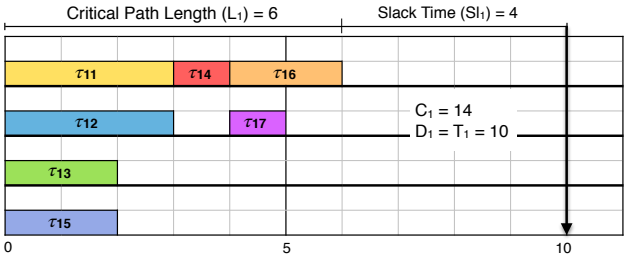


Figure 2. An example of parallel scheduling method of DAG task from Figure 1

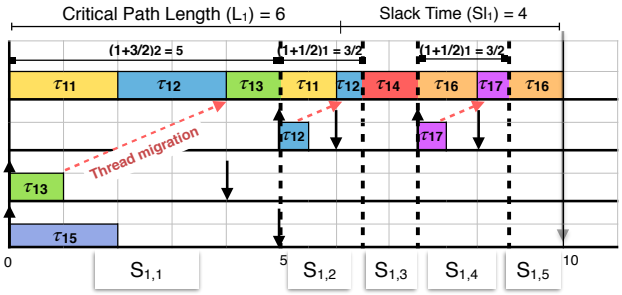


Figure 3. Example of stretching scheduling method for DAGs.

this time instant, and $D_{1,6} = D_{1,7} = 10$. However, subtask $\tau_{1,4}$ has to finish its execution no later than $t = 8$, so as to leave enough time for its successors to execute. Hence, the local deadline $D_{1,4}$ of subtask $\tau_{1,4}$ is equal to 5, and its maximum execution interval is equal to $[t + O_{1,4}, t + D_{1,4})$.

In general, Direct Scheduling approach maintains the general characteristics of DAG tasks. Real-time algorithms take scheduling decisions that are compatible with the parallel structure of DAGs and their internal dependencies. However, the main disadvantage of this approach is that higher priority DAGs can be greedy by occupying multiple processors of the system for the execution of their parallel subtasks, while delaying the execution of lower priority subtasks. This scenario may cause deadline miss in the set.

4.2 DAG Stretching Approach (Sequential Structure)

By using the Model Transformation approach, the scheduling problem of DAG tasks is simplified by avoiding their parallel structure. For example, the DAG Stretching (DAG-Str) algorithm converts each DAG task into a sequence of segments, each consists of independent sequential threads. Briefly, the concept of the DAG-Str algorithm is that the slack time of stretched DAG is filled by non-critical subtasks, and the critical path of the DAG is stretched up to its deadline. The algorithm forces certain subtasks of the DAG to execute sequentially to form a master thread whose utilization is equal to 1. The remaining threads are assigned intermediate offsets and deadlines so as to execute independently. In order to maintain the precedence constraints of stretched DAGs and to avoid that threads of the same subtasks execute in parallel, the generated threads are assigned intermediate offsets and deadlines. In the scheduling process after applying the stretching algorithm, the fully stretched master threads can be assigned dedicated processors since their utilization is equal to 1, and the independent threads can be scheduled using any multiprocessor scheduling algorithm.

Figure 3 shows an example of the DAG-Str algorithm when applied on DAG task τ_1 from Figure 1. The critical path of the DAG consists of subtasks $\tau_{1,1}$, $\tau_{1,4}$ and $\tau_{1,6}$. In order to fill the slack time of the DAG, whose length is equal to 4, parts of subtask $\tau_{1,2}$ and subtask $\tau_{1,7}$ are forced to execute sequentially within the critical path. After stretching, DAG task τ_1 is transformed into 5 sequential segments, where segment $S_{1,1}$ contains two independent threads, and each one of segments $S_{1,2}$ and $S_{1,4}$ contains one independent thread. Intermediate offsets (respectively intermediate deadlines) of threads are represented by upward pointing arrows (respectively downward pointing arrows) on Figure 3. Further details about the DAG-Str algorithm can be found in [23].

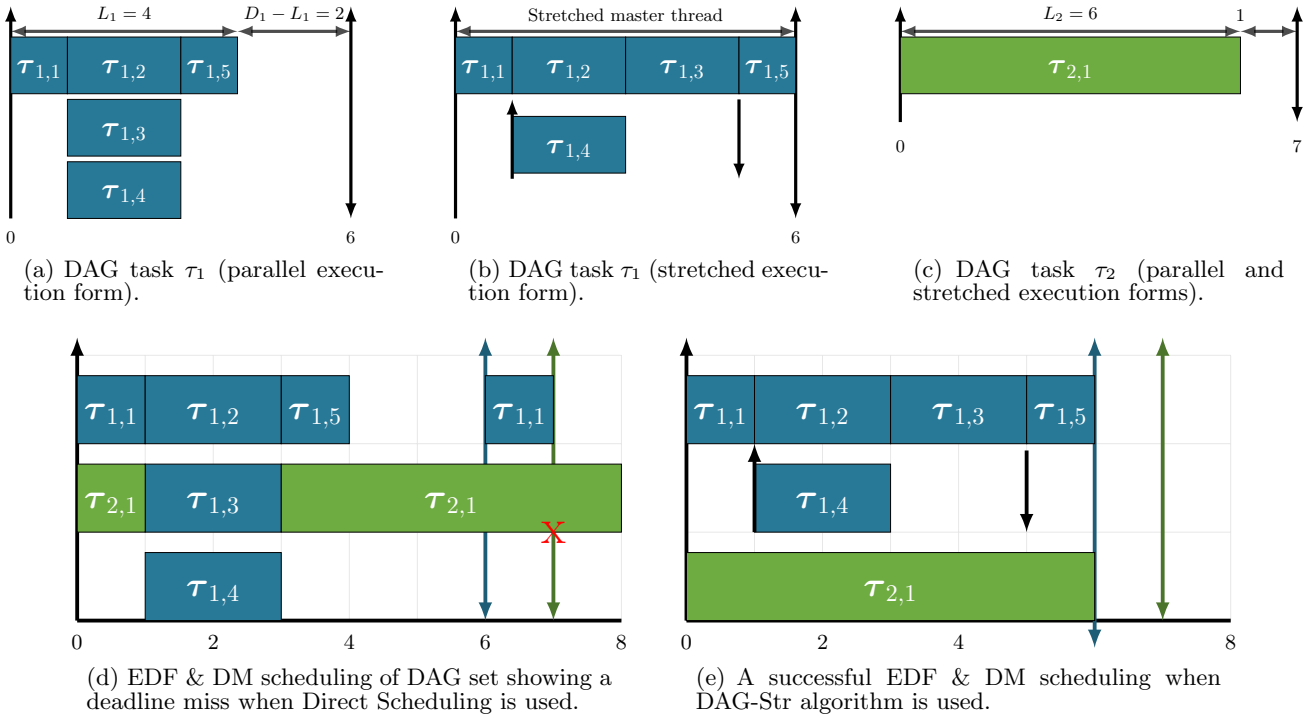
The Model Transformation approach simplifies the scheduling of DAG tasks by assigning intermediate offsets and deadlines to resulting threads, which are used by the scheduling algorithm. The internal dependencies of DAG tasks are eliminated at the expense of generality loss of model characteristics. In other words, the form of DAG tasks is altered because of the stretching algorithm, e.g., subtask $\tau_{1,2}$ from Figure 3 has to execute within the master thread and not in parallel even if there are available processors in the system.

4.3 Incomparability of Scheduling Approaches

In this section, we analyze the schedulability of parallel and stretching approaches of DAGs. We discuss the case of global preemptive scheduling when EDF and DM algorithms are used. In related researches, schedulability analyses of scheduling approaches were provided separately, and the results were never compared with each other. In this paper, we aim at analyzing the performance of these approaches w.r.t. schedulability of DAG sets.

In comparing DAG sets that are scheduled by two different approaches $\mathcal{A}$ and $\mathcal{B}$, there are three possibilities: (i) $\mathcal{A}$ dominates $\mathcal{B}$ if all schedulable DAG sets according to $\mathcal{B}$ belong to the set of schedulable DAG sets according to $\mathcal{A}$, (ii) $\mathcal{A}$ is equivalent to $\mathcal{B}$ if they schedule the exact same DAG sets and (iii) $\mathcal{A}$ and $\mathcal{B}$ are incomparable if there exist DAG sets that are schedulable according to $\mathcal{A}$ that are unschedulable according to $\mathcal{B}$ and vice versa.

By using two scheduling examples, we show that both DAG approaches are not comparable when associated with global preemptive EDF and DM algorithms. Then we evaluate their schedulability performance by extensive simulations. The incomparability of these scheduling approaches means that both approaches are acceptable for DAG scheduling and no one dominates the other w.r.t. DAG schedulability. To this end, we provide two examples to show the scheduling of a given DAG set on multiprocessor system using a global preemptive scheduling algorithm. In Example 1, we show that the DAG set is schedulable when the DAG-Str algorithm is used, while Direct Scheduling approach leads to a deadline miss. Then we show in Example 2 that Direct approach successfully schedules a DAG set which is unschedulable when DAG-Str algorithm is used. In both examples, we consider that EDF and DM algorithms are associated with the DAG scheduling approaches to assign priorities to



(a) DAG task τ_1 (parallel execution form).

(b) DAG task τ_1 (stretched execution form).

(c) DAG task τ_2 (parallel and stretched execution forms).

(d) EDF & DM scheduling of DAG set showing a deadline miss when Direct Scheduling is used.

(e) A successful EDF & DM scheduling when DAG-Str algorithm is used.

Figure 4. An example of scheduling incomparability in favor of DAG-Str when compared to Direct Scheduling.

their jobs.

Example 1: DAG-Str algorithm can outperform Direct Scheduling (Figure 4)

Task Set: In this example, we consider a DAG set τ that consists of two periodic implicit-deadline DAG tasks, where $\tau = \{\tau_1, \tau_2\}$. DAG τ_1 has 5 subtasks with a total WCET equal to 8 and a deadline equal to 6. The timing parameters of subtasks and the structure of DAG τ_1 are shown in Inset 4(a). It has a critical path length equal to 4 and a slack time equal to 2. Inset 4(a) shows the parallel form of τ_1 in which subtasks $\tau_{1,2}$, $\tau_{1,3}$ and $\tau_{1,4}$ execute in parallel. Inset 4(b) shows the sequential structure of DAG τ_1 when DAG-Str algorithm is applied. The critical path of the DAG ($\{\tau_{1,1}, \tau_{1,2}, \tau_{1,5}\}$) is stretched up to its deadline by forcing subtask $\tau_{1,3}$ to execute sequentially after subtask $\tau_{1,2}$. Subtask $\tau_{1,4}$ executes in parallel with an offset equal to 1 and a local relative deadline equal to 4.

DAG task τ_2 is shown in Inset 4(c). It consists of a single subtask $\tau_{2,1}$ which has a WCET equal to 6 and a deadline equal to 7. Since DAG τ_2 is a sequential task, there is no difference between its parallel and sequential execution forms. The utilization of the DAG set $U(\tau)$ is equal to $\frac{8}{6} + \frac{6}{7} = 2.19$ which means that it needs to execute on a platform of at least 3 unit-speed processors. In this example, we consider a platform of 3 processors.

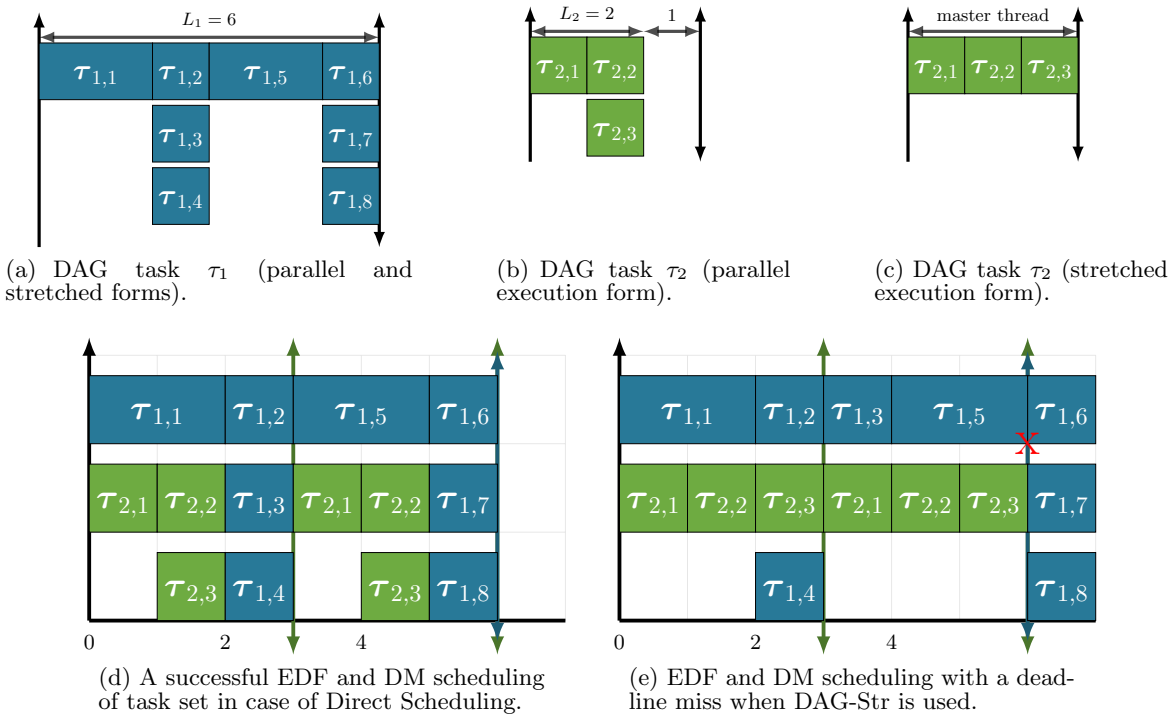
Priority Assignment: If DM scheduling algorithm is considered, then jobs of DAG τ_1 are assigned a higher priority than jobs of DAG τ_2 because τ_1 has a shorter relative deadline. In the case of EDF, if we consider a synchronous scenario in which both DAGs are released at time $t = 0$, then the first job of task τ_1 has an absolute deadline at $t = 6$ while the first job of τ_2 has an absolute deadline at $t = 7$. Hence, EDF assigns the first job of τ_1 a higher priority than the job of τ_2. According to this priority assignment, active jobs in the time interval $[0, 7)$ have the same priorities according to EDF and DM algorithms.

Direct Scheduling Approach: The considered scheduling is done based on the parallel execution form of DAGs while considering the priority assignment of EDF and DM. As shown in Inset 4(d), the first job of τ_1 executes without being interrupted since it has the highest priority. Its parallel subtasks $\{\tau_{1,2}, \tau_{1,3}, \tau_{1,4}\}$ occupy the 3 processors of the system for 2 time units in time interval $[1, 3)$. As a result, the execution of the first job of τ_2 is interrupted and it is delayed for 2 time units. Since its slack is equal to 1 time unit, a deadline miss occurs.

Model Transformation Approach: Inset 4(e) shows the scheduling of the same DAG set when the DAG-Str algorithm is used. Based on the structure of stretched DAG τ_1 from Inset 4(b), each job needs 2 processors so as to execute successfully at all times, because the stretching algorithm forces subtask $\tau_{1,4}$ to execute sequentially within the critical path. For any given scheduling algorithm, the DAG set is schedulable on 3 processors, because DAG task τ_1 occupies 2 processors and the remaining processor is dedicated to the sequential DAG task τ_2.

Conclusion: In the case of preemptive EDF and DM scheduling algorithms, there exists a DAG set that is schedulable on a multiprocessor system when the DAG-Str algorithm from the Model Transformation approach is used, while Direct Scheduling approach fails to schedule the same set.

Example 2: Direct Scheduling can outperform DAG-Str algorithm (Figure 5) Task Set: In this ex-



(a) DAG task τ_1 (parallel and stretched forms).

(b) DAG task τ_2 (parallel execution form).

(c) DAG task τ_2 (stretched execution form).

(d) A successful EDF and DM scheduling of task set in case of Direct Scheduling.

(e) EDF and DM scheduling with a deadline miss when DAG-Str is used.

Figure 5. An example of DAG scheduling incomparability in favor of Direct Scheduling when compared to DAG-Str algorithm.

ample, we consider a DAG set τ that consists of two periodic implicit-deadline DAG tasks $\{\tau_1, \tau_2\}$. DAG task τ_1 has a deadline equal to 6, and it consists of 8 subtasks and a total WCET equal to 10. The WCET of subtasks and the internal structure of the DAG are shown in Inset 5(a). The critical path length of DAG τ_1 is equal to 6 which is the same as its relative deadline. Thus, the DAG has no slack time and it cannot be stretched. In order to avoid a deadline miss, its subtasks have to execute without any delay or interruption. DAG task τ_2 consists of 3 subtasks, in which subtask $\tau_{2,1}$ is the source subtask of the DAG and subtasks $\{\tau_{2,2}, \tau_{2,3}\}$ are its successors, as shown in Inset 5(b). The default execution behavior of these subtasks is that both subtasks $\tau_{2,2}$ and $\tau_{2,3}$ execute in parallel. However, since DAG task τ_2 has a slack time equal to 1 time unit and its utilization is less than 1, then DAG-Str algorithm transforms DAG τ_2 into a sequential task in which all of its subtasks execute sequentially. The stretched form of DAG τ_2 is shown in Inset 5(c).

The total utilization $U(\tau)$ of the DAG set is equal to $(\frac{10}{6} + 1) = 2.66$. Hence, the task set requires an execution platform of at least 3 processors to be feasible.

Priority Assignment: If DM scheduling algorithm is used, then jobs of DAG τ_2 are assigned a higher priority than jobs of DAG τ_1 because τ_2 has a shorter relative deadline. In the case of EDF, if we consider a synchronous scenario in which DAGs are released at time $t = 0$, then the first job of task τ_2 has an absolute deadline at $t = 3$ while the first job of τ_1 has an absolute deadline at $t = 6$. As a result, EDF assigns the job from DAG τ_2 a higher priority. Regarding the second job of DAG τ_2, its absolute deadline is equal to the deadline of the first job of DAG τ_1, hence, priorities are assigned arbitrarily. In this example, we consider that the DAG with the shortest relative deadline is assigned higher priority as a tie breaking rule. According to this priority assignment, DAG jobs have the same priorities according to EDF and DM scheduling algorithms.

Direct Scheduling Approach: In Inset 5(d), we show the global preemptive scheduling of the DAG set on a system of 3 identical processors using the Direct Scheduling approach. First DAG jobs are activated at time $t = 0$ and the job of DAG τ_2 has the highest priority. Subtask $\tau_{2,1}$ executes in interval $[0, 1)$ and its successors $\tau_{2,2}$ and $\tau_{2,3}$ execute in parallel and occupy two processors of the system in $[1, 2)$. Similarly, the second job of DAG τ_2 executes in parallel with τ_1 in time interval $[3, 5)$. According to this scheduling, the first job of DAG task τ_1 executes without interruption and all of its subtasks execute without any delay. As Figure 5(d), the synchronous DAG set is schedulable using DM and EDF scheduling algorithms.

Model Transformation Approach: When DAG-Str algorithm is used, subtasks of DAG τ_2 are forced to execute sequentially as a sequential thread even if there are available processors in the system for subtasks to execute in parallel. The first job of τ_2 has a higher priority according to DM and EDF, then it occupies a single processor by itself in time interval $[0, 3)$, as shown in Inset 5(e). As a result, subtasks of the first job of DAG τ_1 are blocked during this time interval and one of them (subtask $\tau_{1,3}$ in the example) is delayed for 1 time unit and is forced to execute in time interval $[3, 4)$ instead of $[2, 3)$. DAG task τ_1 has no slack time, then a deadline miss happens as shown in the figure.

Conclusion: In the case of preemptive EDF and DM scheduling algorithms, there exists a DAG set that is schedulable on a multiprocessor system when Direct Scheduling

approach is used, while it is unschedulable when the DAG stretching algorithm is used. Based on Examples 1 and 2, both scheduling approaches are not comparable and no one dominates the other.

5. SIMULATION ANALYSIS

Based on the scheduling examples, we conclude that the DAG scheduling approaches are not comparable, and it is not clear which approach outperforms the other. In order to evaluate their schedulability performance, we evaluate them through extensive simulations of randomly-generated DAG tasks on multiple processors. The use of simulation-based evaluations is common in real-time analysis to give an indication regarding the performance of proposed algorithms. Simulation is used to check whether a set of tasks respects its temporal constraints when a specific algorithm is used, or to evaluate the efficiency of a new approach when compared with other algorithms from the state-of-the-art.

In real-time systems, there are many simulation tools that vary in their characteristics and features, e.g., MAST, Cheddar and FORTAS. However, many factors force researchers to implement their own simulation tools without depending on the existing ones, such as the lack of a standard simulator, the difficulty of extending an existing tool to include new features and models and the lack of documentation. In our case, we implemented a new simulation tool which contains the parallel dependent DAG model.

In this section, we present our simulation tool *YARTISS*[6], which is a free open-source simulation tool written in Java for real-time multiprocessor scheduling. We focused during its design on providing a generic simulation tool and an easy-to-use modular design in which new modules can be added easily without the need to decompress, edit nor recompile existing parts. We briefly describe the main features of the simulator w.r.t. random generation of the DAG model. Then we present the simulation results of DAG scheduling approaches and we evaluate their performance when global preemptive EDF and DM algorithms are used.

Task Model Generator

YARTISS offers an open architecture to facilitate the integration of different task models. Its current version contains two models, the first one is the independent sequential task model with energy parameters, in which a task is characterized by its WCET, its period and its relative deadline, in addition to its worst case energy consumption. The second model is the DAG model which belongs to the dependent parallel category.

Performing large-scale scheduling simulations requires a large data set of tasks. In order to avoid biased results and to ensure credibility, the used task sets should be randomly generated and their timing parameters should be varied sufficiently. YARTISS provides the ability to choose a task set generator which defines the various timing parameters of tasks and whether they are periodic/sporadic and implicit/-constrained/arbitrary deadline tasks. The default generator in YARTISS is based on the *UUniFast-Discard algorithm* [4] adapted to energy constraints and parallel tasks (utilization is greater than 1) coupled with a hyper-period limitation technique [13].

The UUnifast-Discard algorithm is used to uniformly distribute the system utilization on all tasks of the set with a complexity of $\mathcal{O}(n)$, where n is the number of tasks in the set. As shown in Algorithm 1, the UUniFast-Discard algorithm generates an array of n random task utilization, in which each element represents a task utilization U_i of $\tau_i \in \tau$, where $0 < U_i < U(\tau)$ for parallel tasks ($U_i \leq 1$ for sequential tasks) and $\sum_{i=1}^{n} U_i \leq U(\tau)$.

Algorithm 1 The UUniFast-Discard Algorithm (from [4])

Require: $U(\tau), n$
Ensure: $vectU$ ▷ An array of utilization of each task τ_i in the set τ.
  $sumU = U(\tau)$
  for $i = 1 : n-1$ **do**
    $nextSumU = sumU \times rand^{(1/(n-i))}$
    $vectU(i) = sumU - nextSumU$
    $sumU = nextSumU$
  end for
  $vectU(n) = SumU$

For each task τ_i, its utilization, which is equal to $U_i = \frac{C_i}{T_i}$, is used to compute the remaining timing parameters. We generate WCET and period values for each task based on its utilization. It is known that a periodic task set repeats its task arrival pattern after an interval called the hyper period. In order to verify the schedulability of a task set, it is necessary to determine the response time of each job on a period of length that may be slightly greater than the hyper period [19, 12]. The hyper period of a task set is calculated as the Least Common Multiple (LCM) of periods of tasks in the set. Hence, the value of LCM is affected by the increase of task periods in the set. In order to limit the length of the hyper period during task generation in YARTISS, we use the hyper-period limitation technique from [13]. The idea of the technique is to generate n periods $\{T_1, T_2, ..., T_n\}$ for each task in the set in a way to bound their resulting LCM. The algorithm uses a matrix $\mathcal{M}$ representing primes and their probabilistic distribution. A period is calculated as the multiple of random number from each line in the matrix. For example, if we consider a matrix $\mathcal{M}$ which consists of 5 primes $(2, 3, 5, 7, 11)$, its structure and probabilistic distribution are provided as follows:

$$\mathcal{M} = \begin{pmatrix} 1 & 2 & 2 & 4 & 4 & 4 & 8 & 16 & 16 \\ 1 & 3 & 3 & 9 & 9 & 9 & 27 & & \\ 1 & 5 & 5 & 25 & 25 & 25 & & & \\ 1 & 1 & 7 & 7 & 7 & 49 & & & \\ 1 & 1 & 1 & 11 & 11 & & & & \end{pmatrix}$$

The largest period, that can be possibly generated from $\mathcal{M}$, is equal to $(16 \times 27 \times 25 \times 49 \times 11 = 5821200)$, which represents the largest hyper period of the task set. Hence, by choosing the prime values of the matrix, we can limit the maximum hyper period of the generated task set, and respectively, the simulation interval.

Based on the UUniFast-Discard and the hyper-period limitation algorithms, the utilization and period of each task are derived. The deadline D_i is derived based on the type of generated task sets. In the case of implicit-deadlines, we consider that $D_i = T_i$. While the deadline of constrained-deadline tasks is less than or equal to the period, where $D_i \leq T_i$. Finally, there is no relation between the deadline and the period in the case of arbitrary-deadline tasks.

For a DAG task τ_i, the total WCET C_i can be calculated based on the utilization and period and it is strictly less than

the deadline ($1 \leq C_i \leq D_i$). The inter-subtask parallelism and the dependencies between the subtasks should be taken into consideration in the generation process. The DAG generator uses the UUniFast-Discard algorithm to determine the WCET of each subtask based on the total WCET of the DAG. The following parameters are necessary for the generation of subtasks and their precedence constraints:

- **Maximum number of subtasks (`MAX_SUBTASKS`)**: it is defined for each DAG set as an upper bound on the number of subtasks in the DAG. This value is important to determine the size of DAGs which affects the probability of its inter-subtask parallelism. Generally, DAG tasks, whose number of subtasks is large, tend to have more internal parallelism and more precedence constraints between their subtasks than smaller DAGs.

  Additionally, the minimum number of subtasks (`MIN_SUBTASKS`) is calculated so as to ensure a feasible generation of DAGs. Its value is calculated when we consider that each subtask $\tau_{i,j}$ in DAG τ_i has a WCET $C_{i,j}$ equal to its relative deadline D_i, which is the maximum execution time that can be assigned to any subtask so as to be feasible. Then, the `MIN_SUBTASKS` is equal to $\left\lceil \frac{C_i}{D_i} \right\rceil$. For each DAG task τ_i in the set, its number of subtasks n_i is equal to rand(`MIN_SUBTASKS`, `MAX_SUBTASKS`). If random generation of subtask timing parameters leads to `MIN_SUBTASKS` greater than `MAX_SUBTASKS`, then the generation process is repeated until this relation becomes true.

- The **WCET** $C_{i,j}$ of each subtask $\tau_{i,j}$ is calculated using the UUniFast-Discard algorithm, where the total WCET C_i of the DAG and the number of subtasks n_i are its inputs. We bound the value of $C_{i,j}$ of each subtask to ensure system feasibility by using the following $C_{i,j}^{max}$ and $C_{i,j}^{min}$ bounds:

  - Any sequential subtask of DAG τ_i cannot exceed the deadline of the DAG. Hence, $C_{i,j}^{max} = D_i$.
  - The minimum WCET $C_{i,j}^{min}$ is calculated when each subtask $\tau_{i,k}$ in the DAG, which is not assigned a WCET yet, is considered to have a WCET $C_{i,k}$ equal to $C_{i,k}^{max}$. This bound is necessary to ensure the feasibility of generated subtasks. For example, let us consider a DAG task τ_i with the following timing parameters: a total WCET $C_i = 6$, a relative deadline $D_i = 4$ and two subtasks $\{\tau_{i,1}, \tau_{i,2}\}$. If subtask $\tau_{i,1}$ is assigned a WCET equal to $C_{i,1} = 1$, then the remaining WCET available for subtask $\tau_{i,2}$ is equal to 5 which is greater than the deadline of the DAG and the subtask is not feasible on a unit-speed processor. Hence, $C_{i,1}^{min}$ has to be at least $(6 - 4) = 2$ time units to ensure feasibility.

- The **probability factor of directed relations** ρ between subtasks, where $0 < \rho < 1$. If ρ is close to 0, then the probability of creating a directed relation between any two subtasks in the DAG is large. This probability is reduced when ρ moves closer to 1. In order to get rid of cyclic dependencies between subtasks, we use a triangular matrix $\mathcal{R}$ whose entries of ones and zeros are generated randomly based on the probability factor ρ. For each DAG task τ_i, $\mathcal{R}$ is a square matrix of size n_i and all of its entries under the main diagonal are zeros. Also, we consider that the main diagonal entries are zeros so that a subtask does not have a precedence relation with itself. The remaining entries represent the precedence relations between subtasks and they are either zeros or ones. For DAG τ_i, if entry $\mathcal{R}_{j,k} = 1$, then we create a precedence relation from subtask $\tau_{i,j}$ to $\tau_{i,k}$. If it is zero, then there is no relation between these two subtasks. An example of the triangular matrix $\mathcal{R}$ of a DAG τ_i of 4 subtasks is considered as follows:

$$\mathcal{R} = \begin{array}{c} \\ \tau_{i,1} \\ \tau_{i,2} \\ \tau_{i,3} \\ \tau_{i,4} \end{array} \begin{array}{c} \begin{array}{cccc} \tau_{i,1} & \tau_{i,2} & \tau_{i,3} & \tau_{i,4} \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

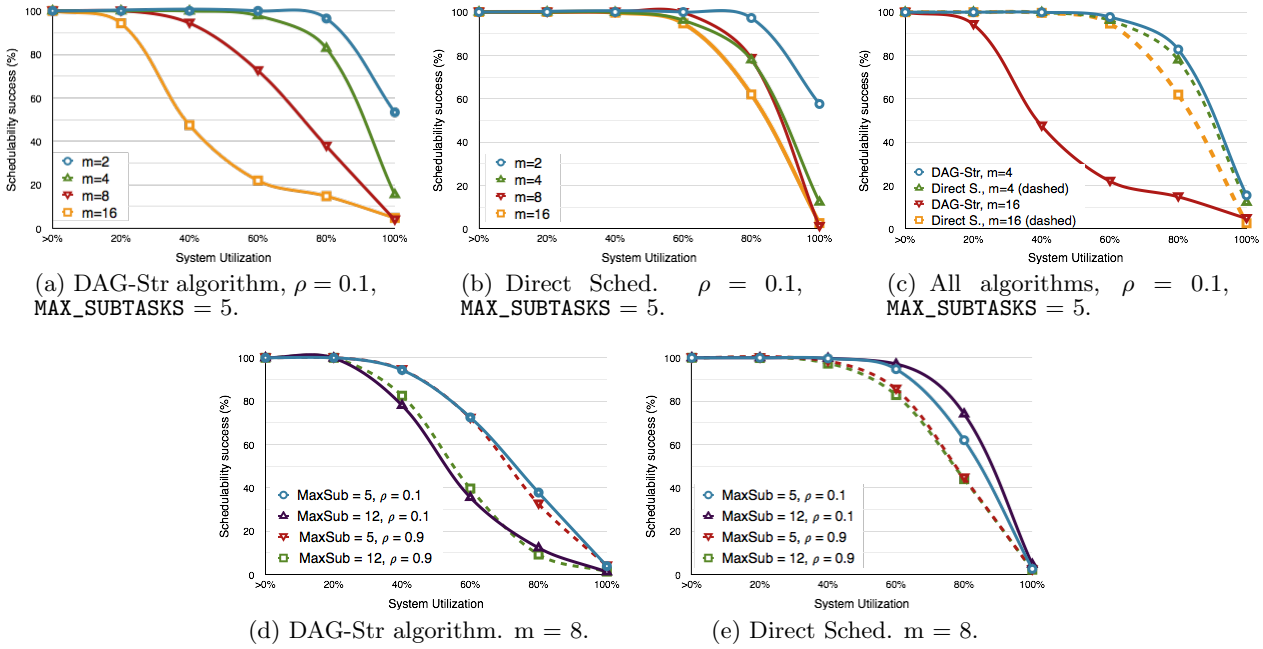
  In this example, subtask $\tau_{i,1}$ has two successors $\tau_{i,2}$ and $\tau_{i,3}$ since $\mathcal{R}_{1,2} = \mathcal{R}_{1,3} = 1$, while there is no directed relations between subtask $\tau_{i,1}$ and subtask $\tau_{i,4}$. Similarly, subtask $\tau_{i,4}$ is the successor of subtasks $\tau_{i,2}$ and $\tau_{i,3}$.

Listing 1. An example of an XML file describing the DAG Tasks

```
<?xml version="1.0" encoding="UTF-8"?>
  ...
  <tasks nbTasks="1" type="Fixed Priority">
    <task deadline="10" firstRelease="0"
        nbSubtasks="2" period="10" priority="1
        " type="graph" wcee="0" wcet="5">
      <subtask children="1" deadline="10"
          firstRelease="0" index="0"
          localDeadline="7" nbProc="1" parents
          ="" period="10" priority="1" type="
          subtask" wcet="2"/>
      <subtask children="-1" deadline="10"
          firstRelease="2" index="1"
          localDeadline="10" nbProc="1"
          parents="0" period="10" priority="1"
           type="subtask" wcet="3"/>
    </task>
  </tasks>
```

Using these parameters, DAG sets are generated randomly and each DAG task is assigned a WCET, a period, a relative deadline and a set of random subtasks with precedence constraints. In YARTISS, each DAG set is encoded in an XML file so as to be used repeatedly in the simulation of different scheduling algorithms. An example of a DAG XML file is shown in Listing 1, which represents a data set of a single task set (tag <`tasks`>) which contains a single DAG task (tag <`task`>). This DAG consists of two subtasks (tag <`subtask`>), in which the first subtask is a parent of the second one (represented by attributes `parents` and `children`). The subtask tag has other attributes such the WCET, deadline, first release time so as to represent their timing parameters.

5.1 Simulation Results for EDF Scheduling Algorithm



(a) DAG-Str algorithm, $\rho = 0.1$, `MAX_SUBTASKS` $= 5$.

(b) Direct Sched. $\rho = 0.1$, `MAX_SUBTASKS` $= 5$.

(c) All algorithms, $\rho = 0.1$, `MAX_SUBTASKS` $= 5$.

(d) DAG-Str algorithm. m $= 8$.

(e) Direct Sched. m $= 8$.

Figure 6. Simulation results of DAG scheduling approaches when EDF is used.

In this subsection, we present simulation results of DAG scheduling approaches when associated with EDF algorithm. These results are analyzed w.r.t. the number of processors in the system, the variation of the size of DAGs and the probability of internal dependencies.

The effect of the number of processors on DAG schedulability

We simulate the DAG scheduling by using the two scheduling approaches with EDF scheduling algorithm. We analyze their schedulability performance w.r.t. the number of processors in the system, and the other simulation parameters (DAG size and parallelism probability) are fixed for each simulation sets.

Figure 6 shows the simulation results of Direct Scheduling and DAG-Str algorithm with EDF. The x-axis of each inset in the Figure denotes the percentage of task set utilization w.r.t. the number of processors in the system, while the y-axis denotes the percentage of schedulable task sets. In the simulations of Insets 6(a), 6(b) and 6(c), we consider that the probability factor ρ is equal to 0.1 (high probability of internal parallelism) and `MAX_SUBTASKS` is equal to 5.

In general, we notice that the performance of all DAG scheduling approaches decreases when the number of processors of the system is increased. However, the performance of the DAG-Str algorithm is more affected by the variation of number of processors than the Direct Scheduling approache. Inset 6(a) shows the simulation results when DAG-Str algorithm is used. For $m = 2$, we notice that the schedulability percentage of stretched DAG sets is almost 100% for all sets whose utilization less than or equal to 80%. Then the schedulability percentage drops to around 50% for utilization equal to 100%. However, the schedulability performance degrades when the number of processors is increased. When $m = 16$, less than 50% of task sets are schedulable when their utilization is greater than 40% of number of processors.

Direct Scheduling approach of DAGs behaves in the same manner but with better schedulability, as shown in Inset 6(b). The simulation results show that this approach schedules successfully more than 60% of DAG sets whose utilization is no more than 80% of any number of processors. As shown in Inset 6(c), the performance of all scheduling approaches is relatively similar when the number of processors is small (although the DAG-Str algorithm has the best performance). However, when we consider $m = 16$, there is a big difference in performance between the DAG-Str algorithm and the Direct Scheduling in favor of the latter.

In conclusion, when EDF is used to schedule DAG tasks on execution platforms of large number of processors, it is better to consider Direct Scheduling approaches rather than DAG-Str algorithm. In other words, the parallel structure of DAG tasks is more compatible with EDF scheduling algorithm.

The effect of the size of DAGs on schedulability

Inset 6(d) (respectively Inset 6(e)) shows the simulation results of DAG-Str algorithm (respectively Direct Scheduling approach) when the size of DAGs is varied. We consider that `MAX_SUBTASKS` is equal to 5 (small DAGs) and 12 (large DAGs) while the number of processors is equal to 8. In these experiments, we analyze the simulation results w.r.t. the maximum and minimum probability of internal parallelism of DAG tasks ($\rho = 0.9$ and $\rho = 0.1$).

We notice that the effect of DAG size depends on the considered scheduling approach. In the case of DAG-Str algorithm, its schedulability performance decreases when the size of DAGs is increased. As shown in Inset 6(d), the DAG-Str algorithm schedules more DAG sets when their size is small. When `MAX_SUBTASKS` is equal to 5, more than 30% of DAG sets, whose utilization is not greater than 80%, are schedulable (when ρ is either 0.1 or 0.9). The schedulabil-

ity percentage drops to around 10% of the same DAG sets when MAX_SUBTASKS is equal to 12. Moreover, we notice that the performance of DAG-Str algorithm is not affected much by the level of internal parallelism of DAG tasks which is represented by the probability factor ρ.

Similarly, Inset 6(e) shows the schedulability of Direct Scheduling approach when the size of DAGs is varied. We notice that its performance is affected by the probability of internal parallelism. When ρ is equal to 0.9, the schedulability performance remained the same even when the size of DAGs is changed. However when ρ is equal to 0.1, Direct approach schedules more DAG sets when MAX_SUBTASKS is equal to 12. As a general remark, Direct Scheduling approach successfully schedules more than 40% of DAG sets whose utilization is less than or equal to 80% of the number of processors.

The effect of internal parallelism of DAGs on schedulability

In this subsection, we analyze the effect of parallelism probability on the schedulability of DAGs using the different scheduling approaches. We present simulation results after considering probability factor ρ equal to 0.1 (solid lines) and 0.9 (dashed lines) in Insets 6(d) and 6(e). As explained earlier, when ρ is close to zero, DAG tasks are more probable to have many dependencies between their subtasks, while a factor close to 1 means that subtasks tend to execute independently within their DAGs. In this experiment set, we choose execution platforms of number processors equal to 8 and MAX_SUBTASKS equal to 5 and 12.

Starting by the DAG-Str algorithm from Inset 6(d), we notice that its performance is not affected by varying the probability factor ρ. This can be explained by mentioning that the DAG-Str algorithm avoids the internal structure of DAG tasks and execute them as sequentially as possible. Inset 6(e) shows the schedulability performance of Direct Scheduling approach improves when the probability of internal dependencies increases. Around 40% of DAG tasks with utilization less than 80% are schedulable when parallelism parameter ρ is equal to 0.9. The percentage of schedulable DAG sets raises up to more than 60% of schedulable DAG tasks with the same utilization when ρ is equal to 0.1.

Conclusion:
Based on the provided simulation results, we conclude that Direct Scheduling approach performs better than DAG-Str algorithm when preemptive EDF algorithm is used.

5.2 Simulation Results for DM Scheduling Algorithm

After analyzing the DAG schedulability performance of EDF algorithm, which is from fixed job priority assignment family, we consider another scheduling algorithm which is from fixed task priority family. DM algorithm assigns priorities to DAGs based on their relative deadlines. Similarly to previous simulations, we present in this subsection performance evaluations of scheduling approaches with DM. We compare the performance of DAG-Str algorithm to the Direct Scheduling approach while varying the number of processors in the system, the size of DAG tasks and the probability of inter-subtask parallelism. The simulation results are shown in Figure 7.

The effect of the number of processors on DAG schedulability

Insets 7(a) and 7(b) show the simulation results which compare the DAG-Str algorithm (solid lines) and Direct Scheduling (dashes lines) w.r.t. the number of processors in the systems. In these experiments, the size of DAGs is fixed to MAX_SUBTASKS equal to 9, and the probability factor ρ is equal to 0.1 (respectively 0.9) in Inset 7(a) (respectively 7(b)). In general, the schedulability performance of both approaches decreases by the increase of number of processors in the system. We notice also that the schedulability of both algorithms is almost identical when $m = 2$. However, DAG-Str algorithm performs better than the Direct Scheduling when the execution platform consists of number of processors m greater than 2. This indicates that DM is more compatible with DAG-Str algorithm and it performs better than EDF scheduling algorithm.

However, when ρ factor is equal to 0.1, the scheduling approaches have better schedulability on large number of processors when compared to the case where ρ is equal to 0.9.

Simulation analysis for Direct Scheduling

In the case of Direct Scheduling approach, the schedulability performance is better for high probability of parallelism ($\rho = 0.1$) rather than low probability, as shown in Inset 7(c). In these simulations, we consider that DAGs consist MAX_SUBTASKS is equal to 12. When ρ is equal to 0.1, around 60% of DAG sets are schedulable with system utilization equal to 80%. When ρ is equal to 0.9, the schedulability percentage drops to less than 20% for the same DAG sets.

Simulation analysis for DAG-Str algorithm

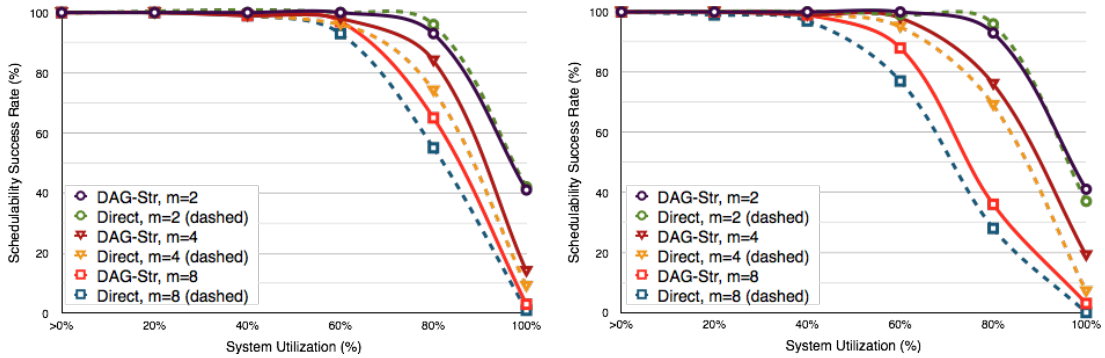
As shown in Inset 7(d) and when the DAG-Str algorithm is applied on DAGs of large sizes (where MAX_SUBTASKS is equal to 12), the schedulability performance decreases by the decrease of parallelism probability but in a rate less than the Direct Scheduling approach.

When ρ is equal to 0.1, the schedulability performance is more than 50% for system utilization equal to 80%. The percentage drops to more than 20% when ρ is equal to 0.9 for the same DAG sets. From these results, we can notice that the schedulability of DAG-Str algorithm is better than Direct Scheduling in the case of low probability of internal parallelism.

As a result and based on the provided simulation results, we conclude that DAG-Str algorithm performs better than Direct Scheduling approach when associated with global preemptive DM algorithm.

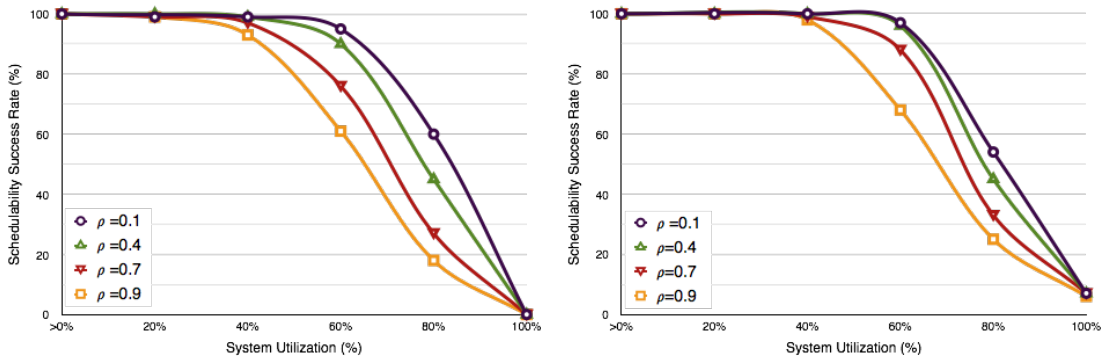
6. CONCLUSION

In this paper, we described two main scheduling approaches for global preemptive parallel real-time DAG tasks on multiprocessor systems. The Direct Scheduling approach, which represents the parallel structure of DAGs, maintains the inter-subtask parallelism of DAGs. While DAG-Str algorithm from the Model Transformation approach transforms DAG tasks into sequential independent model which is easier to schedule. This approach represents the sequential structure of DAGs because it aims at eliminating their internal dependencies. We proved using scheduling examples, that these DAG scheduling approaches are not comparable and



(a) $\rho = 0.1$, `MAX_SUBTASKS` = 9.

(b) $\rho = 0.9$, `MAX_SUBTASKS` = 9.



(c) Direct Sched. m = 16, `MAX_SUBTASKS` = 12.

(d) DAG-Str. m = 16, `MAX_SUBTASKS` = 12.

Figure 7. Simulation results of DAG scheduling approaches when DM is used.

no one of them dominates the other. The comparability analysis was done while considering two global preemptive scheduling algorithms, the Deadline Monotonic (DM) from the fixed task priority assignment family and the Earliest Deadline First (EDF) from the fixed job priority assignment family.

Finally, we performed experimental analyses so as to evaluate the schedulability performance of DAGs. The simulation results showed that DM algorithm is more adapted to the DAG-Str algorithm and EDF scheduling algorithm performs better when DAG tasks are scheduled using the Direct approach.

In the future, we aim at providing more analysis regarding the behavior of such algorithms on the scheduling approaches of DAG scheduling by providing theoretical analysis such as resource augmentation bounds. Moreover, we aim at extending the analysis to consider other scheduling algorithms that can be associated to DAG scheduling approaches, such as Least Laxity First.

7. REFERENCES

[1] OpenMP Application Program Interface version 3.1. http://www.openmp.org/mp-documents/OpenMP3.1.pdf.

[2] POSIX threads programming. https://computing.llnl.gov/tutorials/pthreads/.

[3] S. K. Baruah, V. Bonifaci, 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.

[4] E. Bini and G. C. Buttazzo. Measuring the Performance of Schedulability Tests. *Real-Time Systems*, 30(1-2):129–154, May 2005.

[5] V. Bonifaci, A. Marchetti-Spaccamela, S. Stiller, and A. Wiese. Feasibility Analysis in the Sporadic DAG Task Model. In *Proceedings of the 25th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 225–233, July 2013.

[6] Y. Chandarli, F. Fauberteau, M. Damien, S. Midonnet, and M. Qamhieh. YARTISS: A Tool to Visualize, Test, Compare and Evaluate Real-Time Scheduling Algorithms. In *Proceedings of the 3rd International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, 2012.

[7] H. Chetto, M. Silly, and T. Bouchentouf. Dynamic Scheduling of Real-time Tasks Under Precedence Constraints. *Real-Time Systems*, 2(3):181–194, Sept. 1990.

[8] S. Collette, L. Cucu, and J. Goossens. Algorithm and Complexity for the Global Scheduling of Sporadic Tasks on Multiprocessors with Work-Limited Parallelism. In *Proceedings of the 15th International Conference on Real-Time and Network systems (RTNS)*, 2007.

[9] 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.

[10] D. G. Feitelson. *A Survey of Scheduling in Multiprogrammed Parallel Systems*. IBM TJ Watson Research Center, 1994.

[11] J. Goossens and V. Berten. Gang FTP Scheduling of Periodic and Parallel Rigid Real-time Tasks. In *Proceedings of the 18th Real-Time and Network Systems (RTNS)*, pages 189–196. IRIT Press, Nov. 2010.

[12] J. Goossens and R. Devillers. The Non-Optimality of the Monotonic Priority Assignmentsfor Hard Real-Time Offset Free Systems. *Real-Time Systems*, 13(2):107–126, Sept. 1997.

[13] 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.

[14] C.-C. Han and K.-J. Lin. Scheduling Parallelizable Jobs on Multiprocessors. In *Proceedings of Real-Time Systems Symposium (RTSS)*, pages 59–67, 1989.

[15] M. G. Harbour, M. H. Klein, and J. P. Lehoczky. Fixed Priority Scheduling Periodic Tasks with Varying Execution Priority. In *Proceedings of the 12th IEEE Real-Time Systems Symposium (RTSS)*, pages 116–128, 1991.

[16] P. Jayachandran and T. Abdelzaher. Transforming Distributed Acyclic Systems into Equivalent Uniprocessors under Preemptive and Non-Preemptive Scheduling. In *Proceedings of the 20th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 233–242, 2008.

[17] S. Kato and Y. Ishikawa. Gang EDF Scheduling of Parallel Task Systems. In *Proceedings of the 30th IEEE Real-Time Systems Symposium (RTSS)*, pages 459–468, Dec 2009.

[18] 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.

[19] J. Y.-T. Leung and M. Merrill. A Note on Preemptive Scheduling of Periodic, Real-time Tasks. *Information Processing Letters*, 11(3):115 – 118, 1980.

[20] 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.

[21] J. Li, J.-J. Chen, K. Agrawal, C. Lu, C. Gill, and A. Saifullah. Analysis of Federated and Global Scheduling for Parallel Real-Time Tasks. In *Proceedings of the 26th Euromicro Conference on Real-Time Systems (ECRTS)*, 2014.

[22] 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.

[23] M. Qamhieh, L. George, and S. Midonnet. A Stretching Algorithm for Parallel Real-time DAG Tasks on Multiprocessor Systems. In *Proceedings of the 22Nd International Conference on Real-Time Networks and Systems*, RTNS '14, pages 13:13–13:22. ACM, 2014.

[24] M. Qamhieh and S. Midonnet. Schedulability analysis for directed acyclic graphs on multiprocessor systems at a subtask level. In *Proceedings of the 19th International Conference on Reliable Software Technologies*, Ada-Europe, 2014.

[25] M. Richard, P. Richard, E. Grolleau, and F. Cottet. Contraintes de Precedences et Ordonnancement Monoprocesseur. *The 10 RTS Embedded Systems*, pages 121–138, 2002.

[26] A. Saifullah, K. Agrawal, C. Lu, and C. Gill. Multi-core Real-Time Scheduling for Generalized Parallel Task Models. In *Proceedings of the 32nd IEEE Real-Time Systems Symposium (RTSS)*, pages 217–226, 2011.

[27] A. Saifullah, D. Ferry, J. Li, K. Agrawal, C. Lu, and C. Gill. Parallel Real-Time Scheduling of DAGs. *IEEE Transactions on Parallel and Distributed Systems*, 99(PrePrints):1, 2014.

[28] M. Stigge, P. Ekberg, N. Guan, and W. Yi. The Digraph Real-Time Task Model. In *Proceedings of the 17th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 71–80, April 2011.

[29] J. D. Ullman. NP-Complete Scheduling Problems. *Journal of Computer and System Sciences*, 10(3):384–393, June 1975.

[30] H. X. Zhao, S. Midonnet, and L. George. Worst-Case Response Time Analysis of Sporadic Graph Tasks with Fixed Priority Scheduling on a Uniprocessor. In *Proceedings of Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2005.