Probabilistic Schedulability Analysis for Precedence Constrained Tasks on Partitioned Multi-core

Slim Ben-Amor, Liliana Cucu-Grosjean, Mehdi Mezouak, Yves Sorel

Probabilistic Schedulability Analysis for Precedence Constrained Tasks on Partitioned Multi-core

Slim BEN-AMOR
Kopernic, INRIA
Paris, France
slim.ben-amor@inria.fr

Liliana CUCU-GROSJEAN
Kopernic, INRIA
Paris, France
liliana.cucu@inria.fr

Mehdi MEZOUAK
Kopernic, INRIA
Paris, France
mehdi.mezouak@inria.fr

Yves SOREL
Kopernic, INRIA
Paris, France
yves.sorel@inria.fr

Abstract—**The design of cyber-physical systems (CPSs) is facing the explosion of new functionalities requiring increased computation capacities and, thus, the introduction of multi-core processors. Moreover, some functionalities may impose precedence constraints between the programs implementing these new functionalities. While important effort has been dedicated to the scheduling of precedence constraints tasks on multi-core processors, existing work considers either partitioned scheduling for a single precedence graph defining precedence constraints between tasks, or global scheduling policies.**

In this paper, we consider partitioned scheduling for multiple precedence graphs defining precedence constraints between tasks. The variability of execution times and of communication times is described by probability distributions. We propose a new response time analysis over-performing existing ILP-based results. Thanks to its scalability, our solution is extendable to a probabilistic version and we validate it on a PX4 drone autopilot. Beside this autopilot for our experiments, we implemented a probabilistic extension of a multi-core processor simulator, SimSo. A priority assignment heuristic allowing parallel executions is also proposed. Thanks to its adaptation to partitioned scheduling, our heuristic has better performances than existing solutions and its performances are, also, compared against a genetic-based heuristic.

Index Terms—**Precedence constraints, DAGs, Multi-core, Response time analysis, Partitioning**

I. Introduction

Chip manufacturers are constantly improving hardware performance and they incorporate several cores on the same processor for simultaneous processing, offering a speedup for executing programs. For instance, Intel® proposes the Xeon Phi™ 7920 processor with more than 70 cores [1]. Meanwhile, programming paradigms are evolving as well in order to follow the development of hardware architectures. New parallel programming models are introduced such as OpenMP [2]. These models exploit the possible intra-task parallelism by dividing large tasks into smaller sub-tasks and run them in parallel then merging their results. Such approach creates precedence constraints between several sub-tasks (threads) inside the same task (program). Thus, a Directed Acyclic Graph (DAG) task model may be adopted to describe independent programs as well as dependent threads.

A DAG task model provides a good representation of control parts of a CPS. Indeed, these systems should satisfy precedence constraints between programs in order to ensure their functional correctness. For instance, a specific software executed on the Electronic Control Unit manages and triggers each cycle of the internal combustion engine using different actuators [3] like fuel injectors and valves. The controller is based on the implementation of precedence constraints between software parts to fulfill the expected function.

This research is partially funded by the FR FUI22 CEOS project.

Although in many cases, CPSs may require intensive computation resources, they may not take advantage of the multi-core processors because of interference and communication delays between programs causing important variability on the execution times of those programs. An important variability implies that worst case analysis introduces an increased pessimism as worst case values appear rarely and they are much larger than average values.

In order to reduce the pessimism and its associated over-dimensioning of the hardware architectures, we propose new schedulability techniques complementary to the existing ones. We present a probabilistic schedulability test and a scheduling solution considering the variability of execution times. Our solution is based on a model describing different possible values of execution times by a probability distribution and it estimates a Deadline Miss Probability (DMP) of each task. If large values of the execution times are not frequent, then DMP may be importantly small. Hence, the system becomes schedulable with a high probability, while reducing the pessimism. Such analysis could be applied on soft real-time systems to guarantee a high quality of service, i.e., the DMP is small.

As mentioned in [4], the execution time of a program on a multi-core processor is strongly dependent on the quantity of cross-core interference. DAG task model intensify this interference because of concurrency and communication between the sequential units (sub-tasks) composing them. To reduce interference and interactions between sub-tasks, we focus on partitioned scheduling where each sub-task is assigned to a given core.

Contributions:

- We propose a new task model, where precedence constraints between sub-tasks of the same task are described by a DAG and execution times of sub-tasks are characterized by probability distributions.
- We propose a new Response Time Analysis (RTA) based on iterative equations that are scalable with the number

of sub-tasks. This analysis is adapted to task sets with probabilistic timing parameters.

- We define priorities at sub-task level having as effect reduced response times and less complex analyses.
- We extend the SimSo simulator for DAG tasks, probabilistic execution times and priorities at sub-task level. This extension is available as open source [1].

II. Related Work

The scheduling and the schedulability of tasks with precedence constraints on multi-core processors have received recent attention from the real-time community [5–12]. These results consider, mainly, global scheduling. For instance, Fonseca et al. [13] estimate the response time of sporadic DAG tasks under global and fixed priority scheduling. They use nested fork-join structured DAGs to propose both accurate and efficient solution. In addition, He et al. [14] study the global scheduling of multiple DAG tasks on multi-core processors. They also define sub-tasks execution order inside the same graph to reduce the response time.

Concerning the partitioned scheduling, Rihani et al. [10] study partitioned DAG scheduling on multi-core processors. They consider a single DAG of multi-rate tasks. Authors propose to unfold the graph (duplicate nodes) within an hyper-period obtaining a single graph with a single period. Due to the resulting graph size, the analysis is not scalable. Another result is proposed by Casini et al. [11] focusing on partitioned non-preemptive fixed-priority Scheduling of parallel tasks. The authors propose an approach similar to [9] based on response time analysis of self-suspending tasks [15].

In addition, partitioned scheduling of tasks is studied in the context of distributed systems. Tindell and Clark [16] propose an end-to-end RTA of several independent tasks each composed of a chain of sub-tasks instead of a DAG. This holistic approach was refined later by Palencia et al. [17]. It is used in the MAST tool [18] to analyze multi-path end-to-end flows. This approach is pessimistic since it assumes that higher-priority tasks are always released at each activation of a sub-task from the chain.

In [9], Fonseca et al. tackle the problem of partitioned scheduling of DAG tasks on identical processors according to a preemptive and fixed-priority policy. Authors use self-suspending tasks to model each path on a DAG task. Then, they estimate the response times of self-suspending tasks using approaches presented in [15]. To evaluate their solutions, authors compare them to existing approach [17] on random generated task sets. The holistic approach of Palencia et al. [17] outperforms some approaches from [15], while their ILP-based solution offer the best gain when estimating response time. Therefore, we compare our response time analysis to both [17] and [9].

Existing work on DAG schedulability considers single values for the execution times. To the best of our knowledge, only our previous paper [8] considers multiple values for execution times through probabilistic descriptions and it is dedicated to uniprocessor EDF schedulability of DAGs.

Concerning the priority assignment for sub-tasks inside a DAG task, we compare our priority assignment algorithm to existing heuristics like HLFET (Highest Levels First with Estimated Times), SCEFT [19] (Smallest Co-levels First with Estimated Times) and CPMISF [20] (Critical Path/Most Immediate Successors First).

III. Task Model and Notations

We consider a task set τ of n sporadic tasks $\tau_1, \tau_2, \ldots, \tau_n$ scheduled according to a partitioned preemptive fixed-priority scheduling policy on m identical cores. We denote by π a processor with m identical cores $\pi_1, \pi_2, \ldots, \pi_m$. Each task τ_i is specified by a 3-tuple (G_i, D_i, T_i), where G_i is a DAG describing the internal structure of τ_i, D_i is its deadline and T_i the minimal inter-arrival time between two consecutive arrivals. We consider a constrained deadline tasks set, i.e., $D_i \leq T_i$ for all tasks.

For a task τ_i, the associated DAG G_i is defined by (V_i, E_i), where $V_i = \{\tau_{i,j}\}_{1 \leq j \leq n_i}$ is a set of n_i sub-tasks of τ_i and E_i is the set of the precedence constraints between its sub-tasks.

A sub-task $\tau_{i,j}$ is defined by $(\mathcal{C}_{i,j}, D_i, T_i)$, where $\mathcal{C}_{i,j}$ is its probabilistic worst-case execution time (pWCET) as defined in [21]. In this paper, we consider discrete and finite pWCETs for all sub-tasks. We assume that pWCETs are given and independent, estimating pWCETs is beyond the purpose of this paper.

Each sub-task $\tau_{i,j}$ is mapped to only one core and all its instances are scheduled on that same core denoted $\pi(\tau_{i,j})$. We assume that the mapping between sub-tasks and cores is given. For instance, in Figure 1 the sub-tasks colored in the same colour are scheduled on the same core. In this paper, we consider that the priorities are assigned at sub-task level. Thus, a priority assignment algorithm will assign to each sub-task $\tau_{i,j}$ a priority. We denote by $hp(\tau_{i,j})$ the set of its higher priority sub-tasks.

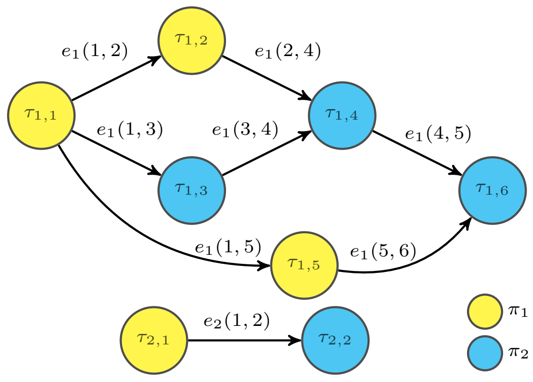

Fig. 1: A task set describing partitioning and precedence constraints between sub-tasks of two DAG tasks τ_1 and τ_2

Each precedence constraint $(\tau_{i,j}, \tau_{i,k}) \in E_i$ imposes that the sub-task $\tau_{i,k}$ is not released until $\tau_{i,j}$ has completed its

[1]Extension of source code available on: https://github.com/SlimBenAmor/simso/tree/DAG_Task

execution. The sub-task $\tau_{i,j}$ is called a "predecessor" of $\tau_{i,k}$, whereas $\tau_{i,k}$ is a "successor" of $\tau_{i,j}$. We call a sub-task without any successors a "sink" sub-task. For the sake of simplicity, we assume that a DAG have a single sink sub-task. Whenever this assumption does not hold, we add an extra sink sub-task with an execution time equal to zero.

For a sub-task $\tau_{i,j}$, we denote the set of its immediate successors by $isucc(\tau_{i,j}) = \{\tau_{i,k} \mid \exists\ (\tau_{i,j}, \tau_{i,k}) \in E_i\}$. Moreover, other sub-tasks may be reachable from $\tau_{i,j}$ by directed paths. We denote the set of these sub-tasks by:

$$succ(\tau_{i,j}) = \{\tau_{i,k} \mid \exists \text{ one directed path from } \tau_{i,j} \text{ to } \tau_{i,k}\}$$

We note that $isucc(\tau_{i,j}) \subseteq succ(\tau_{i,j})$. Similarly, we denote the set of immediate predecessors by $ipred(\tau_{i,j}) = \{\tau_{i,k} \mid \exists(\tau_{i,k}, \tau_{i,j}) \in E_i\}$ and by $pred(\tau_{i,j}) = \{\tau_{i,k} \mid \tau_{i,j} \in succ(\tau_{i,k})\}$.

Two sub-tasks, that are not reachable one from another by a directed path, are called independent and they may execute in parallel if mapped to different cores. We denote by $parallel(\tau_{i,j})$ the set of sub-tasks independent of sub-task $\tau_{i,j}$. More precisely,

$$parallel(\tau_{i,j}) = \{\tau_{i,k} \mid \tau_{i,k} \in V_i \setminus \{pred(\tau_{i,j}) \cup succ(\tau_{i,j})\}\}$$

A weight $e_i(j,k)$ is associated to each precedence constraint $(\tau_{i,j}, \tau_{i,k}) \in E_i, \forall 1 \le i \le n$. This weight accounts for communication costs between $\tau_{i,j}$ and $\tau_{i,k}$ and it is described by a probabilistic worst case communication time distribution. The communication cost is included in our RTA when the sub-tasks are mapped to different cores ($\pi(\tau_{i,j}) \neq \pi(\tau_{i,k})$). Otherwise, if sub-tasks run on the same core, we assume that communication delay is reduced and it is included in the pWCET of each sub-task.

IV. Response Time Analysis

In this section, we present our RTA for the considered DAG task model with probabilistic WCETs. We consider a preemptive and fixed-priority sub-task level scheduling policy where sub-tasks priorities are given, as well as their mapping to the m identical cores. The closest existing analysis for such task model is provided in [9], but a MILP formulation would be difficult to extend to probabilistic WCETs because of scalability problems. Mainly, such extension implies to evolve each MILP equation into a set of $|var_{\mathcal{C}_1}| \times \cdots \times |var_{\mathcal{C}_n}|$ equations, where $|var_{\mathcal{C}_i}|$ is the number of values contained by discrete probability WCETs of a task τ_i. Therefore, our analysis is based on iterative equations inspired by the work of Palencia [17] and extended to provide probability distributions for the response times of sub-tasks.

We define the DMP of a task τ_i as $DMP_i = P(\mathcal{R}_i > D_i)$, where $\mathcal{R}_i = \mathcal{R}^{glob}_{i,sink}$ is the probability distribution of the global response time of the sink sub-task of τ_i as defined in Section IV-B.

A. Probabilistic Operators

We use two probabilistic operators to propose our RTA, applied on independent probability distributions. The assumption of independence is a first step towards stronger further results. Indeed, if a strong dependency exists between distributions, our RTA equations hold by adding the joint probability distributions describing this dependence. For weak dependencies, their introduction has no impact on the RTA result [22]. We leave as future work the use of Bayes nets or express marginal laws with copulas [23] to extend our RTA to dependent case.

The convolution operator sums two probabilistic WCETs.

Definition 1. *The sum $\mathcal{Z}$ of two independent random variables $\mathcal{X}_1$ and $\mathcal{X}_2$ is the* ***convolution*** *$\mathcal{X}_1 \otimes \mathcal{X}_2$ where:*

$$P\{\mathcal{Z} = z\} = \sum_{k=-\infty}^{k=+\infty} P\{\mathcal{X}_1 = k\} P\{\mathcal{X}_2 = z - k\} \tag{1}$$

$$\begin{pmatrix} 3 & 7 \\ 0.1 & 0.9 \end{pmatrix} \otimes \begin{pmatrix} 0 & 4 \\ 0.9 & 0.1 \end{pmatrix} = \begin{pmatrix} 3 & 7 & 11 \\ 0.09 & 0.82 & 0.09 \end{pmatrix}$$

In addition, the maximum operator determines the maximum between two probabilistic WCETs of sub-tasks. This operator compares the probability density function instead of cumulative distribution function as proposed by Diaz et al. [24].

Definition 2. *Let $\mathcal{X}_1$ and $\mathcal{X}_2$ be two independent random variables and $\mathcal{Z} = \max(\mathcal{X}_1, \mathcal{X}_2)$*

If $\mathcal{X}_1$ and $\mathcal{X}_2$ are finite discrete distributions, we may write:

$$p(\mathcal{Z} = t) = \sum_{max(i,j)=t} p(\mathcal{X}_1 = i) p(\mathcal{X}_2 = j) \tag{2}$$

$$\max\left\{\begin{pmatrix} 3 & 7 \\ 0.1 & 0.9 \end{pmatrix}, \begin{pmatrix} 0 & 4 \\ 0.9 & 0.1 \end{pmatrix}\right\} = \begin{pmatrix} 3 & 4 & 7 \\ 0.09 & 0.01 & 0.9 \end{pmatrix}$$

B. Probabilistic Response Time Analysis

We explain our response time equations inspired by the work of Palencia et al. [17]. They consider that synchronous activation of all higher priority tasks always occurs at each sub-task activation in the studied task. Consequently, several higher-priority preemptions could be accounted in the response time while they are not always possible and the estimation of the response time could be pessimistic. Conversely, we compute first the probabilistic response time assuming no running higher-priority DAG that could preempt the sub-task under study $\tau_{i,j}$ and we consider only sub-tasks that are predecessors of $\tau_{i,j}$. We call the resulting response time the **local response time**. Next, we define the **response time in isolation** which includes all sub-tasks from the same graph (predecessor or not) and discards the effect of higher-priority tasks. Last, we compute the **global response time** by adding the effect of preemptions caused by higher-priority DAG tasks.

To illustrate how response time equations work, we use an example of two DAG tasks, described in Table I and Figure 1. For this example, we assume sub-tasks partitioning and their priority assignment are given. We also assume that all communication costs $e_i(j,l)$ are equal to 1 ms if related sub-tasks are mapped to different cores and 0 ms otherwise.

TABLE I: Parameters of sub-tasks in Figures 1

Sub-task	$\mathcal{C}_{\mathbf{i,j}}$	$\mathbf{T_i = D_i}$	**core**	**Priority**
$\tau_{1,1}$	1 ms	50 ms	π_1	3
$\tau_{1,2}$	1 ms	50 ms	π_1	4
$\tau_{1,3}$	2 ms	50 ms	π_2	6
$\tau_{1,4}$	2 ms	50 ms	π_2	7
$\tau_{1,5}$	$\begin{pmatrix} 2 & 7 \\ .6 & .4 \end{pmatrix}$	50 ms	π_1	5
$\tau_{1,6}$	2 ms	50 ms	π_2	8
$\tau_{2,1}$	8 ms	40 ms	π_1	1
$\tau_{2,2}$	10 ms	40 ms	π_2	2

1) Local response time: We provide the local response time analysis for a sub-task $\tau_{i,j}$ in Equation 4. Its proof requires a first preliminary result given by Equation 3, computing the predecessors interference $\mathcal{I}_{i,l}(pred(\tau_{i,j}))$ caused by its predecessors on one of its immediate predecessors $\tau_{i,l}$ and on predecessors of $\tau_{i,l}$ (see Lemma 1 below).

Lemma 1. *The maximum interference caused by a sub-task $\tau_{i,k} \in S^0_{i,l}(\tau_{i,j})$ on the response time of a sub-task $\tau_{i,j}$ is defined as follows:*

$$\mathcal{I}_{i,l}(pred(\tau_{i,j})) = \bigotimes_{\tau_{i,k} \in S^0_{i,l}(\tau_{i,j})} \mathcal{C}_{i,k} \tag{3}$$

where the set $S^0_{i,l}(\tau_{i,j})$ is composed of sub-tasks that are predecessors of $\tau_{i,j}$ but not predecessor of $\tau_{i,l}$ and that could preempt. $\tau_{i,l}$ or one of its predecessors. A sub-task $\tau_{i,k}$ preempts $\tau_{i,a}$, if it is parallel to $\tau_{i,a}$, it executes on the same core as $\tau_{i,a}$ and it has a higher-priority than $\tau_{i,a}$. More formally,

$$S^0_{i,l}(\tau_{i,j}) = \{\tau_{i,k} \in pred(\tau_{i,j}) \setminus \{pred(\tau_{i,l}) \cup \tau_{i,l}\} \mid \exists$$
$$\tau_{i,a} \in pred(\tau_{i,l}) \cup \tau_{i,l} \textit{ such that } \tau_{i,k} \in parallel(\tau_{i,a}),$$
$$\tau_{i,k} \in hp(\tau_{i,a}),\ \pi(\tau_{i,k}) = \pi(\tau_{i,a})\}$$

Proof. To estimate the maximum interference caused by a sub-task $\tau_{i,k} \in S^0_{i,l}(\tau_{i,j})$, we note that the execution of $\tau_{i,k}$ on core $\pi(\tau_{i,k})$ is delaying not only predecessors of $\tau_{i,l}$ executing on core $\pi(\tau_{i,k})$, but all sub-tasks $\tau_{i,a} \in pred(\tau_{i,l}) \cup \{\tau_{i,l}\}$ (whether they execute on $\pi(\tau_{i,k})$ or not). In fact, such sub-task $\tau_{i,a}$ may be a successor of another sub-task in $pred(\tau_{i,l}) \cup \{\tau_{i,l}\}$ that executes on $\pi(\tau_{i,k})$ and it is delayed by $\tau_{i,k}$. Therefore, the maximum interference that sub-tasks of $S^0_{i,l}(\tau_{i,j})$ cause to the local response time of $\tau_{i,l}$, is equal to the convolution (sum) of execution times of all sub-tasks $\tau_{i,k} \in S^0_{i,l}(\tau_{i,j})$. □

For instance, in Figure 1, the set $S^0_{i,l}(\tau_{i,j})$ for all sub-tasks is empty ($= \varnothing$) except for $S^0_{1,5}(\tau_{1,6})$. In fact, $S^0_{1,5}(\tau_{1,6}) = \{\tau_{1,2}\}$ because $\tau_{1,2}$ is parallel to $\tau_{1,5}$ and it is executed on the same core and has higher priority than $\tau_{1,5}$. Also, $\tau_{1,2}$ is predecessor of $\tau_{1,6}$ but it is not predecessor of $\tau_{1,5}$.

The local response time of sub-task $\tau_{i,j}$ is, then, obtained by summing its probabilistic execution time $\mathcal{C}_{i,j}$ and the maximum time needed for all its predecessors to finish their execution as described below.

Theorem 1. *The local response time of sub-task $\tau_{i,j}$ is defined as follows:*

$$\mathcal{R}^{local}_{i,j} = \mathcal{C}_{i,j} \otimes \max_{\substack{\tau_{i,l} \in \\ ipred(\tau_{i,j})}} \left\{\mathcal{R}^{local}_{i,l} \otimes e_i(l,j) \otimes \mathcal{I}_{i,l}(pred(\tau_{i,j}))\right\} \tag{4}$$

Proof. We prove this theorem by mathematical induction from source sub-tasks to their successors until reaching $\tau_{i,j}$. For the first induction step, we verify that Equation 4 holds for source sub-tasks. The local response time of a sub-task $\tau_{i,j}$ considers only $\tau_{i,j}$ and its predecessors while discarding the effect of parallel sub-tasks in the same graph and higher-priority sub-tasks in other graphs. If $\tau_{i,j}$ is a source sub-task without any predecessor, then its local response time is equal to its execution time. Meanwhile, since there is no predecessor sub-tasks to $\tau_{i,j}$, the "maximum" term, in Equation 4 is equal to zero. Therefore, the computed $\mathcal{R}^{local}_{i,j}$ is equal to the execution time $\mathcal{C}_{i,j}$ and Equation 4 is verified for source sub-tasks.

Now, we assume that Equation 4 is valid for all predecessors of a sub-task $\tau_{i,j}$ and we prove that Equation 4 is correct for $\tau_{i,j}$. Indeed, for each immediate predecessors $\tau_{i,l}$ of $\tau_{i,j}$, we assume that $\mathcal{R}^{local}_{i,l}$ is enough for $\tau_{i,l}$ and all its predecessors to finish their executions. Besides, the maximum interference caused on $\tau_{i,l}$ by other predecessors of $\tau_{i,j}$ is equal to $\mathcal{I}_{i,l}(pred(\tau_{i,j}))$ as explained in Equation 3. Since we consider only predecessor sub-tasks in local response time, the latest start time of $\tau_{i,j}$ is equal to the maximum, over immediate predecessors $\tau_{i,l}$, of the convolution (sum) of: (i) the local response time of $\tau_{i,l}$ with the corresponding communication delay $e_i(l,j)$. (ii) the maximum interference $\mathcal{I}_{i,l}(pred(\tau_{i,j}))$ caused by other predecessors of $\tau_{i,j}$. The term $\max_{\tau_{i,l} \in ipred(\tau_{i,j})} \left\{\mathcal{R}^{local}_{i,l} \otimes e_i(l,j) \otimes \mathcal{I}_{i,l}(pred(\tau_{i,j}))\right\}$ provides sufficient time for all predecessors of $\tau_{i,j}$ to be executed and then $\tau_{i,j}$ starts executing. Under the assumption of no preemption from parallel sub-tasks and higher-priority DAG tasks, $\tau_{i,j}$ finishes its execution after $\mathcal{C}_{i,j}$ from its start. Hence, we add, to the "maximum" term, the execution time $\mathcal{C}_{i,j}$ of $\tau_{i,j}$ in order to get the local response time $\mathcal{R}^{local}_{i,j}$. □

2) Response time in isolation: It takes into consideration the effect of all parallel sub-tasks from the same graph (predecessors or not) and it discards preemptions of higher-priority DAG tasks. Since the local response time considers only predecessor sub-tasks, we add to this latter the sum of execution times of sub-tasks, in the set $S^1_{i,j}$, that are not predecessor of $\tau_{i,j}$ and that could preempt $\tau_{i,j}$ or one of its predecessors.

$$S^1_{i,j} = \{\tau_{i,k} \notin pred(\tau_{i,j}) \cup \tau_{i,j} \mid \exists\ \tau_{i,a} \in pred(\tau_{i,j}) \cup \tau_{i,j}$$
$$\text{such that } \tau_{i,k} \in parallel(\tau_{i,a}),\ \tau_{i,k} \in hp(\tau_{i,a}),$$
$$\pi(\tau_{i,k}) = \pi(\tau_{i,a})\}$$

For example, in Figure 1, the set $S^1_{1,5} = \{\tau_{1,2}\}$ because $\tau_{1,2}$ is parallel to $\tau_{1,5}$ and it is executed on the same core and has higher priority than $\tau_{1,5}$. Also, $\tau_{1,2}$ is not a predecessor of $\tau_{1,5}$.

Theorem 2. *The response time in isolation of the sub-task $\tau_{i,j}$ is defined as follows:*

$$\mathcal{R}^{isol}_{i,j} = \mathcal{R}^{local}_{i,j} \otimes \bigotimes_{\tau_{i,k} \in S^1_{i,j}} \mathcal{C}_{i,k} \tag{5}$$

Proof. Similarly to the proof of Equation 4, we prove that the maximum interference that sub-tasks of $S^1_{i,j}$ cause on the response time in isolation of $\tau_{i,j}$, is equal to the convolution (sum) of execution time of all sub-tasks $\tau_{i,k} \in S^1_{i,j}$. $\square$

3) Global response time: The global response time takes into consideration all possible preemptions of higher-priority DAG tasks. It is calculated recursively by Equation 6 similar to the iterative equation in [17] for a deterministic task model. We add to the response time in isolation, the effect of higher priority sub-tasks from other graphs that execute on the same core as $\tau_{i,j}$ or one of its predecessors (grouped in the set $S^2_{i,j}$).

$$S^2_{i,j} = \{\tau_{p,q} \mid p \neq i, \exists\ \tau_{i,l} \in pred(\tau_{i,j}) \cup \tau_{i,j} \text{ such that } \tau_{p,q} \in hp(\tau_{i,j}),\ \pi(\tau_{p,q}) = \pi(\tau_{i,l})\}$$

For instance, in Figure 1, the set $S^2_{1,1} = \{\tau_{2,1}\}$ because $\tau_{2,1}$ belongs to another task τ_2. Also, $\tau_{2,1}$ is executed on the same core and has higher priority than $\tau_{1,1}$.

$$\mathcal{R}^{glob}_{i,j} = \mathcal{R}^{isol}_{i,j} \otimes \bigotimes_{\tau_{p,q} \in S^2_{i,j}} \left\lceil \frac{\mathcal{R}^{glob}_{i,j} \otimes \mathcal{J}_{p,q}}{T_p} \right\rceil \mathcal{C}_{p,q} \tag{6}$$

The jitter: $\mathcal{J}_{i,j} = \max_{\tau_{i,k} \in ipred(\tau_{i,j})} \left\{ \mathcal{R}^{glob}_{i,k} \otimes e_i(k,j) \right\}$. We note that the effect of higher priority DAG tasks is equal to the convolution (sum) of all sub-tasks in $S^2_{i,j}$ in order to upper bound the maximum delays that could be introduced as we do in the proof of Equation 4. The iterative update of $\mathcal{R}^{glob}_{i,j}$ stops when earliest activation of the next job of higher-priority tasks is greater than the maximum value in $\mathcal{R}^{glob}_{i,j}$ distribution (this job cannot preempt $\tau_{i,j}$) or it is greater than the deadline of $\tau_{i,j}$ (even if it preempts $\tau_{i,j}$ it cannot change its DMP).

TABLE II: Response times of sub-tasks described in Figures 1 and Table I

	$\mathcal{R}^{local}_{i,j}$	$S^1_{i,j}$	$\mathcal{R}^{isol}_{i,j}$	$S^2_{i,j}$	$\mathcal{R}^{glob}_{i,j}$
$\tau_{1,1}$	1	∅	1	$\tau_{2,1}$	9
$\tau_{1,2}$	2	∅	2	$\tau_{2,1}$	10
$\tau_{1,3}$	4	∅	4	$\tau_{1,2}, \tau_{2,2}$	22
$\tau_{1,4}$	6	∅	6	$\tau_{1,2}, \tau_{2,2}$	24
$\tau_{1,5}$	$\begin{pmatrix} 3 & 8 \\ .6 & .4 \end{pmatrix}$	$\tau_{1,2}$	$\begin{pmatrix} 4 & 9 \\ .6 & .4 \end{pmatrix}$	$\tau_{2,1}$	$\begin{pmatrix} 12 & 17 \\ .6 & .4 \end{pmatrix}$
$\tau_{1,6}$	$\begin{pmatrix} 8 & 12 \\ .6 & .4 \end{pmatrix}$	∅	$\begin{pmatrix} 8 & 12 \\ .6 & .4 \end{pmatrix}$	$\tau_{1,2}, \tau_{2,2}$	$\begin{pmatrix} 26 & 30 \\ .6 & .4 \end{pmatrix}$
$\tau_{2,1}$	8	∅	8	∅	8
$\tau_{2,2}$	19	∅	19	∅	19

We note that the probabilistic worst-case response time (WCRT) of task τ_1 is equal to $\begin{pmatrix} 26 & 30 \\ .6 & .4 \end{pmatrix}$. However, if we use the approach adopted by Palencia et al [17], we find $R_1 = 46$ with the two possible values for $\mathcal{C}_{1,5}$. Hence, we observe that our analysis reduces pessimism when estimating the WCRT of the DAG task. On the other hand, by using Fonseca et al. [9] approach, $R_1 = 26$ for $\mathcal{C}_{1,5} = 2$ and $R_1 = 30$ for $\mathcal{C}_{1,5} = 7$. However, our analysis is faster by finding the result in 0.002 seconds, while [9] requires 0.3 seconds.

V. Priority Assignment

Fixed-priority scheduling defines priorities at task level according to policies like *Deadline Monotonic* [25] and *Audsley's algorithm* [26]. Applying such policy imposes to any sub-task from the task τ_i to have a higher priority than all sub-tasks of τ_j, if τ_i has a higher priority than τ_j. Since all sub-tasks from the same DAG have the same priority, they would have an arbitrary order of execution.

In our previous work [12], we show that sub-task level priorities decrease the response times and we use this approach in this paper, by defining priorities at sub-task level. We propose two sub-task level priority assignments (i) an heuristic and (ii) a genetic algorithm. These proposed methods are applied on each DAG task and we show that they do reduce response times. They may be used both on probabilistic or non-probabilistic set of tasks.

A. Sub-task Priority Assignment Heuristic

Our assignment heuristic prioritizes a sub-task with the maximum successor workload that executes on a different core than the sub-task itself. Indeed, when such sub-task completes its execution, it allows workload on other cores to start their execution concurrently. In case of equality between two sub-tasks according to the first criteria, we prioritize using topological ordering described by Kahn [27]. In fact, we split a graph into levels that respect precedence constraints and we prioritize the sub-task that belongs to the previous level. This strategy gives higher priority to a predecessor sub-task than its successors which is coherent with precedence constraints.

Algorithm 1: Priority assignment heuristic at sub-task level

```
Data: Tasks $\tau_i$ and $\pi$ set of $m$ cores
Result: Sub-tasks priorities
1  $suc\_sum = zeros(n_i)$
2  for $\tau_{i,j} \in \tau_i$ do
3      for $\tau_{i,l} \in succ(i,j)$ do
4          if $\pi(\tau_{i,l}) \neq \pi(\tau_{i,j})$ then
5              $suc\_sum[j] = suc\_sum[j] + \mathbb{E}(\mathcal{C}_{i,l})$
6          end
7      end
8  end
9  $levels = topologic\_order(\tau_i)$
10 $Priority = argsort(\tau_i, order = [-suc\_sum, levels])$
11 return $Priority$
```

B. Genetic Algorithm

Inspired from Genetic Algorithm (GA) [28], we propose another priority assignment method. Indeed, we create a population composed of a set of several possible priority assignments of sub-tasks inside the same graph. At each generation (iteration) of GA, we combine best members (priority assignments corresponding to the least response time) from the population to obtain a new priority assignment (child member). The proposed algorithm is composed of several steps (cf. Figure 2) detailed as following:

- **Initialization:** use random priority assignment derived from topological order to initialize population members.
- **Evaluation:** compute the response time (fitness, objective function) of the studied graph for each member in the population corresponding to a possible priority assignment at sub-task level.
- **Selection:** choose the two best members in the population (winner and loser), keep the winner in the next generation and replace the loser by the child member obtained after crossover and mutation. The used elitist selection prevents the degradation of the population fitness and preserves the best member in the next generation.
- **Crossover:** select the priority order of a subset of nodes from the loser and insert it in the winner to obtain the child member.
- **Mutation:** swap the priorities of two parallel sub-tasks (not predecessor nor successor) from the child member.

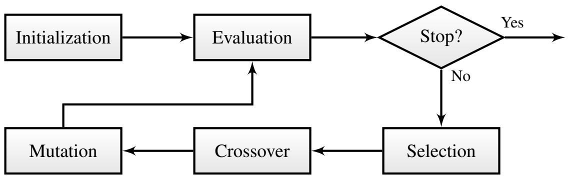

Fig. 2: Flowchart of the used Genetic Algorithm

For the stop condition, we use a limited number of iterations equal to 100 because we note beyond this number, the objective function (response time) becomes almost constant.

VI. Evaluation Results

We evaluate both our RTA and priority assignment algorithms on a real use case (PX4 autopilot) and on randomly generated task sets. We also compare our solutions to similar existing ones by comparing the average performance over 100 generated task sets. Each generated task set is composed of 5 DAG tasks with 100 sub-tasks scheduled on 4 cores. We limit the number of tasks and sub-tasks because the algorithm of [9], used for comparison does not scale with large graphs.

To generate a task set, we use the "*randfixedsum*" algorithm [29] to split a total utilization equal to 50% of the system capacity, into individual task utilization for each DAG task. We also use "*log-uniform*" distribution [30] to generate tasks' periods in the range $[10, 1000\ ms]$. After choosing a period for each task, we set deadlines to their task periods $D_i = T_i$. We calculate the execution time according to $C_i = T_i \times U_i$. Then, we share the total execution time C_i between sub-tasks composing DAG task τ_i. From the individual execution time of each sub-task, we generate a discrete exponential distribution with 5 values. This distribution has an expected value equal to the individual execution time of the sub-task.

In addition, we use a "*layer-by-layer*" method [31] to generate a DAG graph with unbiased structure describing precedence constraints of a task. We consider a probability of 20% to create a precedence constraint between a sub-task and its subsequent sub-tasks in the DAG.

A. Deterministic Response Time Analysis

TABLE III: WCRT ratio regarding MILP based approach [9]

	Avg ratio	**Min ratio**	**Max ratio**
Our RTA	1.61	1	8
Holistic [17]	2.52	1	9.73
MILP [9]	1	1	1

From Table III, we note that our analysis and the holistic one overestimate the WCRT computed wrt the [9] approach. However, our approach introduces less overestimation and pessimism compared to the holistic approach. In average, our WCRT is 1.61 times larger than the WCRT of MILP approach, while the WCRT of the holistic approach is 2.52 times larger. Thus, in average the WCRT of the holistic approach is almost twice larger than our WCRT. Moreover, for some generated task sets, we obtain the same WCRT for three approaches. Meanwhile, with other task sets, we could have large difference between MILP approach and other approaches. The computed WCRT is up to 8 and 9 times larger.

TABLE IV: Comparison of run-time of RTA

	Avg run-time	**Min run-time**	**Max run-time**
Our RTA	2.1 s	0.001 s	11.2 s
Holistic [17]	1.9 s	0.0009 s	10.17 s
MILP [9]	74.7 s	0.01 s	7648 s

Table IV illustrates the run-time performance of the three RTA approaches. We note that run-times of our RTA and holistic analysis are comparable and they are much faster than MILP approach. In average, our algorithm takes 2.1 seconds to deliver a WCRT estimation and the holistic analysis takes 1.9 seconds, while [9] analysis takes 74.7 seconds. Besides, the maximum run-time, over 100 generated task sets, is 11.2 seconds for our approach and 10.17 seconds for the holistic approach, while MILP analysis take more than 2 hours.

B. Probabilistic Response Time Analysis

Here, we compare the results of our RTA when applied to task sets with deterministic and probabilistic execution times. The deterministic analysis is based on worst case reasoning. Hence, it considers the highest execution time from the pWCET and it declares a task set schedulable when the probabilistic analysis finds a probability of schedulability equals to 100%. We note that, in Figure 3, none of the generated task sets reaches the probability of 100% so they won't

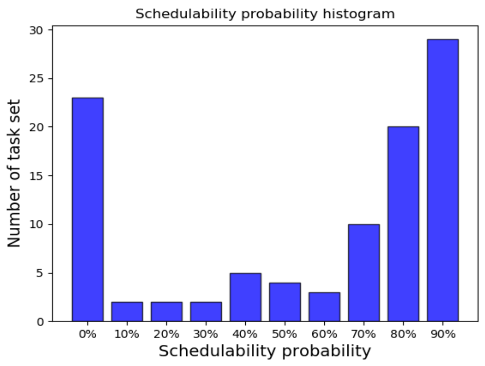

Fig. 3: schedulability of 100 task sets randomly generated

be schedulable using deterministic analysis. However, about half of generated tasks are schedulable with high probability (more than 80%) which highlights the pessimism of the worst case reasoning of deterministic analyses. There is a significant number of task sets with 0% probability to be schedulable (not schedulable under any timing parameters values). This is explained by the random generation of timing and precedence constraints that may be too stringent to be respected.

C. Priority Assignment Algorithm

TABLE V: comparison with other priority assignment algorithm for sub-tasks

	HLFET	SCEFT	CPMISF	GA
WCRT Analysis	114.66%	119.09%	110.81%	113.9%
Simso simulation	106.57%	110.71%	104.23%	107.18%

In order to compare our proposed priority assignment heuristic to HLFET, SCEFT and CPMISF heuristics [19, 20] and to our genetic algorithm, we compute the response times under each priority assignment algorithm. Next, we calculate the ratio of obtained WCRTs by each heuristic and our Genetic Algorithm (GA) over the one obtained by our heuristic. Response times are derived using: (i) our proposed RTA and (ii) our extension of the SimSo simulation.

Extension of SimSo: Simso is a simulation tool developed by Chéramy et al. [32] to evaluate real-time scheduling algorithms. It supports single and multiprocessor. It also supports several models and scheduling policies. However, it does not include DAG task model, priority on sub-task level and probabilistic execution times. Therefore, we extend SimSo[2] to add precedence constraints inside tasks and to specify a static priority for sub-tasks. Hence, our extension of this tool allows to simulate the generated task sets over an hyper-period and to derive the response time from the simulator events log.

Table V shows the obtained results. We note that our proposed priority assignment heuristic reduces both computed and simulated response times compared to other heuristics (HLFET, SCEFT and CPMISF) and the genetic algorithm.

[2]Extension of source code available on: https://github.com/SlimBenAmor/simso/tree/DAG_Task

D. Use Case: PX4 Autopilot

In this section, we present numerical results obtained for DAG tasks corresponding to the open source PX4 autopilot programs of a drone[3]. The structure of the DAG tasks is illustrated in Figure 4.

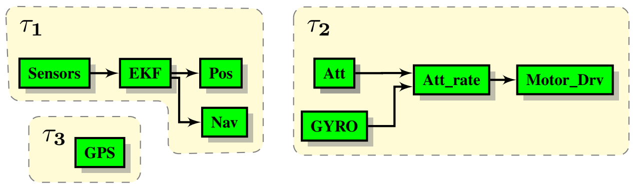

Fig. 4: DAGs describing precedence constraints between sub-tasks of the three tasks representing PX4 Autopilot programs.

The execution time traces have been obtained from hardware-in-the-loop measurements while the sensors and the output drivers are simulated on predefined flying missions on a Pixhawk 4 hardware[4] on top of a NuttX OS[5]. Moreover, when measured, each sub-task is executed with highest priority in order to avoid any preemptions from other sub-tasks. The execution time measurements of the sub-tasks are obtained by executing them on a single core processor (ARM family). In order to obtain the probabilistic bounds, we extract from each empirical distribution several quantiles. The execution times traces will be made to the reader to ensure the reproductibility of our results.

TABLE VI: Comparison of computed DMP and drone behavior

Periods	Drone behavior	DMP
3 ms	Could not fly	0.9999
3.5 ms	Could not fly	0.994
4 ms	Poor stability	0.2696
4.5 ms	Medium stability	0.0049
5 ms	Good stability	1.4959×10^{-14}

First, we compute DMP of the PX4 drone autopilot under different period values (same period for the 3 tasks). Then, we compare them to drone behavior already evaluated with different period settings. Results are illustrated in Table VI. We note that the obtained DMPs are coherent with the drone behavior obtained from simulation. For instance, when the tasks' period is relatively small the DMP is very high (near to one) and the drone could not fly because the execution frequency of programs is very high and they cannot finish their execution before deadline ($D_i = T_i$). Moreover, the DMP is reduced to 10^{-14} when period is not small and the drone shows a good stability.

Now, we assume that the set of three tasks of the PX4 autopilot (Figure 4) is scheduled on a dual core processor with two identical cores. Then, we compute their DMP to study the schedulability of the system on such hardware architecture.

[3]https://en.wikipedia.org/wiki/PX4_autopilot
[4]https://pixhawk.org
[5]http://nuttx.org

TABLE VII: DMP of PX4 autopilot tasks under dual core processor with different period configurations

$\mathbf{T_1}$	$\mathbf{T_2}$	$\mathbf{T_3}$	DMP τ_1	DMP τ_2	DMP τ_3
$4\ ms$	$7\ ms$	$10\ ms$	0	0.1536	0
$3\ ms$	$7\ ms$	$10\ ms$	2.7×10^{-8}	0.9147	0
$3\ ms$	$6\ ms$	$10\ ms$	2.7×10^{-8}	0.9993	0
$3\ ms$	$6\ ms$	$7\ ms$	2.7×10^{-8}	0.9993	0.0006
$3\ ms$	$6\ ms$	$7\ ms$	2.7×10^{-8}	0.9993	0.0006
$2\ ms$	$4\ ms$	$5\ ms$	0.7082	0.9999	0.9271
$4\ ms$	$2\ ms$	$5\ ms$	0	0	0
$3\ ms$	$2\ ms$	$5\ ms$	5.5×10^{-6}	0	0.0451

We consider that sub-tasks **Sensors**, **EKF**, **Nav**, **GYRO** and **GPS** are assigned to the first core while sub-tasks **Pos**, **Att**, **Att_rate** and **Motor_Drv** are assigned to the second core.

Results are illustrated in Table VII for different periods combinations. Since priorities at task level are defined by *rate monotonic*, all programs of task τ_1 have higher priorities than τ_2 ($T_1 < T_2$) in the first six experiments in Table VII. We note that DMPs of the three tasks increase as we decrease periods because their period are too small to respect their deadlines. For the two last experiments, we inverse the priorities of τ_1 and τ_2 by choosing $T_2 < T_1$. We notice that DMP are significantly reduced even with smaller periods. Thus, we suggest to change the priorities of programs to accord the highest priority to task τ_2. We note also that under this configuration, we guarantee a low DMP with smaller periods than in case of single core. Hence, the parallelization on a dual core processor, allows to reach a more reactive and schedulable system.

VII. Conclusion

In this paper, we tackle the problem of partitioned scheduling of tasks with precedence constraints defined by multiple DAGs. We develop iterative response time equations inspired from [17] and we extend them to fit our task model with probabilistic execution and communication times. Our RTA gives comparable results, in average, with Fonseca et al. [9] method based on MILP while extensively decreasing the calculation time. Besides, we propose to assign priorities to sub-tasks and we show the effectiveness of our ordering strategy.

Our approach is validated on a PX4 drone autopilot. First, we computed the DMP of autopilot tasks on a single core processor and we analysed the stability of the drone with different execution periods. Then, we considered a dual core processor. We succeed to reduce tasks' periods while guaranteeing low DMP by reordering tasks' priorities. Hence, we make the system more reactive.

As future work, we consider the study of dependent probability distributions and their impact on the RTA, as well as the consideration of heterogenous cores.

References

[1] Intel, ""Intel Xeon Phi Product Family"," (2016). [Online]: https://www.intel.com/content/products/processors/xeon-phi.html.

[2] API specification for parallel programming, "OpenMP 5.0," (2018). [Online]:www.openmp.org/press-release/openmp-5.

[3] J. G. Kassakian, H. . Wolf, J. M. Miller, and C. J. Hurton, "Automotive electrical systems circa 2005," *IEEE Spectrum*, 1996.

[4] S. Altmeyer, R. I. Davis, L. Indrusiak, C. Maiza, V. Nelis, and J. Reineke, "A generic and compositional framework for multicore response time analysis," in *RTNS*, 2015.

[5] N. Ueter, G. von der Bruggen, J.-J. Chen, J. Li, and K. Agrawal, "Reservation-based federated scheduling for parallel real-time tasks," in *RTSS*, 2018.

[6] X. Jiang, N. Guan, X. Long, and W. Yi, "Semi-federated scheduling of parallel real-time tasks on multiprocessors," in *RTSS*, 2017.

[7] R. Medina, É. Borde, and L. Pautet, "Scheduling multi-periodic mixed-criticality dags on multi-core architectures," in *RTSS*, 2018.

[8] S. Ben-Amor, D. Maxim, and L. Cucu-Grosjean, "Schedulability analysis of dependent probabilistic real-time tasks," in *RTNS*, 2016.

[9] J. C. Fonseca, G. Nelissen, V. Nélis, and L. M. Pinho, "Response time analysis of sporadic DAG tasks under partitioned scheduling," in *11th IEEE Symposium on Industrial Embedded Systems, SIES*, 2016.

[10] H. Rihani, M. Moy, C. Maiza, R. Davis, and S. Altmeyer, "Response Time Analysis of Synchronous Data Flow Programs on a Many-Core Processor," in *RTNS*, 2016.

[11] D. Casini, A. Biondi, G. Nelissen, and G. Buttazzo, "Partitioned fixed-priority scheduling of parallel tasks without preemptions," in *2018 IEEE Real-Time Systems Symposium (RTSS)*, 2018.

[12] S. Ben-Amor, D. Maxim, and L. Cucu-Grosjean, "Worst-case response time analysis for partitioned fixed-priority dag tasks on identical processors," in *ETFA, WIP session*, 2019.

[13] J. Fonseca, G. Nelissen, and V. Nélis, "Improved response time analysis of sporadic DAG tasks for global FP scheduling," in *Proceedings of the 25th International Conference RTNS*, 2017.

[14] Q. He, x. jiang, N. Guan, and Z. Guo, "Intra-task priority assignment in real-time scheduling of dag tasks on multi-cores," *IEEE Transactions on Parallel and Distributed Systems*, 2019.

[15] G. Nelissen, J. C. Fonseca, G. Raravi, and V. Nélis, "Timing analysis of fixed priority self-suspending sporadic tasks," *ECRTS*, 2015.

[16] K. Tindell and J. Clark, "Holistic schedulability analysis for distributed hard real-time systems," *Microprocessing and Microprogramming*, vol. 40, 1994.

[17] J. C. P. Gutiérrez, J. J. G. García, and M. G. Harbour, "On the schedulability analysis for distributed hard real-time systems," in *Proceedings of the Ninth Euromicro Workshop on Real-Time Systems, RTS*, 1997.

[18] M. Gonzalez Harbour, J. J. Gutierrez Garcia, J. C. Palencia Gutierrez, and J. M. Drake Moyano, "Mast: Modeling and analysis suite for real time applications," in *ECRTS*, 2001.

[19] Y.-K. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Comput.*, vol. 31, 1999.

[20] Kasahara and Narita, "Practical multiprocessor scheduling algorithms for efficient parallel processing," *IEEE Transactions on Computers*, 1984.

[21] R. I. Davis and L. Cucu-Grosjean, "A survey of probabilistic timing analysis techniques for real-time systems," *LITES*, vol. 6, 2019.

[22] M. Santos, B. Lisper, G. Lima, and V. Lima, "Sequential composition of execution time distributions by convolution," in *CRTS*, 2011.

[23] G. Bernat, A. Burns, and M. Newby, "Probabilistic timing analysis: An approach using copulas," *J. Embedded Comput.*, vol. 1, 2005.

[24] J. L. Diaz, D. F. Garcia, Kanghee Kim, Chang-Gun Lee, L. Lo Bello, J. M. Lopez, Sang Lyul Min, and O. Mirabella, "Stochastic analysis of periodic real-time systems," in *RTSS*, 2002.

[25] J. Y.-T. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic, real-time tasks," *Perform. Eval.*, vol. 2, 1982.

[26] N. C. Audsley, "Optimal priority assignment and feasibility of static priority tasks with arbitrary start times," Department of Computer Science, University of York, Tech. Rep. YCS-164, 1991.

[27] A. B. Kahn, "Topological sorting of large networks," *Commun. ACM*, vol. 5, no. 11, Nov. 1962.

[28] I. Harvey, "The microbial genetic algorithm," in *ECAL*, 2009.

[29] P. Emberson, R. Stafford, and R. Davis, "Techniques for the synthesis of multiprocessor tasksets," in *WATERS*, 2010.

[30] R. I. Davis, A. Zabos, and A. Burns, "Efficient exact schedulability tests for fixed priority real-time systems," *IEEE Transactions on Computers*, vol. 57, no. 9, 2008.

[31] D. Cordeiro, G. Mounié, S. Perarnau, D. Trystram, J.-M. Vincent, and F. Wagner, "Random graph generation for scheduling simulations," in *3rd International Conference on Simulation Tools and Techniques*, 2010.

[32] M. Chéramy, P.-E. Hladik, and A.-M. Déplanche, "Simso: A simulation tool to evaluate real-time multiprocessor scheduling algorithms," in *WATERS*, 2014.