Probabilistic Schedulability Analysis for Real-time Tasks with Precedence Constraints on Partitioned Multi-core

Abstract—The design of embedded systems 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 them. While important effort has been dedicated to the scheduling of precedence constraints tasks on multi-core, most of existing work consider either partitioned scheduling for single precedence graph defining precedence between tasks or global scheduling policies.

In this paper, we consider partitioned scheduling of tasks with precedence constraints defined by multiple Directed Acyclic Graphs (DAGs). The variability of execution and communication times is taken into account by describing them by probability distributions. As the only existing response time analysis RTA for partitioned multiple DAG tasks is based on a non-scalable ILP solving, we propose a new RTA performing faster than existing results. Our solution is extendable to the probabilistic case and validated it on a PX4 drone autopilot. A priority assignment algorithm promoting parallel execution is also proposed. This algorithm is over-performing existing solutions because it is adapted for partitioned scheduling.

Index Terms—Precedence constraints, DAG task, Multi-core, Response time analysis, Partitioning

I. INTRODUCTION

Chip manufacturers are constantly seeking to improve hardware performance. They have incorporated several cores on the same processor to allow simultaneous processing, which offers a speedup for executing programs. For instance, Intel® has proposed the Xeon Phi™ 7920 processor with more than 70 cores [1]. In the meantime, programming paradigms have evolved to keep up with the development of hardware architecture. New parallel programming models have been introduced such as OpenMP [2] and Intel Threading Building Blocks [3]. These models try to exploit the possible intra-task parallelism. They divide large tasks into smaller sub-tasks and run them in parallel. Then, they synchronize and merge their results. This approach creates precedence constraints between several sub-tasks (threads) inside the same task (program). Thus, a DAG task model is adopted to describe different independent programs as well as dependent threads inside them.

Moreover, the DAG task model could provide a good representation of control systems. In fact, these systems should satisfy some precedence constraints between different programs and components in order to ensure the functional correctness. For instance, a specific software executed on the Electronic Control Unit (ECU) manages and triggers each cycle of the internal combustion engine using different actuators [4] like fuel injectors and valves. The controller should enforce several precedence constraints between different parts of software to achieve the desired functioning and optimal performance.

Although in many cases, real-time systems require intensive computation resources, they do not take advantage of the parallel processing provided by multi-cores because traditional timing analysis techniques do not allow such systems to be validated. In fact, these techniques are designed to analyze sequential software that runs on simple architectures. However, most of multi-cores represent a complex and unpredictable hardware that introduce a lot of variability on the execution times of programs. They also add interference and communication delays. As a result, the estimated worst-case execution time may become relatively high. Since traditional timing analysis techniques are based on the worst-case reasoning, using these techniques to analyze such systems will increase pessimism while making the validation process harder.

Pessimism is introduced during the validation because of the important difference between the average and the worst-case execution time that appears rarely. As this difference increases for systems designed on multi-cores, the validation process tends to reject a design solution even if it is feasible under a significant number of execution scenarios. For example, a task with an error recovery mechanism invokes recovery routine when encountering errors with a very low probability (less than $10^{-12}$ per hour of functioning). Such task may have two possible values for its execution time. A small value when no recovery routine needed and a larger value otherwise. A worst case reasoning assumes that errors occur always and considers only large value of worst-case execution times (WCETs) for that task. This leads to pessimistic response times. Therefore, additional computation resources are required for validation and the system will be oversized.

For the purpose of reducing pessimism and over-sizing, we consider new schedulability techniques complementary to the traditional ones, especially that future systems are expected to face significant more evolving external environments because of the current trend of introducing artificial intelligence mechanisms on top of the more classical control ones. Consequently, we present a probabilistic schedulability and scheduling solution that takes into consideration the variability of execution times. Our solution is based on a model describing different possible values of a parameter by a probability distribution. It includes the study of different execution scenarios 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 small. Hence, the system becomes schedulable 
with high confidence, which reduces the pessimism. Such 
an analysis could be applied on soft real-time systems to 
guarantee a high quality of service when the DMP is small. It 
could also be used on industrial systems with safety standards 
that require low probability of failure.

As mentioned in [3], the execution time on multi-core 
platforms is strongly dependent on the quantity of cross-
core interference. DAG task model and parallel tasks 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. In addition, we study a fixed-priority 
scheduling policy on identical cores.

Contributions: In this work, we use the potential parallelism 
between the sub-tasks to decrease the response time of 
a set of DAG tasks. We also define the execution order of sub-
tasks to simplify the schedulability analysis and to reduce the 
response time and enhance the reactivity of the system. Our 
contributions can be summarized as follows:

- We propose a new task model, where the precedence 
  constraints between the sub-tasks of a task are described 
  by a DAG and probabilistic bounds on the execution times 
  of sub-tasks are considered.
- We propose a new RTA adapted to the task set with proba-
  bilistic execution times. This analysis is based on iterative 
equations instead of mixed integer linear programming 
MILP formulation used by previous work.
- We assign priorities at the sub-task level to define the 
  execution order between different nodes from the same 
  graph, which reduces the response time of the entire DAG 
task, while considering communication times for the sub-
tasks scheduled on different cores.

This paper is organized into seven sections. After presenting 
the context and the motivation of our work (Section I), we 
define the task model in Section II. Then, we propose a new 
RTA in Section III. Next, we describe the algorithm used for 
sub-task priorities assignment in Section IV. In Section V we 
illustrate numerical evaluation results on both synthetic pro-
grams and PX4 autopilot programs. We present relevant related 
work with respect to our contribution in Section VI. Finally, 
we present our conclusions and future work in Section VII.

II. TASK MODEL AND NOTATIONS

We consider a real-time system of n sporadic tasks sched-
uled according to a partitioned preemptive fixed-priority 
scheduling policy on m identical cores. We denote by τ 
the set of n tasks τ₁, τ₂, ..., τₙ and by π the processor 
that has m identical cores π₁, π₂, ..., πₘ. Each task τᵢ is 
specified by a 3-tuple (Gᵢ, Dᵢ, Tᵢ), where Gᵢ is a DAG 
describing the internal structure of τᵢ, Dᵢ is its deadline and 
Tᵢ the minimal inter-arrival time between two consecutive 
arrivals. We consider a constrained deadline for all tasks, i.e., 
Dᵢ ≤ Tᵢ, ∀ i ∈ {1, 2, ..., n}.

For a task τᵢ, the associated DAG Gᵢ is defined by (Vᵢ, Eᵢ), 
where Vᵢ = {τᵢ,j}₁≤j≤nᵢ is a set of nᵢ sub-tasks of τᵢ and Eᵢ 
is the set of the precedence constraints between sub-tasks of 
task τᵢ while different tasks are independent between each 
others. A sub-task τᵢ,j is defined by (Ci,j, Di,j, Ti), where 
Ci,j is its probabilistic worst-case execution time (pWCET). 
By pWCET of a sub-task we understand the upper bound 
on the execution time distribution of the program for every 
valid scenario of operation, where a scenario of operation 
is defined as an infinitely repeating sequence of input states 
and initial hardware states that characterise a feasible way in 
which recurrent execution of the program may occur [6]. We 
consider that the probability distributions are independent 
and given. Obtaining such distributions is beyond the purpose 
of this paper.

Each sub-task τᵢ,j, ∀1 ≤ i ≤ n, ∀1 ≤ j ≤ nᵢ is assigned 
to a core and all instances of that sub-task are scheduled on 
the same core denoted π(τᵢ,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 τᵢ,j, ∀1 ≤ i ≤ n, ∀1 ≤ j ≤ nᵢ a 
priority. We denote hp(τᵢ,j) as the set of sub-tasks τᵢ,q with 
higher priority than τᵢ,j.

Each precedence constraint (τᵢ,j, τᵢ,k) ∈ Eᵢ imposes that 
the sub-task τᵢ,k is not released until τᵢ,j has completed its 
execution. The sub-task τᵢ,j is called a “predecessor” of τᵢ,k, 
wheras τᵢ,k is a “successor” of τᵢ,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 τᵢ,j, we denote the set of its immediate 
successors by isucc(τᵢ,j) = {τᵢ,k | ∃ (τᵢ,j, τᵢ,k) ∈ Eᵢ}. 
Moreover, other sub-tasks may be reachable from τᵢ,j by

![Fig. 1: Example of a DAG graph describing partitioning and precedence constraints between sub-tasks of task τ₁]
directed paths. We denote the set of these sub-tasks by:

\[ \text{succ}(\tau_{i,j}) = \{ \tau_{i,k} \mid \exists \text{ at least one path from } \tau_{i,j} \text{ to } \tau_{i,k} \} \]

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

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

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

A weight \( e_i(j, k) \) is associated to each precedence constraint \( (\tau_{i,j}, \tau_{i,k}) \in E_i, \forall 1 \leq i \leq n \). 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 (pWCCT). The communication cost is included in the 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. Thus, the communication cost becomes equal to zero.

### III. Schedulability Analysis

In this section, we present our RTA for sub-tasks with precedence constraints described by DAGs and with probabilistic WCET, scheduled according to a given fixed-priority and partitioned policy.

The closest existing analysis for such tasks systems is provided in \([7]\), but their Mixed Integer Linear Programming (MILP) formulation is not scalable if extended to take into account probabilistic WCET. Mainly such extension implies to evolve each MILP equation into a set of \([\text{var}_1] \times \cdots \times [\text{var}_n] \) equations, where \([\text{var}_i], \forall 1 \leq i \leq n \) is the number of values contained by the discrete probability distribution describing the WCET of a task \( \tau_i \). Therefore, our analysis is based on fixed-point response time equations inspired by Palencia results \([8]\) and it provides probability distributions for the worst-case response times (pWCRTs) of sub-tasks. We define the DMP for task \( \tau_i \) as \( DMP_i = P(R_i > D_i) \), where \( R_i \) is the probability distribution of the WCRT of a task \( \tau_i \) and it is calculated as the response time of the sink sub-task of task \( \tau_i \) (see Equation \( 6 \)).

#### A. Probabilistic Operators

We use two probabilistic operators required for our RTA that operate on independent probability distributions. The assumption of independence its necessary to work with such simple operators. However, if a strong dependency exists between distributions, the following RTA equations hold and we need to use other operators taking this dependency into account. In future work, we will use Bayes nets or express marginal laws with copulas like in \([9]\) to deal with dependency.

The convolution operator sums two random variables that represent, for example, probabilistic execution times.

**Definition 1.** The sum \( Z \) of two independent random variables \( X_1 \) and \( X_2 \) is the convolution \( X_1 \ast X_2 \) where:

\[
P\{Z = z\} = \sum_{k=-\infty}^{k=+\infty} P\{X_1 = k\}P\{X_2 = z-k\} \tag{1}
\]

For example:

\[
\begin{pmatrix} 3 & 7 \\ 0.1 & 0.9 \end{pmatrix} \ast \begin{pmatrix} 0 & 4 \\ 0.9 & 0.1 \end{pmatrix} = \begin{pmatrix} 3 & 7 & 11 \\ 0.09 & 0.82 & 0.09 \end{pmatrix}
\]

In addition, the maximum operator determines the maximum between two random variables such as the probabilistic execution times of sub-tasks. This operator compares the probability density function instead of cumulative distribution function like the one proposed by Diaz et al. \([10]\).

**Definition 2.** Let \( X_1 \) and \( X_2 \) be two independent random variables and \( Z = \max(X_1, X_2) \)

\[
p(Z \leq t) = p(\max(X_1, X_2) \leq t) = p(X_1 \leq t, X_2 \leq t) = p(X_1 \leq t)p(X_2 \leq t) = \sum_{i=\min(X_1)}^{t} p(X_1 = i) \sum_{j=\min(X_2)}^{t} p(X_2 = j)
\]

If \( X_1 \) and \( X_2 \) are finite discrete distributions, we may write:

\[
p(Z = t) = \sum_{\max(i,j) = t} p(X_1 = i)p(X_2 = j) \tag{2}
\]

**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

Then, we explain the proposed response time equations inspired by Palencia et al. work \([8]\). The authors in \([8]\) consider that the worst-case scenario for a DAG task occurs when all the higher-priority tasks are released at each activation of a sub-task in the studied graph. Consequently, several higher-priority preemptions could be accounted in the response time. In fact, these preemptions are not always possible and the assumption about worst-case scenario could be very pessimistic. Conversely, we compute first the response time of the whole graph assuming no higher-priority DAG that could preempt the sub-task under study \( \tau_{i,j} \) are running 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 considers only sub-tasks from the same graph and discards the effect of higher-priority DAG tasks. Last, we compute the global response time by adding the effect of preemptions of higher-priority DAG tasks.

To illustrate how response time equations work, we use an example of two DAG tasks, described in Table \( 1 \) and in Figures \( 1 \) and \( 2 \). We assume that 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.
Fig. 2: Precedence constraints of DAG task $\tau_2$

TABLE I: Parameters of sub-tasks in Figures 1 and 2

<table>
<thead>
<tr>
<th>Sub-task</th>
<th>$C_{i,j}$</th>
<th>$T_i = D_j$</th>
<th>core</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_{1,1}$</td>
<td>1 ms</td>
<td>50 ms</td>
<td>$\pi_1$</td>
<td>3</td>
</tr>
<tr>
<td>$\tau_{1,2}$</td>
<td>1 ms</td>
<td>50 ms</td>
<td>$\pi_1$</td>
<td>4</td>
</tr>
<tr>
<td>$\tau_{1,3}$</td>
<td>2 ms</td>
<td>50 ms</td>
<td>$\pi_2$</td>
<td>6</td>
</tr>
<tr>
<td>$\tau_{1,4}$</td>
<td>2 ms</td>
<td>50 ms</td>
<td>$\pi_2$</td>
<td>7</td>
</tr>
<tr>
<td>$\tau_{1,5}$</td>
<td>$\left(\frac{7}{\text{0.4}}, \frac{7}{\text{0.4}}\right)$</td>
<td>50 ms</td>
<td>$\pi_1$</td>
<td>5</td>
</tr>
<tr>
<td>$\tau_{1,6}$</td>
<td>2 ms</td>
<td>50 ms</td>
<td>$\pi_2$</td>
<td>8</td>
</tr>
<tr>
<td>$\tau_{2,1}$</td>
<td>8 ms</td>
<td>40 ms</td>
<td>$\pi_1$</td>
<td>1</td>
</tr>
<tr>
<td>$\tau_{2,2}$</td>
<td>10 ms</td>
<td>40 ms</td>
<td>$\pi_2$</td>
<td>2</td>
</tr>
</tbody>
</table>

1) Local response time: For the calculation of the local response time of sub-task $\tau_{i,j}$, we sum the probabilistic execution time of $\tau_{i,j}$ and the maximum probabilistic response time over its predecessors:

$$R^\text{local}_{i,j} = C_{i,j} \times \max_{\tau_{i,l} \in \text{ipred}(\tau_{i,j})} \{ R^\text{local}_{i,l} \times e_{i,l} \times I_{i,l}(\text{pred}(\tau_{i,j})) \}$$

(3)

Where $\tau_{i,l} \in \text{ipred}(\tau_{i,j})$ and $I_{i,l}(\text{pred}(\tau_{i,j}))$ represent the interference caused by predecessors of $\tau_{i,j}$ on $\tau_{i,l}$ and its predecessors. We compute this interference by summing the execution time of any predecessor of $\tau_{i,j}$ that has a higher-priority, is executed on the same core and is parallel to $\tau_{i,l}$ or any of its predecessors. It is given by:

$$I_{i,l}(\text{pred}(\tau_{i,j})) = \bigotimes_{\tau_{i,k} \in S^0_{i,j}(\tau_{i,j})} C_{i,k}$$

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

2) Response time in isolation: The RTA in isolation discards only higher-priority sub-tasks from other sub-tasks. Since the local response time considers only predecessors sub-tasks, we add to this latter the sum of execution times of parallel sub-tasks with higher-priority from the same graph that are executed on the same core as the studied sub-task $\tau_{i,j}$ and that are not predecessors of $\tau_{i,j}$.

$$R^\text{isol}_{i,j} = R^\text{local}_{i,j} \times \bigotimes_{\tau_{i,k} \in S^1_{i,j}} C_{i,k}$$

(4)

$$S^1_{i,j} = \{\tau_{i,k} \in V_i \setminus \{\text{pred}(\tau_{i,j}) \cup \tau_{i,j}\} | \exists \tau_{i,l} \in \text{pred}(\tau_{i,j}) \cup \tau_{i,j} \text{ such that } \tau_{i,k} \in \text{parallel}(\tau_{i,l}), \tau_{i,k} \in \text{hp}(\tau_{i,l}), \pi(\tau_{i,k}) = \pi(\tau_{i,l})\}$$

For example, in Figure 1 the set $S^1_{1,1} = \{\tau_{1,2}\}$ 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 $\tau_{1,5}$.

3) Global response time: The global response time takes into consideration all possible preemptions of higher-priority tasks. It is calculated recursively by Equation 5. We add to the response time in isolation the effect of higher-priority tasks.

$$R^\text{glob}_{i,j} = R^\text{isol}_{i,j} \times \bigotimes_{\tau_{p,q} \in S^2_{i,j}} \left[\frac{R^\text{glob}_{i,j} \times J_{p,q}}{T_p} \right] C_{p,q}$$

(5)

Where:

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

$$J_{i,j} = \max_{\tau_{i,k} \in \text{pred}(\tau_{i,j})} \{ R^\text{glob}_{i,k} \}$$

The set $S^2_{1,1} = \{\tau_{2,1}\}$ because $\tau_{2,1}$ belongs to another higher-priority task $\tau_{2}$ and it is executed on the same core.

The response time of DAG task $\tau_i$ is equal to the global response time of the sink task (Equation 6):

$$R_i = R^\text{glob}_{i,sink}$$

(6)

TABLE II: Response times of sub-tasks in Figures 1 and 2

<table>
<thead>
<tr>
<th>$R^\text{local}_{i,j}$</th>
<th>$S^1_{i,j}$</th>
<th>$R^\text{isol}_{i,j}$</th>
<th>$S^2_{i,j}$</th>
<th>$R^\text{glob}_{i,j}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_{1,1}$</td>
<td>1</td>
<td>$\emptyset$</td>
<td>1</td>
<td>$\tau_{2,1}$</td>
</tr>
<tr>
<td>$\tau_{1,2}$</td>
<td>2</td>
<td>$\emptyset$</td>
<td>2</td>
<td>$\tau_{2,1}$</td>
</tr>
<tr>
<td>$\tau_{1,3}$</td>
<td>4</td>
<td>$\emptyset$</td>
<td>4</td>
<td>$\tau_{2,2}$</td>
</tr>
<tr>
<td>$\tau_{1,4}$</td>
<td>6</td>
<td>$\emptyset$</td>
<td>6</td>
<td>$\tau_{2,2}$</td>
</tr>
<tr>
<td>$\tau_{1,5}$</td>
<td>$\left{\begin{array}{ll} 8 &amp; 0.6 \ 8 &amp; 0.4 \end{array}\right.$</td>
<td>$\tau_{2,2}$</td>
<td>$\left{\begin{array}{ll} 9 &amp; 0.6 \ 12 &amp; 0.4 \end{array}\right.$</td>
<td>$\tau_{2,2}$</td>
</tr>
<tr>
<td>$\tau_{2,1}$</td>
<td>19</td>
<td>$\emptyset$</td>
<td>19</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$\tau_{2,2}$</td>
<td>19</td>
<td>$\emptyset$</td>
<td>19</td>
<td>$\tau_{2,2}$</td>
</tr>
</tbody>
</table>

We note that the pWCRT of task $\tau_1$ is equal to $\left(\begin{array}{ll} 26 & 30 \\ 0.6 & 0.4 \end{array}\right)$. However, if we use the approach adopted by Palencia et al. [8], we find $R_1 = 46$ with the two possible values for $C_{1,5}$. Hence, we observe that our analysis helps to reduce pessimism when estimating the WCRT of the DAG task. On the other hand, by using Fonseca et al. [7] approach, we find $R_1 = 26$ for $C_{1,5} = 2$ and $R_1 = 31$ for $C_{1,5} = 7$. However, our analysis is faster than [7] approach. This latter found the result in 0.3 seconds, while our analysis require only 0.002 seconds.
IV. PRIORITY ASSIGNMENT

In general, fixed-priority scheduling define priorities at task level according to one of the priority assignment algorithms from the state-of-the-art, such as Rate Monotonic [11], Deadline Monotonic [12] and Audsley’s algorithm [13]. Applying a fixed-priority policy at task level to DAG tasks imposes to any sub-task from the task \( \tau_i \) to have a higher priority than all sub-tasks of \( \tau_j \), if a \( \tau_i \) has a higher priority than \( \tau_j \). Since all sub-tasks from the same DAG have the same priority, they may have an arbitrary order of execution. Depending on the structure of the dependency graph and the core mapping, the execution order of sub-tasks may promote parallel executions. Hence, it has an impact on the response time of the whole graph. We illustrate this impact through an example in Figure 3.

In this paper, we propose a priority assignment algorithm. This algorithm defines priorities for sub-tasks from the same DAG. Then, we should apply it to each DAG on our task model. Our proposed method operates on task models with deterministic as well as those with probabilistic execution times by using the expected value of the distribution instead of a deterministic value.

A. Motivation Example

In order to illustrate the purpose of defining priority at sub-task level, we give an example of two schedulings with different sub-task priorities of the DAG task defined by Figure 1 and Table I. We assume that sub-tasks \( \tau_{1,1}, \tau_{1,2} \) and \( \tau_{1,5} \) run on the same core \( \pi_1 \) and the rest of sub-tasks run on a second core \( \pi_2 \). For sub-task \( \tau_{1,5} \), we consider the expected value of its pWCET which is \( E(C_{1,5}) = 4 \). After the execution of \( \tau_{1,1} \), sub-tasks \( \tau_{1,2} \) and \( \tau_{1,5} \) become ready for execution on core \( \pi_1 \). If we apply HLFET (Highest Levels First with Estimated Times) heuristic [14], sub-task \( \tau_{1,5} \) would have higher priority than \( \tau_{1,2} \) and it would start execution before \( \tau_{1,2} \), as in the first scheduling (Figure 3a). In this case, we note that the response time will be equal to \( R_1 = 10 \). On the other hand, if we start with \( \tau_{1,2} \) (Figure 3b), the response time will be equal to \( R_1 = 8 \). We conclude that the execution order of sub-tasks could influence the response time of a DAG task by exploiting or not the possible parallelism in the precedence graph.

B. Sub-task Priority Assignment Algorithm

Our assignment algorithm prioritizes a sub-task with the maximum successor workload that execute on a different core than the sub-task itself. Because 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 [15]. In fact, we split a graph into levels that respect precedence constraints and we prioritize the sub-task that belongs to previous level. This strategy, give higher priority to a predecessor sub-task than its successors which is coherent with precedence constraints.

The pseudo code in algorithm II illustrates how the priority assignment works. First, we calculate, for each sub-task \( \tau_{i,j} \), the sum of execution time averages of successors \( \text{succ}(\tau_{i,j}) \) that execute on a different core than \( \tau_{i,j} \) (lines 2-8). Then, we separate sub-tasks of \( \tau_i \) into levels (line 9). Finally, we sort sub-tasks (line 10) in decreasing order of sum of successor workload. We break tie by comparing sub-tasks levels.

Algorithm 1: Priority assignment algorithm

<table>
<thead>
<tr>
<th>Data:</th>
<th>Tasks ( \tau_i ) and ( \pi ) set of ( m ) cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result:</td>
<td>Sub-tasks priorities</td>
</tr>
<tr>
<td></td>
<td>( \text{suc_sum} = \text{zeros}(n_i) )</td>
</tr>
<tr>
<td></td>
<td>for ( \tau_{i,j} \in \tau_i ) do</td>
</tr>
<tr>
<td></td>
<td>for ( \tau_{i,l} \in \text{suc}(i,j) ) do</td>
</tr>
<tr>
<td></td>
<td>if ( \pi(\tau_{i,l}) \neq \pi(\tau_{i,j}) ) then</td>
</tr>
<tr>
<td></td>
<td>( \text{suc_sum}[j] = \text{suc_sum}[j] + E(C_{i,l}) )</td>
</tr>
<tr>
<td></td>
<td>end</td>
</tr>
<tr>
<td></td>
<td>end</td>
</tr>
<tr>
<td></td>
<td>end</td>
</tr>
<tr>
<td></td>
<td>levels = \text{topologic_order}(\tau_i)</td>
</tr>
<tr>
<td></td>
<td>( \text{Priority} = \text{argsort}(\tau_i, \text{order} = [-\text{suc_sum}, \text{levels}]) )</td>
</tr>
<tr>
<td></td>
<td>return \text{Priority}</td>
</tr>
</tbody>
</table>

V. EVALUATION RESULTS

In this section, we evaluate different proposed algorithms on randomly generated task sets, then, on a real use case. We generate 100 sets of 5 DAG tasks and with 100 sub-tasks in total and we assume that we have 4 cores. We use “randfixesum” algorithm [16] to generate task utilization for each task. We also use “log-uniform” distribution to generate tasks periods in the range \([10, 1000 \text{ ms}]\). Then, we set deadlines to their task periods \( D_i = T_i \). We calculate the execution time according to the formulas: \( C_i = T_i \times U_i \). We generate several sub-tasks per DAG task while their execution time sums up to
the total execution time $C_i$. In addition, we use layer by layer method [17] to generate DAGs with unbiased structure.

**A. 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 find a probability of schedulability of 100%. We note that, in Figure 4, none of the generated task sets reaches the probability of 100% so they won’t be schedulable using deterministic analysis. However, about half of task sets have a high schedulability probability ($\geq 80\%$) which highlights the pessimism of deterministic analysis.

![Schedulability probability histogram](image)

Fig. 4: schedulability of 100 generated task sets

**B. Priority Assignment Algorithm**

Here, we compare our priority assignment algorithm to some existing algorithm like HLFET (Highest Levels First with Estimated Times), SCEFT (Smallest Co-levels First with Estimated Times) and CPMISF (Critical Path/Most Immediate Successors First) heuristics [14, 18]. First, we compute the response time of each DAG task using our proposed RTA. Then, we use Simso which is a simulation tool developed by Chéramy et al. [19] to evaluate real-time scheduling algorithms. It supports several models and scheduling policies. However, it does not include DAG task model, priority on sub-task level and probabilistic execution time. Therefore, we adapted the source code[1] to be able to simulate our task model and to derive the response time from the simulator events log. Table III shows the ratio of the WCRT obtained by each heuristic over the one obtained by our priority assignment algorithm. We note that our method reduces both computed and simulated response time compared to other heuristics.

<table>
<thead>
<tr>
<th>WCRT Analysis</th>
<th>HLFET</th>
<th>SCEFT</th>
<th>CPMISF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simso simulation</td>
<td>105, 04%</td>
<td>113, 35%</td>
<td>104, 04%</td>
</tr>
</tbody>
</table>

**C. 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[2]. The structure of the DAG tasks is illustrated in Figure 5.

![DAGs describing precedence constraints between sub-tasks of the three tasks representing PX4 Autopilot programs.](image)

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[3] on top of a NuttX OS[4]. The execution time measurements of the sub-tasks are obtained by executing them on a single core processor (ARM family) with a highest priority in order to avoid any preemptions from other sub-tasks. In order to obtain the probabilistic bounds, we extracted from each empirical distribution several quartiles. The execution times traces will be provided to the reader to ensure the reproducibility of our results.

**TABLE IV: Comparison of computed DMP and drone behavior**

<table>
<thead>
<tr>
<th>Periods</th>
<th>Drone behavior</th>
<th>DMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 ms</td>
<td>Could not fly</td>
<td>0.9999</td>
</tr>
<tr>
<td>3.5 ms</td>
<td>Could not fly</td>
<td>0.994</td>
</tr>
<tr>
<td>4 ms</td>
<td>Poor stability</td>
<td>0.2696</td>
</tr>
<tr>
<td>4.5 ms</td>
<td>Medium stability</td>
<td>0.0019</td>
</tr>
<tr>
<td>5 ms</td>
<td>Good stability</td>
<td>$1.4959 \times 10^{-14}$</td>
</tr>
</tbody>
</table>

First, we compute various DMP by setting the period of the PX4 autopilot with different values. Then, we compare them to

https://en.wikipedia.org/wiki/PX4_autopilot
https://pixhawk.org
http://nuttx.org
drone behavior already evaluated with different period values. Results are illustrated in Table V. 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 \((D_i = T_i)\). In the other hand, DMP is reduced to \(10^{-14}\) when period is not too small and the drone shows a good stability.

### Table V: DMP of PX4 autopilot tasks under dual core processor with different period configurations

<table>
<thead>
<tr>
<th>(T_1)</th>
<th>(T_2)</th>
<th>(T_3)</th>
<th>DMP (\tau_1)</th>
<th>DMP (\tau_2)</th>
<th>DMP (\tau_3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 ms</td>
<td>7 ms</td>
<td>10 ms</td>
<td>0</td>
<td>0.1536</td>
<td>0</td>
</tr>
<tr>
<td>3 ms</td>
<td>7 ms</td>
<td>10 ms</td>
<td>2.7 \times 10^{-5}</td>
<td>0.9147</td>
<td>0</td>
</tr>
<tr>
<td>3 ms</td>
<td>6 ms</td>
<td>10 ms</td>
<td>2.7 \times 10^{-5}</td>
<td>0.9993</td>
<td>0</td>
</tr>
<tr>
<td>3 ms</td>
<td>6 ms</td>
<td>7 ms</td>
<td>2.7 \times 10^{-5}</td>
<td>0.9993</td>
<td>0.0006</td>
</tr>
<tr>
<td>3 ms</td>
<td>6 ms</td>
<td>7 ms</td>
<td>2.7 \times 10^{-5}</td>
<td>0.9993</td>
<td>0.0006</td>
</tr>
<tr>
<td>2 ms</td>
<td>4 ms</td>
<td>5 ms</td>
<td>0.7082</td>
<td>0.9999</td>
<td>0.9271</td>
</tr>
<tr>
<td>4 ms</td>
<td>2 ms</td>
<td>5 ms</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5 ms</td>
<td>2 ms</td>
<td>5 ms</td>
<td>5.5 \times 10^{-6}</td>
<td>0</td>
<td>0.0451</td>
</tr>
</tbody>
</table>

Now, we assume that the three tasks of the PX4 autopilot (Figure 5) is scheduled on a dual core processor. Then, we compute their DMP to study the schedulability on such hardware architecture. 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 V for different periods combinations. Since priorities of tasks are defined by rate monotonic, all programs of task \(\tau_1\) have higher priorities than \(\tau_2\) in the first six experiment in Table V. We note that DMPs of the three tasks increase as we decrease periods. For the two last experiments, we inverse the priorities of \(\tau_1\) and \(\tau_2\) by choosing \(T2 < T1\). 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 \(\tau_2\). 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 dual core processor, allows to reach a more reactive system with smaller periods.

### VI. Related Work

In this section, we present related work on the scheduling and the schedulability of DAG tasks on multi-cores. The DAG tasks scheduled on several processors have received recently much attention from the real-time community [20–23].

Most of these results consider DAGs to model precedence constraints between tasks, except for [24] that consider that each task is described by a DAG and the vertices of the DAG are the sub-tasks of the task described by the DAG. While the authors of [24] consider global scheduling, Fonseca et al. [7] are studying the response time for sub-tasks scheduled on identical processors according to a partitioned policy. Moreover, Fonseca et al. use self-suspending task [25] to model the DAG task. Then, they estimate the response time of each task by solving a mixed ILP problem. This approach provides a good estimation of response time compared to existing works but it is not scalable for higher number of tasks due to the complexity of the MILP to solve.

Another recent work on partitioned DAG scheduling is proposed in [26], but it considers a single DAG with multiple tasks inside. To analyze this system, the authors propose to unfold (duplicate node) the graph within an hyper-period obtaining a single graph with a single period. This transformation could explode the size of the graph and makes the analysis not scalable. Besides, this analysis is not flexible since it operates offline to generate a scheduling table.

In addition, partitioned scheduling of parallel tasks is studied in the context of distributed system. Tindell and Clark [27] propose an end-to-end RTA of several independent sequences of sub-tasks. These sequences can execute in parallel on multiprocessor platform. This holistic approach was refined later by Palencia et al. [8]. It is used, now, in MAST tool [28] to analyze multi-path end-to-end flows. This approach is pessimistic since it assumes that a worst-case scenario occurs at each activation of a sub-task from the sequence.

Concerning the priority assignment to sub-tasks of DAG tasks, several priority assignments have been proposed in the past like HLFET, SCFET [14] and CPMISF [18]. We compare our solution to each of these existing work.

Our contribution covers also the case of sub-tasks with the probabilistic bounds on the execution times. To our best knowledge, our contribution is the first result in the real-time literature proposing RTA for such task model.

### VII. Conclusion

In this paper, we tackle the problem of partitioned scheduling analysis of tasks with precedence constraints defined by multiple DAGs. We develop the fixed-point response time equations inspired from Palencia results [8] to fit our problem and we extend them to tasks with variable execution and communication times by considering probability distributions. We propose to assign the priorities to sub-tasks of DAG tasks as well and we prove 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. Our approach enabled us to reduce task periods and make the system more reactive. In addition, by reordering tasks we further reduced the response time.

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. Moreover, we will consider a comparison between global scheduling and partitioned scheduling policies.

REFERENCES
