CISTER
Research Centre in
Real-Time & Embedded
Computing Systems

Technical Report

Timing Analysis for DAG-based and GFP Scheduled Tasks

José Marinho

Stefan M. Petters

CISTER-TR-181125

Timing Analysis for DAG-based and GFP Scheduled Tasks

José Marinho, Stefan M. Petters

*CISTER Research Centre

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail:

http://www.cister.isep.ipp.pt

Abstract

Modern embedded systems have made the transition from single-core to multi-core architectures, providing performance improvement via parallelism rather than higher clock frequencies. DAGs are considered among the most generic task models in the real-time domain and are well suited to exploit this parallelism. In this paper we provide a schedulability test using response-time analysis exploiting exploring and bounding the self interference of a DAG task. Additionally we bound the interference a high priority task has on lower priority ones.

Timing Analysis for DAG-based and GFP Scheduled Tasks

José Marinho *, Stefan M. Petters *
*CISTER-ISEP Research Centre, Polytechnic Institute of Porto, Portugal
Email: {jmsm,smp}@isep.ipp.pt

***Abstract*—Modern embedded systems have made the transition from single-core to multi-core architectures, providing performance improvement via parallelism rather than higher clock frequencies. DAGs are considered among the most generic task models in the real-time domain and are well suited to exploit this parallelism. In this paper we provide a schedulability test using response-time analysis exploiting exploring and bounding the self interference of a DAG task. Additionally we bound the interference a high priority task has on lower priority ones.**

I. Introduction

The strive for higher computational power has brought about the multicore platforms as a compelling solution first in general purpose and now also in the embedded real-time systems arena. Rather than relying on the increase of the throughput of single processors, the multicore paradigm, while providing its ability to perform a greater number of simultaneous calculations, has given rise to a new challenge. It often forces system designers to utilize the hardware facilities and use parallel algorithms in order to perform tasks of high computational demand in a predefined time window. However, this implies a subtle difference in the way schedulability conditions are posed since parts of the workload from the same task are allowed to execute concurrently; each task is then referred to as a *parallel* or *Directed Acyclic Graph* (DAG) task. This paper presents a framework to address this issue for fully preemptive *global fixed task priority* (GFP) schedulers and homogeneous multicores in which all cores have the same computing capabilities and are interchangeable. It is worth to mention that GFP schedulers are commonly adopted and supported out of the box on several industry grade real-time operating systems such as VXWorks [11].

a) Related Work: Valuable works such as [5], [12], [1], [9], [4] addressed the scheduling problem of DAG tasks upon homogeneous multicores. Saifullah et al. [12] presented a method to decompose a generic DAG task into a set of virtual sequential tasks and after the decomposition, the popular global earliest deadline first (GEDF) density-based schedulability test is applied. Andersson and Niz [1] presented an analysis for GEDF where an upper bound on the workload that each task may execute in a given time window is computed. Nevertheless, this upper-bound is computed for a special case of DAG tasks, namely the "fork-join" tasks. For such a task: (i) the parallel workloads have the same execution requirement; (ii) they are spawned after a common point; and (iii) they join again after a common point. Note: When a task is executing a section of workload in parallel no further path forks can occur. Chwa et al. [5] provided a method to compute the interference that each task would suffer in a system of so-called "synchronous parallel" tasks – Each task is composed of multiple and potentially contiguous regions of parallel workloads with distinct parallelism levels –. In more than one aspect DAG tasks cover a broader area as they allow for parallel workloads to yield distinct execution requirements and a different immediate predecessor for each node. Previous works using GFP schedulers exist, but in a partitioned environment, i.e., tasks are assigned to cores at design time and no migration is allowed at runtime [6], [8]. For example, Lakshmanan et al. [8] presented a basic form of DAG tasks, namely "Gang tasks", in which all the parallel workloads have to be scheduled simultaneously on the processing platform.

b) This Research: In this paper, we present a sufficient schedulability test applicable to constrained deadline DAG tasks (see Section II for a formal definition) scheduled by using a GFTP scheduler on a homogeneous multicore platform.

II. System Model

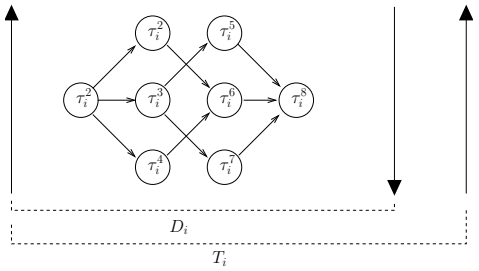

Fig. 1: Task τ_i

c) Task specifications: We consider a task-set $\mathcal{T} \stackrel{\text{def}}{=} \{\tau_1, \ldots, \tau_n\}$ composed of n sporadic tasks. Each sporadic task $\tau_i \stackrel{\text{def}}{=} \langle G_i, D_i, T_i \rangle$, $1 \leq i \leq n$, is characterized by a DAG G_i, a *relative deadline* D_i and a *minimum timespan* T_i (also called period) between two consecutive activations of τ_i. These parameters are given with the following interpretation. Nodes in G_i (also called sub-jobs in the literature) stand for a vector of execution requirements at each activation of τ_i, and the edges represent dependencies between the nodes. A node is denoted by τ_i^j, with $1 \leq j \leq n_i$, where n_i is the total number of nodes in G_i. The execution requirement of node τ_i^j is denoted by $c_i^j \in [c_{i,\min}^j, c_{i,\max}^j]$. A direct edge from node τ_i^j to node τ_i^k, denoted as $\tau_i^j \rightarrow \tau_i^k$, implies

that the execution of τ_i^k cannot start unless that of τ_i^j has completed. In this case, τ_i^j is called a *parent* of τ_i^k, while τ_i^k is its child. We denote the *set of all children* of node τ_i^j by $succ(\tau_i^j) \stackrel{\text{def}}{=} \{\tau_i^k \mid \tau_i^j \rightarrow \tau_i^k\}$ and the *set of all parents* of node τ_i^j by $pred(\tau_i^j) \stackrel{\text{def}}{=} \{\tau_i^k \mid \tau_i^k \rightarrow \tau_i^j\}$. If $(\tau_i^k \notin succ(\tau_i^j) \cup pred(\tau_i^j)) \wedge (\tau_i^j \notin succ(\tau_i^k) \cup pred(\tau_i^k))$, then τ_i^j and τ_i^k may execute concurrently. In this case, we state that $\tau_i^k \in conc(\tau_i^j)$, and reversely, $\tau_i^j \in conc(\tau_i^k)$. A node without parent is called an *entry node*, while a node without child is called an *exit node*. We assume that a node can start executing only after all its parents have completed. For brevity sake, we consider *only* DAG tasks with a single entry and exit nodes. For each task τ_i, we assume $D_i \leq T_i$, which is commonly referred to as the constrained deadline task model. Figure 1 illustrates a DAG task τ_i with $n_i = 8$ nodes. Note: the analysis presented in this paper is easily tunable for DAG tasks with multiple entry and exit nodes.

The *total execution requirement* of τ_i, denoted by C_i, is the *sum* of the execution requirements of all the nodes in G_i, i.e., $C_i \stackrel{\text{def}}{=} \sum_{j=1}^{n_i} c_i^j$. The task set $\mathcal{T}$ is said to be $\mathcal{A}$-*schedulable*, if $\mathcal{A}$ can schedule $\mathcal{T}$ such that all the nodes of every task $\tau_i \in \mathcal{T}$ meet its deadline D_i.

Definition 1 (Critical path)**.** *A critical path for task τ_i, denoted by $\mathcal{P}_i^{\text{crit}}$, is a directed path that has the maximum execution requirement among all paths in G_i.*

Definition 2 (Critical path length)**.** *The critical path length for task τ_i, denoted by C_i^{crit}, is the sum of execution requirements of the nodes belonging to a critical path in G_i.*

d) Platform and scheduler specifications: We consider a platform $\pi \stackrel{\text{def}}{=} [\pi_1, \pi_2, \ldots, \pi_m]$ consisting of m-unit capacity cores, and a *fully preemptive* GFP scheduler. That is: (i) a priority is assigned to each DAG task at system design-time and then, at run-time, every node inherits the priority of the DAG task it belongs to; (ii) different nodes of the same DAG task may execute upon different cores; and finally (iii) a preempted node may resume execution upon the same or a different core, at no cost or penalty. We assume that each node may execute on at most one core at any time instant and that the lower the index of a task the higher its priority.

III. Timing Analysis and Self-Interference Extraction

Intrinsically, some nodes of a given DAG task τ_i may prevent some others of the same task from executing. This constitutes a form of self-interference. Since G_i may be viewed as a set of paths, say $\mathcal{P}_i$, each path $\mathcal{P}_i^k \in \mathcal{P}_i$ represents a set of sequential nodes in G_i connected to each other via an edge, i.e., from the view-point of any node of $\mathcal{P}_i^k$, the other nodes of $\mathcal{P}_i^k$ are either children or parents. We denote the complementary set of $\mathcal{P}_i^k$ which contains all the nodes that do not belong to $\mathcal{P}_i^k$ by $\overline{\mathcal{P}_i^k}$. Note: the nodes in $\overline{\mathcal{P}_i^k}$ are not necessarily concurrent to all the nodes in $\mathcal{P}_i^k$.

Let $\mathcal{P}(\tau_i^\ell, \tau_i^r)$ be the set of all partial paths in G_i which connect nodes τ_i^ℓ and τ_i^r, and let $\mathcal{P}_i^k(\tau_i^\ell, \tau_i^r) \in \mathcal{P}(\tau_i^\ell, \tau_i^r)$ be a specific path. For brevity sake, we denote $\mathcal{P}_i^k(\tau_i^\ell, \tau_i^r)$ by $\mathcal{P}_i^{k(\ell,r)}$ for the remainder of this paper. Since $\tau_i^r \in succ(\tau_i^\ell)$ by definition of $succ(\cdot)$, each path $\mathcal{P}_i^{k(\ell,r)}$ has a worst-case execution requirement $C_i^{k(\ell,r)}$ which is computed by summing up the execution requirements of all its nodes, i.e., $C_i^{k(\ell,r)} \stackrel{\text{def}}{=} \sum_{\tau_i^j \in \mathcal{P}_i^{k(\ell,r)}} c_i^j$. Note: $\mathcal{P}(\tau_i^\ell, \tau_i^r)$ also has a critical path defined as in Definition 1, i.e., the path with the largest execution requirement between τ_i^ℓ and τ_i^r. It is fairly straightforward that if either τ_i^ℓ or τ_i^r is not part of the end-to-end critical path $critpi$ then it follows that the critical partial path between τ_i^ℓ and τ_i^r is not contained in $\mathcal{P}_i^{\text{crit}}$ either. Now we can quantify the maximum self-interference that a DAG task may generate on a given subset of G_i.

Let R_i denote the worst-case response time of the DAG task τ_i – The response time of every activation of τ_i is the timespan between its workload completion and its release – Hence R_i is the largest value from all the activations of τ_i. On the roadway for the computation of an upper-bound on R_i, $\forall i \in [1, n]$, there are some important checkpoints we must investigate.

Definition 3 (partial worst-case response time)**.** *The partial worst-case response time of the set of partial paths $\mathcal{P}(\tau_i^\ell, \tau_i^r)$ is the largest timespan between node τ_i^r completion time and node τ_i^ℓ release time.*

Lemma 1 (Critical Self-interference Path)**.** *Considering only self-interference, the partial path of $\mathcal{P}(\tau_i^\ell, \tau_i^r)$ which leads to the worst-case response time of τ_i is the critical partial path $\mathcal{P}_i^{\text{crit}(\ell,r)}$ among all partial paths in $\mathcal{P}(\tau_i^\ell, \tau_i^r)$.*

Proof (made by contradiction): Initially $C_i^{\text{crit}(\ell,r)} \geqslant C_i^{d(\ell,r)}$ for any other partial path $\mathcal{P}_i^{d(\ell,r)}$. Baker and Cirinei [2] provided an upper-bound on the interference of a *Liu & Layland (LL) task* (in the LL model, each task τ_i generates a potentially infinite sequence of jobs and is characterized by a 3-tuple $\tau_i = \langle C_i, D_i, T_i \rangle$, where C_i is the worst-case execution time of each job, D_i is the relative deadline and $T_i \geq D_i$ is the *minimum* inter-arrival time between two consecutive jobs of τ_i) on a m-multicore platform ($m > 1$). In this work, we extend this result to compute the interference that concurrent nodes induce on $\mathcal{P}_i^{\text{crit}(\ell,r)}$ in the same manner (see Eq. 1).

$$C_i^{\text{crit}(\ell,r)} + \tfrac{1}{m} \sum_{\tau_i^j \in \overline{\mathcal{P}_i^{\text{crit}(\ell,r)}}} c_i^j \tag{1}$$

Let us assume that for some $\mathcal{P}_i^{d(\ell,r)} \neq \mathcal{P}_i^{\text{crit}(\ell,r)}$ we have:

$$C_i^{\text{crit}(\ell,r)} + \tfrac{1}{m} \sum_{\tau_i^j \in \overline{\mathcal{P}_i^{\text{crit}(\ell,r)}}} c_i^j < C_i^{d(\ell,r)} + \frac{1}{m} \sum_{\tau_i^j \in \overline{\mathcal{P}_i^{d(\ell,r)}}} c_i^j \tag{2}$$

Then it follows that:

$$\sum_{\tau_i^j \in \mathcal{P}_i^{\text{crit}(\ell,r)}} c_i^j + \frac{1}{m} \sum_{\tau_i^j \in \overline{\mathcal{P}_i^{\text{crit}(\ell,r)}}} c_i^j < \sum_{\tau_i^j \in \mathcal{P}_i^{d(\ell,r)}} c_i^j + \frac{1}{m} \sum_{\tau_i^j \in \overline{\mathcal{P}_i^{d(\ell,r)}}} c_i^j \tag{3}$$

Since

$$\sum_{\tau_i^j \in \overline{\mathcal{P}_i^{\text{crit}(\ell,r)}}} c_i^j - \sum_{\tau_i^j \in \overline{\mathcal{P}_i^{d(\ell,r)}}} c_i^j = \sum_{\tau_i^j \in \mathcal{P}_i^{d(\ell,r)}} c_i^j - \sum_{\tau_i^j \in \mathcal{P}_i^{\text{crit}(\ell,r)}} c_i^j \tag{4}$$

By substituting Eq. (4) into Eq. (3), Eq. (2) leads us to:

$$\sum_{\tau_i^j \in \mathcal{P}_i^{\text{crit}(\ell,r)}} c_i^j - \frac{1}{m} \sum_{\tau_i^j \in \mathcal{P}_i^{\text{crit}(\ell,r)}} c_i^j < \sum_{\tau_i^j \in \mathcal{P}_i^{d(\ell,r)}} c_i^j - \frac{1}{m} \sum_{\tau_i^j \in \mathcal{P}_i^{d(\ell,r)}} c_i^j \quad (5)$$

which trivially means $C_i^{\text{crit}(\ell,r)} < C_i^{d(\ell,r)}$, contradicting the initial assumption. The Lemma follows. ∎

Informally speaking Lemma 1 infers, for any non-parallel pair of fringe nodes τ_i^ℓ and τ_i^r, that an upper-bound on the response time of τ_i is obtained by considering $\mathcal{P}_i^{\text{crit}(\ell,r)}$ between any τ_i^ℓ and τ_i^r. At the same time, the nodes which do not belong to $\mathcal{P}_i^{\text{crit}(\ell,r)}$ are assumed to induce the maximum interference over it. As this is proven for any pair of nodes, the result also holds for the extreme nodes. Now we focus on

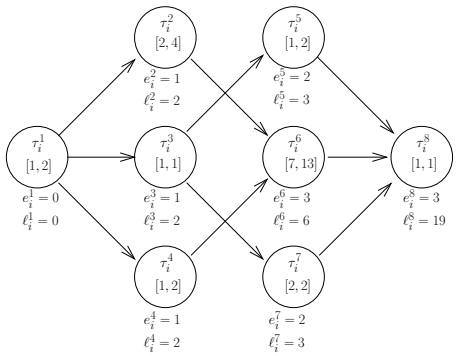

Fig. 2: Earliest and latest release times for nodes in a DAG

deriving the critical path $\mathcal{P}_i^{\text{crit}}$ in G_i. For every node τ_i^j in G_i, we denote by e_i^j and ℓ_i^j its *earliest* and *latest* release times, respectively. Note: These quantities can be computed through a breadth-first [10] traversal of G_i. Assuming τ_i^1 and τ_i^{last} are the entry and exit nodes of τ_i, the earliest release time of any node τ_i^j without any interference can be computed as follows.

$$e_i^1 \stackrel{\text{def}}{=} 0 \quad (6)$$

$$e_i^j \stackrel{\text{def}}{=} \max_{\tau_i^x \in pred(\tau_i^j)} \{e_i^x + c_{i,\min}^x\} \quad (7)$$

where $c_{i,\min}^x$ is the minimum execution requirement of τ_i^x. In the same manner, a breath-first traversal of G_i starting from τ_i^{last} provides the latest release time of τ_i^j as follows.

$$\ell_i^{\text{last}'} \stackrel{\text{def}}{=} 0 \quad (8)$$

$$\ell_i^{j'} \stackrel{\text{def}}{=} \min_{\tau_i^x \in succ(\tau_i^j)} \{\ell_i^{x'}\} - c_{i,\min}^j \quad (9)$$

$$\ell_i^j = \ell_i^{j'} - \ell_i^{1'} \quad (10)$$

Eq. (7) and Eq. (10) clearly represent a *lower-* and an *upper-*bound on the best-case and worst-case start times of node τ_i^j, respectively. This can be observed in the following two scenarios: (i) Node τ_i^j does not suffer any external interference and all its parents request for their minimum execution requirements purveys e_i^j; (ii) Node τ_i^j suffers the maximum possible external interference and its parents request for their maximum execution requirements purveys ℓ_i^j. With these equations, we can derive the worst-case response time of τ_i in isolation, denoted by R_i^{isol}. To do so, without explicitly referring to $\mathcal{P}_i^{\text{crit}}$, we compute the critical path length C_i^{crit} of τ_i as the two problems can be addressed separately. From Eq. (8) and (9) and by starting from the exit node of τ_i, C_i^{crit} is obtained as follows.

$$\text{exe}_i^{\text{last}} \stackrel{\text{def}}{=} 0 \quad (11)$$

$$\text{exe}_i^j \stackrel{\text{def}}{=} \max_{\tau_i^x \in succ(\tau_i^j)} \{\ell_i^x\} - \text{exe}_{\text{i,max}}^j \quad \forall \tau_i^j \in G_i \quad (12)$$

$$C_i^{\text{crit}} \stackrel{\text{def}}{=} \text{exe}_i^1 \quad (13)$$

For any $\tau_i^j \in \mathcal{P}_i^{\text{crit}}$, the execution requirement of the nodes in $conc(\tau_i^j)$ is yielded by $\text{SI}_i \stackrel{\text{def}}{=} C_i - C_i^{\text{crit}}$. From Lemma 1, an upper-bound on the response time of task τ_i, including only the self-interference is thus given by:

$$R_i^{\text{isol}} = C_i^{\text{crit}} + \tfrac{1}{m} \cdot \text{SI}_i \quad (14)$$

IV. Upper-bound on the Interference and Schedulability Condition

In this section we provide an upper-bound on the interference of any DAG task τ_i and we derive a sufficient schedulability condition. To this end, we distinguish between two scenarios: (i) The scenario where τ_i does not suffer any interference from higher priority tasks, and (ii) The scenario where τ_i suffers the maximum possible interference. For brevity sake we assume that all tasks have carry-in at this stage, and will relax this assumption in Section V.

Regarding Scenario (i), we recall that e_i^j is a lower-bound on the release time of node τ_i^j. This leads to an upper-bound function $f_{i,j}^U(t)$ on the workload request of τ_i^j at any time t (see Figure 3) defined as follows.

$$f_{i,j}^U(t) \stackrel{\text{def}}{=} \min\left(\max((t \mod T_i) - e_i^j, 0), c_{i,\max}^j\right) \quad (15)$$

Since the workload request of the DAG task τ_i is the sum over the workload requests of all its nodes, then an upper-bound on the workload request of τ_i at any time t (see Figure 4) is defined as follows.

$$F_i^U(t) \stackrel{\text{def}}{=} \begin{cases} \sum_{\tau_i^j \in G_i} f_{i,j}^U(t) & \text{if } t < T_i - K_i^{\text{crit}} \\ \left\lfloor \frac{t+K_i^{\text{crit}}}{T_i} \right\rfloor \cdot C_i + \sum_{\tau_i^j \in G_i} U_i^j(t) & \text{otherwise} \end{cases} \quad (16)$$

where $U_i^j(t) = f_{i,j}^U\left((t + K_i^{\text{crit}}) \mod T_i\right)$ and $K_i^{\text{crit}} \stackrel{\text{def}}{=} R_i - C_i^{\text{crit}}$.

Regarding scenario (ii), τ_i^1 is assigned to a core at most K_i^{crit} time units after the task is released and τ_i^j is released at most ℓ_i^j time units after node τ_i^1 has started execution. This leads to a lower-bound function $f_{i,j}^L(t)$ on the workload request of τ_i^j at any time t (see Figure 3) defined as follows.

$$f_{i,j}^L(t) \stackrel{\text{def}}{=} \min\left(\max((t \mod T_i) - \ell_i^j, 0), c_{i,\max}^j\right) \quad (17)$$

A lower-bound on the workload request of τ_i at any time t (see Figure 4) is thus defined as follows.

$$F_i^L(t) \stackrel{\text{def}}{=} \begin{cases} \sum_{\tau_i^j \in G_i} f_{i,j}^L(t) & \text{if } t < T_i - K_i^{\text{crit}} \\ \left\lfloor \frac{t+K_i^{\text{crit}}}{T_i} \right\rfloor \cdot C_i + \sum_{\tau_i^j \in G_i} L_i^j(t) & \text{otherwise} \end{cases} \quad (18)$$

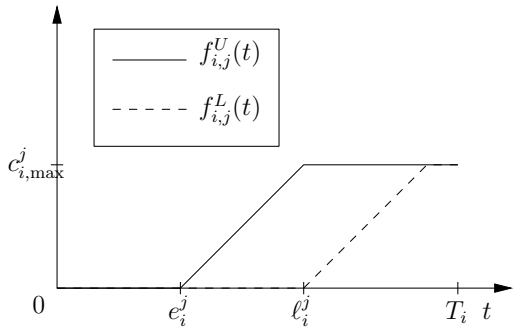

Fig. 3: Extreme cases for node τ_i^j execution requirements

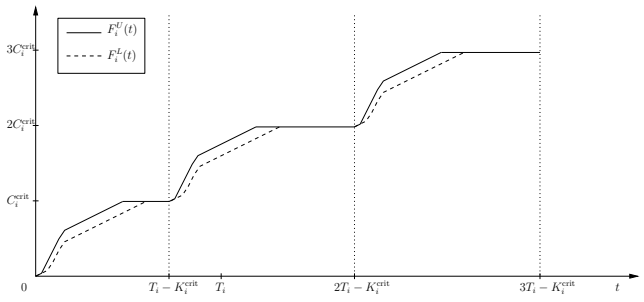

Fig. 4: Extreme cases for DAG task τ_i execution requirements

where $L_i^j(t) = f_{i,j}^L\left((t + K_i^{\text{crit}}) \mod T_i\right)$.

Eq. (16) and Eq. (18) can be used to obtain an upper-bound on the workload request of τ_i in a time window of length Δ. To this end, we consider that an activation of τ_i occurs ϕ time units prior to the beginning of the targeted window. Then two situations can lead to increasing the workload request of τ_i in the window: 1) At the beginning of the window, say at time 0, τ_i suffers the maximum possible interference and its nodes are released as late as possible; 2) At the end of the window, say at time Δ, τ_i does not suffer any interference and its nodes are released as early as possible.

Lemma 2 (Upper-bound on the Workload of τ_i with Carry-in)**.** *Assuming task τ_i has carry-in, an upper-bound on its workload request in a window of length Δ is given by:*

$$W_i^{\text{CI}}(\Delta) \stackrel{\text{def}}{=} \max_{\phi \in [0, C_i^{\text{crit}}]} \left\{ F_i^U(\Delta + \phi) - F_i^L(\phi) \right\} \tag{19}$$

Proof Sketch: We consider an activation of τ_i occurring at time $t_r = -K_i^{\text{crit}} = -R_i + C_i^{\text{crit}}$. The worst-case scenario for task τ_i is when it is prevented from execution on any core by higher priority tasks in the interval $[-K_i^{\text{crit}}, 0]$. Let us assume this worst-case scenario and let us assume that all nodes $\tau_i^j \in G_i$ are released at time ℓ_i^j but one specific node τ_i^k is released at time $\ell_i^{k'} < \ell_i^k$. Since $f_{i,k}^L(t)$ is a lower-bound on the workload request of τ_i^k at any time t, it follows that the workload executed after t, when τ_i^k is released at time ℓ_i^k, is greater than or equal to the workload request of τ_i^k when it is released at time $\ell_i^{k'}$. Hence on the left border of the window of length Δ (i.e., at the beginning of the window), if the nodes are assumed to be released as late as possible, then the workload request in the window is maximized. On the right border of the window (i.e., at the end of the window), we assume the earliest release time of all the nodes $\tau_i^j \in G_i$ but one specific node τ_i^k. By applying the same logic, it follows that the workload request in the window is maximized since the nodes are assumed to be released as early as possible and $f_{i,k}^U(t)$ is an upper-bound on the workload request of τ_i^k at any time t.

Now, let n_i^{pmax} denote the maximum number of parallel nodes in G_i. We recall that the summation of the workload requests of all the nodes $\tau_i^j \in G_i$ is a piecewise linear function, where each segment has its first derivative in the interval $[0, n_i^{\text{pmax}}]$. In order to compute the maximum workload request of each DAG task τ_i in an interval of length Δ, we must evaluate the workload request in all windows of length Δ assuming an offset $\phi \geqslant 0$. Since on the one hand the first derivative of $F_i^U(.)$ (resp. the first derivative of $F_i^L(.)$) is clearly periodic from time $T_i - K_i^{\text{crit}}$ with a period T_i (see Fig. 4), and on the other hand, the next activation of τ_i occurs only at time $t_r^{\text{next}} = t_r + T_i = T_i - K_i^{\text{crit}}$, it is not necessary to check the offsets ϕ over C_i^{crit} as there is no extra workload after C_i^{crit} by construction. Hence $\phi \in [0, C_i^{\text{crit}}]$ and the lemma follows. ■

In order to obtain the solution of Eq. (19), instead of exaustively testing all the values of ϕ in the continuous interval $[0, C_i^{\text{crit}}]$, we derive the finite set $V_i(\Delta)$ of offsets ϕ which maximizes it hereafter.

As previously mentioned, both $F_i^U(\cdot)$ and $F_i^L(\cdot)$ are piecewise linear functions. Hence, the set of points where the first derivative of $F_i^U(\cdot)$ increases and the set points where the first derivative of $F_i^L(\cdot)$ decreases should be considered respectively at the left and at the right border of the targeted window of length Δ. The points in these sets maximize the workload request in the window. Formally, let $\Gamma_i(\Delta) \stackrel{\text{def}}{=} \Gamma_i^1(\Delta) \cup \Gamma_i^2(\Delta)$ where $\Gamma_i^1(\Delta) \stackrel{\text{def}}{=} \{\phi \in [0, C_i^{\text{crit}}]$, the first derivative of $F_i^L(\phi)$ increases$\}$ and $\Gamma_i^2(\Delta) \stackrel{\text{def}}{=} \{\phi \in [0, C_i^{\text{crit}}]$, the first derivative of $F_i^U(\Delta + \phi)$ decreases$\}$. Since $0 \leqslant \phi \leqslant C_i^{\text{crit}}$, then for each node τ_i^j, the first derivative of $F_i^L(\cdot)$ can increase only at points ℓ_i^j. Similarly, the first derivative of $F_i^U(\Delta + \ cdot)$ can decrease only at points $kT_i - K_i^{\text{crit}} + e_i^j + c_i^j - \Delta$ such that $k \in \mathbb{N}$ and $\Delta \leqslant kT_i - K_i^{\text{crit}} \leqslant \Delta + C_i^{\text{crit}}$. Therefore $V_i(\Delta)$ can be defined as follows.

$$V_i(\Delta) \stackrel{\text{def}}{=} \bigcup_{\tau_i^j \in G_i} \Big(\Big\{ \ell_i^j \Big\} \cup \Big\{ kT_i - K_i^{\text{crit}} + e_i^j + c_i^j - \Delta, \text{ such that } k \in \mathbb{N} \text{ and } \Delta \leqslant kT_i - K_i^{\text{crit}} \leqslant \Delta + C_i^{\text{crit}} \Big\} \Big) \tag{20}$$

The computation of $W_i^{\text{CI}}(\Delta)$ for each τ_i (with $i \in [1, n]$) makes it easy to assess an upper-bound on the interference it will induce on the workload of the lower priority tasks in any given time window. From [2], it has been proven that every unit of execution of a LL task can interfere for at most $\frac{1}{m}$ units on the workload request of any other LL task with a lower priority. Thus, an upper-bound on the interference suffered by

the DAG task τ_i in a window of size Δ is provided as:

$$I_i(\Delta) \stackrel{\text{def}}{=} \frac{1}{m} \cdot \sum_{j \in hp(\tau_i)} W_j^{\text{CI}}(\Delta) \tag{21}$$

where $hp(\tau_i)$ is the set of tasks with a higher priority than τ_i. A sufficient schedulability condition for a DAG task-set $\mathcal{T}$ is derived from Eq. (21) as follows.

Theorem 1 (Sufficient schedulability condition)**.** *A* DAG *task-set $\mathcal{T}$ is schedulable on a m-homogeneous multicores using a* GFP *scheduler if:*

$$\forall \tau_i \in \mathcal{T}, R_i \leqslant D_i \tag{22}$$

where R_i is computed by the following fixed-point algorithm.

$$\begin{cases} R_i^{\{0\}} = R_i^{\text{isol}} & \text{if } k = 0 \\ R_i^{\{k\}} = I_i\left(R_i^{\{k-1\}}\right) + R_i^{\text{isol}} & \text{if } k \geqslant 1 \end{cases}$$

Note: This iterative algorithm stops as soon as for any $k \geq 1$, $R_i^{\{k\}} = R_i^{\{k-1\}}$ or $R_i^{\{k\}} > D_i$. In the latter case, τ_i is deemed not schedulable.

Proof: This theorem follows directly from Lemma 1, Lemma 2, Eq. (14) and Eq. (21). ∎

V. Reduction of the number of tasks with carry-in

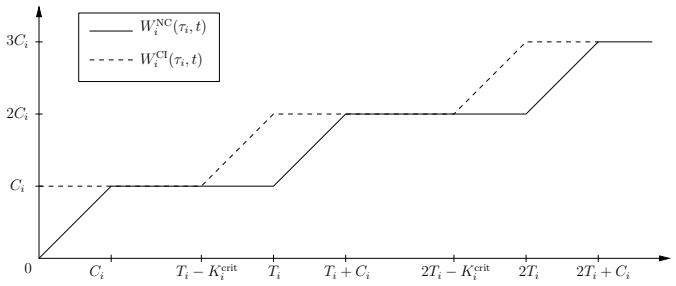

Fig. 5: Functions $W^{\text{NCseq}}(\tau_j, \Delta)$ and $W^{\text{CIseq}}(\tau_j, \Delta)$ for task τ_j

Rather than considering that each DAG task has a carry-in as in Section IV, the intuitive idea of this section consists of reducing the number of tasks with carry-in to at most $(m-1)$ tasks (where m is the number or cores). Since it is usually the case that $m \ll n$, we thus obtain a tighter upper-bound on the interference that each task may suffer at run-time and finally a better schedulability condition for each task. To accomplish this, first let us recall some fondamental results regarding the "Liu & Layland (LL) task model".

Upper-bound on the workload request of a LL task without carry-in. Let τ_j be a LL task with no pending workload at the beginning of a window of length Δ. An upper-bound on its workload request in this window is recalled (see [3], [7]):

$$W^{\text{NC-LL}}(\tau_j, \Delta) = \left\lfloor \frac{\Delta}{T_j} \right\rfloor \cdot C_j + \min(\Delta \mod T_j, C_j) \tag{23}$$

Upper-bound on the workload request of a LL task with carry-in. Let τ_k be a LL task with some pending workload at the beginning of a window of length Δ. An upper-bound on its workload request in this window is recalled (see [3], [7]):

$$\begin{aligned} W^{\text{CI-LL}}(\tau_k, \Delta) &= \left\lfloor \frac{\max(\Delta - C_k, 0)}{T_k} \right\rfloor \cdot C_k + C_k \\ &+ \max\left((\max(\Delta - C_k, 0) \mod T_k) - (T_k - R_k), C_k\right) \end{aligned} \tag{24}$$

Extra workload request of a LL task. The difference between the upper-bounds –*with* and *without*– carry-in of a LL Task τ_i in a window of length Δ is thus recalled as:

$$W^{\text{diff-LL}}(\tau_i, \Delta) \stackrel{\text{def}}{=} W^{\text{CI-LL}}(\tau_i, \Delta) - W^{\text{NC-LL}}(\tau_i, \Delta) \tag{25}$$

Upper-bound on the interference of a LL task. Assume a GFP scheduler and a DAG task-set $\mathcal{T}$ in which tasks are in a decreasing priority order. An upper-bound on the interference that higher priority tasks induce on the execution of task τ_i in a targeted window of length Δ is recalled (see [3], [7]):

$$\begin{aligned} I_i^{\text{LL}}(\Delta) \stackrel{\text{def}}{=} \frac{1}{m} \cdot \Bigg(& \sum_{l=1}^{m-1} \max_{\tau_j \in \{\tau_1, \ldots, \tau_{i-1}\}}^{l} W^{\text{diff-LL}}(\tau_j, \Delta) \Bigg) \\ & + \sum_{\tau_j \in \{\tau_1, \ldots, \tau_{i-1}\}} W^{\text{NC-LL}}(\tau_j, \Delta) \end{aligned} \tag{26}$$

In Eq. (26), $\max_{\tau_h \in \{\tau_1, \ldots, \tau_{i-1}\}}^{l}(\cdot)$ returns the ℓ^{th} greatest value among the workload of tasks with a higher priority than τ_i. For a LL task-set, it has been proven in [7] that a worst-case scenario in terms of total workload request in a targeted window of length Δ can be constructed by considering $(m-1)$ tasks with carry-in. Therefore, it follows that the workload induced by these carry-in tasks in this window of concern cannot exceed the difference between (i) the maximum workload assuming no carry-in for all tasks (see Eq. (23)) and (ii) the workload assuming the carry-in scenario (see Eq. (24)). Consequently, from the view-point of task τ_i, if $i < m$, then τ_i does not suffer any interference, otherwise, if $i \geqslant m$, then we can choose the $(m-1)$ tasks among $\{\tau_1, \ldots, \tau_{i-1}\}$ such that the difference between the workload assuming the non-carry-in scenario and the workload assuming the carry-in scenario is the largest possible for each selected task. By summing up these differences and the remaining "$(i-1)-(m-1) = i-m$" workloads corresponding to the tasks without carry-in, an upper-bound on the workload that higher priority tasks induce in the window of length Δ is computed.

Before we extend Eq. (26) to the scheduling problem of DAG tasks using a GFP scheduler, let us present an alternative formal proof to the one provided by Guan et al. [7] for the analysis considering $(m-1)$ tasks with carry-in.

Theorem 2 (Eq. (26) is an Upper-bound for LL tasks [7])**.** *Let τ be a feasible LL task-set scheduled by using a* GFP *scheduler on a m-homogeneous multicores. Let task $\tau_i \in \tau$. Eq.* (26) *is an upper-bound on the interference on τ_i in any window of length Δ.*

Proof: Since τ is feasible, let t_0 be the latest time-instant such that at least one core is idle at time $t_0 - \epsilon, \forall \epsilon \geqslant 0$, then at most $(m-1)$ tasks have a carry-in workload at time instant $t_0 - \epsilon, \forall \epsilon \geqslant 0$. Let Δ_0 be the window of length Δ starting at

t_0. By considering the $(m-1)$ tasks with the largest possible carry-in, we are conservative w.r.t. the workload request of the tasks with carry-in in Δ_0. In the same vein, by considering $(i-m)$ tasks without carry-in to be simultaneously released at time t_0 with the future activations of each of these tasks to occur as soon as it is legally permitted to do so, we are also conservative w.r.t. the workload request of the tasks without carry-in in Δ_0.

Now let Δ_1 be a window of length Δ starting at time $t_1 \geqslant t_0$ with the offset $\Phi \stackrel{\text{def}}{=} t_1 - t_0$. Assume that the beginning of Δ_1 triggers the first activation of τ_i. The earliest time-instant at which τ_i may start executing is at t_f such that $t_f \geqslant \max(t_1, t_0 + \Delta)$. Indeed: (i) $t_f \geqslant t_1$ (as τ_i cannot start executing before its activation time), and (ii) $t_f \geqslant t_0 + \Delta$ (as all the m cores are busy executing higher priority tasks between t_0 and $t_0+\Delta$), by construction. As all the m cores are busy executing higher priority tasks between t_0 and t_1, getting the first activation of τ_i at any time-instant in the interval $[t_0, t_f]$ (i.e., by sliding Δ_1 towards Δ_0), we can only increase the interference on τ_i (as the end of the execution of τ_i remains unchanged). The maximum interference is obtained when τ_i is activated simultaneously with all higher priority tasks, i.e., at time t_0 as then we have the largest possible carry-in as well as non-carry-in interference on the execution of τ_i. The theorem follows. ∎

VI. Extension to DAG-based Tasks

In this section we extend the reduction of the number of tasks with carry-in obtained in the framework of LL tasks to the DAG task model. To accomplish this end, we distinguish between the upper-bound on the workload request of the tasks *with* carry-in (see Eq. (19)) and *without* carry-in (which is detailed hereafter). These expressions will be considered when computing the interference of higher priority tasks on the execution of every DAG task τ_i in a window of length Δ.

Upper-bound on the workload request of DAG tasks *without* carry-in. Let us assume that τ_i is a DAG task *without* carry-in. An upper-bound on its workload request in a targeted window of length Δ can be constructed by distinguishing between the same two scenarios as those which allowed us to derive Eq. (19) in Section IV.

Regarding Scenario (i) where τ_i does not suffer any interference from higher priority tasks, an upper-bound on the workload request of τ_i at any time t is defined as follows.

$$F_i^{\text{U-NC}}(t) \stackrel{\text{def}}{=} \left\lfloor \frac{t}{T_i} \right\rfloor \cdot C_i + \sum_{\tau_i^j \in G_i} f_{i,j}^U(t \mod T_i) \tag{27}$$

In a similar manner, regarding Scenario (ii) where τ_i suffers the maximum interference from higher priority tasks, an lower-bound on the workload request of τ_i at any time t is defined as follows.

$$F_i^{\text{L-NC}}(t) \stackrel{\text{def}}{=} \left\lfloor \frac{t}{T_i} \right\rfloor \cdot C_i + \sum_{\tau_i^j \in G_i} f_{i,j}^L(t \mod T_i) \tag{28}$$

As for the carry-in tasks case, Eq. (27) and Eq. (28) can be used to obtain an upper-bound on the workload request of τ_i in a time window of length Δ as claimed in Lemma 3.

Lemma 3 (Upper-bound on the Workload of τ_i Without Carry-in)**.** *Assiming no carry-in of task τ_i, an upper-bound on its workload request in a window of length Δ is given by:*

$$W_i^{\text{NC}}(\Delta) \stackrel{\text{def}}{=} \max_{\phi \in [0, C_i^{\text{crit}}]} \left\{ F_i^{\text{U-NC}}(\Delta + \phi) - F_i^{\text{L-NC}}(\phi) \right\} \tag{29}$$

Proof Sketch: The proof sketch of this lemma follows the same reasoning as that of Lemma 2. ∎

From Lemma 2 and Lemma 3, it follows that the difference between the upper-bounds –*with* and *without*– carry-in for a DAG task τ_i in a window of length Δ is can be written as:

$$W^{\text{diff-DAG}}(\tau_i, \Delta) \stackrel{\text{def}}{=} W^{\text{CI}}(\tau_i, \Delta) - W^{\text{NC}}(\tau_i, \Delta) \tag{30}$$

All the results presented so far enable us to present a tighter upper-bound on the interference of a DAG task τ_i together with the corresponding sufficient schedulability condition.

Tighter Upper-bound on the Interference of a DAG Task. Assume a GFP scheduler and a DAG task-set $\mathcal{T}$ in which tasks are in a decreasing priority order as in Section V. An upper-bound on the interference that higher priority tasks induce on the execution of task τ_i in a targeted window of length Δ is obtained as follows.

$$I_i^{\text{DAG}}(\Delta) \stackrel{\text{def}}{=} \frac{1}{m} \cdot \left(\sum_{l=1}^{m-1} \max_{\tau_j \in \{\tau_1, \ldots, \tau_{i-1}\}}^{l} W^{\text{diff-DAG}}(\tau_j, \Delta) \right) + \sum_{\tau_j \in \{\tau_1, \ldots, \tau_{i-1}\}} W^{\text{NC}}(\tau_j, \Delta) \tag{31}$$

Each term in Eq. (31) is explained as the corresponding term in Eq. (26) and a tighter schedulability test follows.

Theorem 3 (Tighter Sufficient Schedulability Condition)**.** *A DAG task-set $\mathcal{T}$ is schedulable on a m-homogeneous multi-cores using a GFP scheduler if:*

$$\forall \tau_i \in \mathcal{T}, R_i \leqslant D_i \tag{32}$$

where R_i is computed by the following fixed-point algorithm.

$$\begin{cases} R_i^{\{0\}} = R_i^{\text{isol}} & \text{if } k = 0 \\ R_i^{\{k\}} = I_i^{\text{DAG}}\left(R_i^{\{k-1\}}\right) + R_i^{\text{isol}} & \text{if } k \geqslant 1 \end{cases}$$

Note: This algorithm also stops as soon as for any $k \geq 1$, $R_i^{\{k\}} = R_i^{\{k-1\}}$ or $R_i^{\{k\}} > D_i$. Again, in the latter case, τ_i is deemed not schedulable.

Proof: The proof of this theorem is similar to that of Theorem 2. The difference here resides in the evaluation of the upper-bound on the workload of tasks without carry-in. Instead of considering a synchronous activation at these tasks at the beginning of the targeted window and assume their subsequent activations to occur as soon as it is legally permitted to do so, the upper-bound has to be computed by using Eq. 29). ∎

VII. Conclusions

In this paper, a sufficient schedulability test for fully preemptive DAG-based tasks with constrained deadlines is presented. A global fixed task priority (GFP) scheduler and a homogeneous multicore platform are assumed. Under these

settings, this work is the first to address this problem to the best of our knowledge. As future work we intend to evaluate the properties of a task model where nodes belonging to each task may execute with different priorities rather than directly inheriting their priority from the task they belong to.

REFERENCES

[1] B. Andersson and D. Niz. Analyzing global-edf for multiprocessor scheduling of parallel tasks. In *OPODIS*, 2012.

[2] T. Baker and M. Cirinei. A unified analysis of global edf and fixed-task-priority schedulability of sporadic task systems on multiprocessors. *Journal of Embedded Computing*, 4(2):55–69, 2010.

[3] M. Bertogna and M. Cirinei. Response-time analysis for globally scheduled symmetric multiprocessor platforms. In *RTSS*, 2007.

[4] V. Bonifaci, A. Marchetti-Spaccamela, S. Stiller, and A. Wiese. Feasibility analysis in the sporadic DAG task model. In *ECRTS*, 2013.

[5] H. Sung Chwa, J. Lee, K. Phan, A. Easwaran, and I. Shin. Global edf schedulability analysis for synchronous parallel tasks on multicore platforms. In *ECRTS*, 2013.

[6] F. Fauberteau, M. Qamhieh, and S. Midonnet. Partitioned scheduling of parallel real-time tasks on multiprocessor systems. In *WIP ECRTS*, 2011.

[7] Nan Guan, Martin Stigge, Wang Yi, and Ge Yu. New response time bounds for fixed priority multiprocessor scheduling. In *RTSS*, 2009.

[8] K. Lakshmanan, S. Kato, and R. Rajkumar. Scheduling parallel real-time tasks on multi-core processors. In *RTSS*, 2010.

[9] J. Li, K. Agrawal, C. Lu, and C. Gill. Analysis of global edf for parallel tasks. In *ECRTS*, 2012.

[10] J. Marinho, V. Nélis, S. Petters, and I. Puaut. Preemption delay analysis for floating non-preemptive region scheduling. In *DATE*, 2012.

[11] Wind River. VxWorks Platforms. http://www.windriver.com/products/product-notes/PN_VE_6_9_Platform_0311.pdf.

[12] A. Saifullah, K. Agrawal, C. Lu, and C. Gill. Multi-core real-time scheduling for generalized parallel task models. In *RTSS*, 2010.