Response-Time Analysis of DAG Tasks under Fixed Priority Scheduling with Limited Preemptions

Maria A. Serrano
Barcelona Supercomputing Center and Technical University of Catalonia, Barceclona, Spain
maria.serranogracia@bsc.es

Alessandra Melani
Scuola Superiore Sant'Anna, Pisa, Italy
alessandra.melani@sssup.it

Marko Bertogna
University of Modena and Reggio Emilia, Modena, Italy
marko.bertogna@unimore.it

Eduardo Quinones
Barcelona Supercomputing Center, Barcelona, Spain
eduardo.quinones@bsc.es

***Abstract*—Limited preemptive (LP) scheduling has been demonstrated to effectively improve the schedulability of fully preemptive (FP) and fully non-preemptive (FNP) paradigms. On one side, LP reduces the preemption related overheads of FP; on the other side, it restricts the blocking effects of FNP. However, LP has been applied to multi-core scenarios only when completely sequential task systems are considered. This paper extends the current state-of-the-art response time analysis for global fixed priority scheduling with fixed preemption points by deriving a new response time analysis for DAG-based task-sets.**

I. Introduction

The introduction of multi-core embedded processors [1], [2] has motivated the development of new schedulability analysis methods that allow providing response time guarantees of different parallel execution models, e.g. the fork/join model [3], the synchronous parallel task model [4], the DAG-based task model [5].

In this paper, we focus on the DAG-based task model, which resembles the tasking model of OpenMP4 [6], the de-facto standard for shared memory parallel programming in high-performance computing (HPC). OpenMP has recently gained a lot of attention in the embedded and real-time domains [7], [8], [9], due to its capability to define explicit sub-tasks and the data dependencies existing among them, which allows expressing very sophisticated types of fine-grained and irregular parallelism. Moreover, OpenMP is supported in the newest multi-core embedded architectures, becoming a firm candidate to develop future real-time embedded systems.

The global fixed priority scheduling of DAG-based task-sets has been analyzed under both fully-preemptive (FP) [10] and fully non-preemptive (FNP) [11] strategies. The FP strategy may lead to prohibitively high preemption overheads, mainly related to task context switches, cache related preemption and migration delays, and network contention costs [12], which degrade schedulability and can potentially cause deadline misses. Accurately accounting for preemption delays is very difficult (if not impossible) due to the "infinite" potential preemption points, i.e., at any execution point of the task. Moreover, some of these points have associated a particularly high overhead, e.g., when the task makes intensive use of local memories. The FNP strategy offers an alternative that avoids preemption related overheads, at the cost of introducing significant blocking effects. For example, a task τ may have access to less cores if there exists another task having a worst-case execution time (WCET) longer than τ's deadline.

The limited preemptive (LP) scheme [13] has been proposed as an effective scheduling scheme that reduces preemption-related overheads of FP, while constraining the amount of blocking of FNP and thus improving schedulability. In LP, preemptions can only take place at certain points during the execution of a task, dividing its execution in *non-preemptive regions* (NPR). So far, the response time analysis of LP has only considered sequential task-set, and has never been applied to parallel execution models.

This paper proposes a response time analysis of DAG-based tasks scheduled under global fixed priority with fixed preemption points. The execution model of LP with fixed preemption points resembles the OpenMP4 tasking model [8], hence the proposed analysis could be potentially applied to provide timing guarantees to OpenMP parallel programs [14]. The predictability of the OpenMP tasking model is not addressed in this paper, and is left as future work.

In global fixed priority scheduling with LP, tasks can be blocked by both higher-priority and lower-priority tasks. In [15], the authors upper-bounded the lower-priority interference in a multi-core system executing sequential tasks by considering the longest NPR of lower-priority tasks. However, in parallel DAG-based task-sets, multiple NPRs from different tasks can execute in parallel on different cores. That is, the precedence constraints defined in the DAG determine the maximum number of cores where a task can be spawn, and so its blocking impact on higher-priority tasks.

Our analysis, which builds upon [10] and [15], determine the interferences and blocking impact of higher- and lower-priority tasks respectively. In order to derive the lower-priority interference, we propose two methods: (1) a pessimistic but easy-to-compute method that upper-bounds the blocking impact based on the task-set's longest NPRs (named *LP-max*); and (2) a tighter but computationally-intensive ILP-based method (named *LP-ILP*) that analyzes which NPRs can actually execute in parallel to refine the blocking estimation.

II. Related Work

In the real-time literature, parallel task models are increasingly being used to deal with the fine-grained execution provided by current parallel programming paradigms. In the sporadic DAG model [16], [5], [10], which is probably the most general parallel model, tasks are represented by means of a directed acyclic graph, where each node corresponds to a sequential piece of code, and edges represent precedence constraints between pairs of nodes. The first attempt to study the similarities of DAGs and parallel programming models such as OpenMP has been recently introduced in [8], [9] and [14], where the authors studied how to construct an OpenMP-DAG considering the tasking semantics of OpenMP4.

Despite the significant amount of work on parallel task models, none of the existing works investigates the potential of combining the LP framework with the current schedulability analyses for DAG task-systems. For sequential task-sets, the limited preemptive approach has

The research leading to these results is funded by the EU project P-SOCRATES (FP7-ICT-2013-10) and the Spanish Ministry of Science and Innovation under contract TIN2015-65316-P.

978-3-9815370-6-2/DATE16/

been proven to be an effective scheduling scheme. We refer to [13] for a complete survey on the existing approaches based on limited preemptive scheduling. Optimized preemption point placement techniques [17], [18] have also been proposed in the literature to reduce the cost of preemption related overheads incurred by a task.

For multi-core systems, two main approaches have been identified to deal with preemptions after the release of a higher-priority task. In the *eager* approach, the first lower-priority task to reach a preemption point is preempted, implying that the preempted task may not be the lowest-priority running one. This strategy is opposite to the *lazy* approach, which delays preemption until a preemption point is reached by the lowest-priority running task. In the latter category, an analysis based on link-based scheduling has been proposed in [19]. Schedulability analyses for global fixed priority scheduling with eager preemptions, which is the focus of this paper, have been proposed in [20], [21]. By assuming a simple task model with a single final non-preemptive region for each task, they showed the significant schedulability improvement that this approach may introduce if task priorities and the length of their final non-preemptive regions are carefully selected. Moreover, they showed that the eager and lazy approaches are incomparable. A full schedulability analysis in the case of eager preemptions and multiple non-preemptive regions has been recently proposed in [15].

III. System Model and Background

A. Task Model

In this paper we consider a task-set composed of n sporadic DAG tasks $\mathcal{T} = \{\tau_1, \cdots, \tau_n\}$ executing according to global fixed-priority scheduling on a platform composed of m identical cores. Each task $\tau_k \in \mathcal{T}$ is represented as a directed acyclic graph $G_k = (V_k, E_k)$. The nodes in $V_k = \{v_{k,1}, \ldots, v_{k,q_{k+1}}\}$ represent *non-preemptive regions* (NPRs) of code (or *task parts* in the OpenMP nomenclature [8]), with a number of potential preemption points equal to $q_k = |V_k| - 1$. Each node $v_{k,j} \in V_k$ is labeled with its *worst-case execution time* (WCET) $C_{k,j}$. Edges in $E_k \subseteq V_k \times V_k$ represent precedence constraints among pairs of nodes in V_k. Each task τ_k releases an infinite sequence of jobs, each of which is separated from the subsequent one by a minimum inter-arrival time of T_k time-units, and has a constrained relative deadline $D_k \leq T_k$.

We assume that tasks are ordered according to their decreasing unique priority, so that τ_i has a higher priority than τ_j if $i < j$. We denote by $hp(k)$ and $lp(k)$ the subsets of tasks with higher and lower priorities than τ_k, respectively. In any time interval of length t, a task τ_k can be preempted by higher-priority tasks at most h_k times, where $h_k = \sum_{\forall \tau_i \in hp(k)} \left\lceil \frac{t}{T_i} \right\rceil$. Therefore, the number of preemptions suffered by task τ_k can be upper-bounded by $p_k = min(q_k, h_k)$.

B. Background

We now review prior work on which we propose a new schedulability analysis for DAG tasks incorporating limited preemptions.

1) Response time analysis for DAG tasks: In [10], the authors considered a fully-preemptive global fixed-priority scheduler and derived the following upper-bound on the response time of a DAG task when conditional nodes are not considered:

$$R_k^{ub} \leftarrow L_k + \frac{1}{m}\big(vol(G_k) - L_k\big) + \left\lfloor \frac{1}{m}\big(I_k^{hp}\big) \right\rfloor \quad (1)$$

The bound is iteratively computed starting from the highest-priority task with the initial value $R_k^{ub} = L_k + \frac{1}{m}(vol(G_k) - L_k)$. In Equation (1), L_k denotes the length of the longest path in the DAG, which also corresponds to the minimum amount of time needed to execute the task on a sufficiently large (possibly infinite) number of processors, while $vol(G_k)$ denotes the volume of the DAG and corresponds to the WCET of the task when executing on a dedicated single-core platform. The factor $\frac{1}{m}(vol(G_k) - L_k)$ upper-bounds the *self-interference* (or intra-task interference), i.e., the interfering contribution from the task itself, and the factor $\left\lfloor \frac{1}{m}(I_k^{hp}) \right\rfloor$ computes the *higher-priority interference* (or inter-task interference) from higher-priority tasks in the system. The term I_k^{hp} in Equation (1) is given by:

$$I_k^{hp} = \sum_{\forall \tau_i \in hp(k)} \mathcal{W}_i(R_k^{ub}) \quad (2)$$

where $\mathcal{W}_i(L)$ is an upper-bound on the workload of an interfering task τ_i in a window of length L.

2) Limited preemptive scheduling on multi-cores: According to the LP scheduling strategy, the execution of a task cannot be suspended until a preemption point is reached. As a consequence, the response time of each task must account not only for the interference from higher-priority tasks, but also for the *lower-priority interference* caused by NPRs of lower-priority tasks blocking the task under analysis.

In [15], the authors derived the *lower-priority interference* that a sequential task can suffer due to lower-priority tasks, considering global fixed-priority scheduling with eager preemptions:

$$I_k^{lp} = \Delta_k^m + p_k \times \Delta_k^{m-1} \quad (3)$$

where p_k is an upper-bound on the number of preemptions suffered by τ_k (see Section III-A), and Δ_k^m and Δ_k^{m-1} are upper-bounds on the lower-priority interference on the first NPR and the p^{th} NPRs ($2 \leq p \leq q_k + 1$) of task τ_k, respectively. In the next section, we describe how to compute such quantities when DAG tasks are considered.

IV. Response Time Analysis of Limited Preemption Global Fixed Priority Scheduler on DAG-Based Task-sets

The response time analysis in Equation (1) can be easily extended to incorporate the impact of the limited preemption strategy on DAG-based task-sets. To do so, the factor that computes the higher-priority interference (see Equation (2)), must be augmented to incorporate the impact of lower-priority interference as presented in Equation (3). Overall, the response time upper-bound can be computed as follows:

$$R_k^{ub} \leftarrow L_k + \frac{1}{m}\big(vol(G_k) - L_k\big) + \left\lfloor \frac{1}{m}\big(I_k^{lp} + I_k^{hp}\big) \right\rfloor \quad (4)$$

With LP, tasks are not only interfered by higher-priority tasks, but also by already started lower-priority tasks whose execution has not reached a preemption point yet, and so cannot be suspended. In the worst-case scenario, when a high-priority task τ_k is released, all the m processors have just started executing the m largest NPRs of m different lower priority tasks. After τ_k started executing, it could be blocked again by at most $m - 1$ lower priority tasks at each preemption point. Therefore, for sequential task-sets, the lower-priority interference is upper-bounded considering: (1) the set of the longest NPR of each lower-priority task and then (2) the sum of the m and $m - 1$ longest NPRs of this set, as computed in [15]. This no longer holds for DAG-based task-sets, because multiple NPRs from the same task can execute in parallel. Next, we present two methods to compute the lower-priority interference in DAG-based task-sets.

A. Blocking Impact of the Largest NPRs (LP-max)

The easiest way of deriving the lower priority interference is to account for the m and $m-1$ largest NPRs among all lower-priority tasks:

$$\Delta_k^m = \sum \max_{\tau_i \in lp(k)}^{m} \left(\max_{1 \le j \le q_{i+1}}^{m} C_{i,j} \right)$$
$$\Delta_k^{m-1} = \sum \max_{\tau_i \in lp(k)}^{m-1} \left(\max_{1 \le j \le q_{i+1}}^{m-1} C_{i,j} \right) \quad (5)$$

where $\sum \max_{\tau_i \in lp(k)}^{m}$ and $\sum \max_{\tau_i \in lp(k)}^{m-1}$ denote the sum of the m and $m-1$ largest values among the NPRs of all tasks $\tau_i \in lp(k)$ respectively, while $\max_{1 \le j \le q_{i+1}}^{m}$ and $\max_{1 \le j \le q_{i+1}}^{m-1}$ denote the m and $m-1$ largest NPRs of a task τ_i. Despite its simplicity, this strategy is pessimistic because it considers that the largest m and $m-1$ NPRs can execute in parallel, regardless of the precedence constraints defined in the DAG.

B. Blocking Impact of the Largest Parallel NPRs (LP-ILP)

The edges in the DAG determine the maximum level of parallelism a task may exploit on m cores, which in turn determines the amount of blocking impacting over higher-priority tasks. This information must therefore be incorporated in the analysis to better upper-bound the lower-priority interference. To do so, we propose a new analysis method that incorporates the precedence constraints among NPRs, as defined by the edges in the DAG, into the LP response-time analysis. Our analysis uses the following definitions:

Definition 1: The *worst-case workload* of a task executing on c cores is the sum of the WCET of the c largest NPRs that can execute in parallel.

Definition 2: The *overall worst-case workload* of a set of tasks executing on m cores is the maximum time used for executing this set in a given *execution scenario*, i.e. fixing the number of cores used for each task.

Given a task τ_k, our analysis derives the lower-priority interference of $lp(k)$ by computing new Δ_k^m and Δ_k^{m-1} factors in a three-step process: (1) Identify the *worst-case workload* of each task in $lp(k)$ when executing on 1 to m cores; (2) Compute the *overall worst-case workload* of $lp(k)$ for all possible *execution scenarios*; and (3) Select the scenario that maximizes the lower-priority interference.

In order to facilitate the explanation of the three steps, the next sections consider a $lp(k)$ composed of four DAG-tasks $\{\tau_1, \tau_2, \tau_3, \tau_4\}$ (see Figure 1), executed on a $m = 4$ core platform. The nodes (NPRs) of $\tau_i \in lp(k)$ are labeled as $v_{i,j}$, with their WCET ($C_{i,j}$) in parenthesis.

1) Worst-case workload of a task: Given a task $\tau_i \in lp(k)$, this step computes an array μ_i of size m, that includes the worst-case workload of τ_i when NPRs are distributed over c cores, being $c = \{1, \cdots, m\}$ the index inside μ_i. Each element $\mu_i[c]$ is computed as follows:

$$\mu_i[c] = \sum \max_{c}^{parallel} \{C_{i,j}\} \quad (6)$$

where $\sum \max_{c}^{parallel}$ is the sum of the c largest NPRs of τ_i that can execute in parallel, maximizing the interference when using c cores. To this aim, the sum must consider the edges of τ_i's DAG to determine which NPRs can actually execute in parallel. Section V presents the algorithm that derives, for each NPR of τ_i, the set of NPRs from the same task that can potentially execute in parallel with it.

Table I shows the array μ_i for each of the tasks shown in Figure 1 with $m = 4$. For example, the worst-case workload $\mu_4[2]$ occurs when NPRs $v_{4,3}$ and $v_{4,4}$ execute in parallel, with an overall impact of 9 time units. τ_2 has a maximum parallelism of 2, so $\mu_2[3]$ and $\mu_2[4]$ are equal to 0.

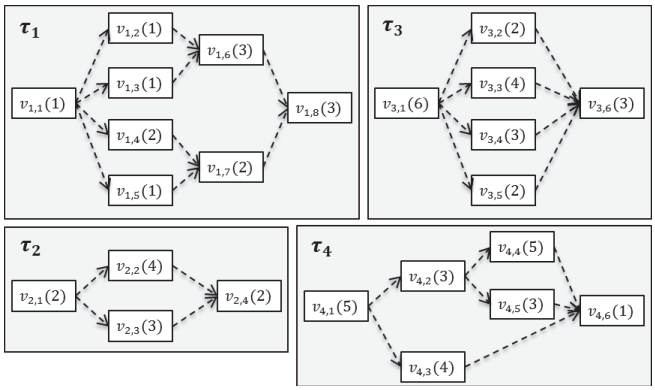


Fig. 1. DAGs of $lp(k)$ tasks; the $C_{i,j}$ of each node $v_{i,j}$ is presented in parentesis.

TABLE I. WORST-CASE WORKLOAD OF THE TASKS SHOWN IN FIGURE 1; EACH ROW CORRESPONDS TO TASKS EXECUTING ON $c = 1 \ldots 4$ CORES.

$\mu_1[c]$	$\mu_2[c]$	$\mu_3[c]$	$\mu_4[c]$
$C_{1,6}$or$C_{1,8} = 3$	$C_{2,2} = 4$	$C_{3,1} = 6$	$C_{4,1}$or$C_{4,4} = 5$
$C_{1,6}$+$C_{1,7} = 5$	$C_{2,2}$+$C_{2,3} = 7$	$C_{3,3}$+$C_{3,4} = 7$	$C_{4,4}$+$C_{4,3} = 9$
$C_{1,6}$+$C_{1,4}$+ $C_{1,5} = 6$	0	$C_{3,3}$+$C_{3,4}$+ $C_{3,2}$or$C_{3,5} = 9$	$C_{4,4}$+$C_{4,3}$+ $C_{4,5} = 12$
$C_{1,2}$+$C_{1,3}$+ $C_{1,4}$+$C_{1,5} = 5$	0	$C_{3,2}$+$C_{3,3}$+ $C_{3,4}$+$C_{3,5} = 11$	0

TABLE II. SET OF EXECUTION SCENARIOS $e^4 = \{s_1, s_2, s_3, s_4, s_5\}$.

$\mathbf{s_p} \in \mathbf{e^4}$	$\lvert\mathbf{s_p}\rvert$	**Execution scenario description**
$s_1 = \{1, 1, 1, 1\}$	4	Each task runs in 1 core
$s_2 = \{2, 2\}$	2	Each task runs in 2 cores
$s_3 = \{2, 1, 1\}$	3	1 task runs in 2 cores and 2 task in 1 cores each
$s_4 = \{3, 1\}$	2	1 task runs in 3 cores and 1 task in 1 core
$s_5 = \{4\}$	1	1 task runs in 4 cores

2) Overall worst-case workload: The lower-priority interference depends on how the execution of $lp(k)$ is distributed across the m cores. We define $e^m = \{s_1, s_2, ...s_{p(m)}\}$ as the set of different *execution scenarios* (and so interference scenarios) of $lp(k)$ running on m cores. $p(m)$ is equal to the *number of partitions*[1] of m, and can be computed with the pentagonal number theorem from the Euler's formulation: $p(m) = \sum_q (-1)^{q-1} p(m - q(3q-1)/2)$, where the sum is over all nonzero integers q (positive and negative) [22].

Table II shows the five possible execution scenarios assuming 4 cores (e^4, $p(4) = 5$). The number of tasks being executed in each execution scenario $s_l \in e^m$ is given by its cardinality, i.e., $|s_l|$. Each execution scenario $s_l \in e^m$ has an associated *overall worst-case workload*, computed as:

$$\rho_k[s_l] = \sum \max_{|s_l|}^{s_l} \{\mu_i\} \quad (7)$$

where $\sum \max_{|s_l|}^{s_l}$ is the sum of the $|s_l|$ largest combinations of μ_i that fits in the scenario s_l, and so maximizes the interference. Section V formulates Equation (7) as an ILP.

Table III shows the $\rho_k[s_l]$ of each execution scenario presented in Table II, considering the $\mu_i[c]$ from Table I. For instance, the overall

[1] In number theory and combinatory, a partition of a positive integer m is a way of writing m as a sum of positive integers. Two sums that differ only in the order of their summands are considered the same partition.

TABLE III. OVERALL WORST-CASE WORKLOAD OF TASKS WITHIN THE SET $lp(k)$ FOR EACH OF THE SCENARIOS GIVEN BY e^4.

$\mathbf{s_l}$	$\rho_\mathbf{k}[\mathbf{s_l}]$
s_1	$\mu_1[1]+\mu_2[1]+\mu_3[1]+\mu_4[1]=18$
s_2	$\mu_2[2]$ *or* $\mu_3[2]+\mu_4[2]=16$
s_3	$\mu_4[2]+\mu_2[1]+\mu_3[1]=19$
s_4	$\mu_4[3]+\mu_3[1]=18$
s_5	$\mu_3[4]=11$

worst-case workload of s_3, $\rho_k[s_3]=19$ results when τ_4 executes on 2 cores ($\mu_4[2]=9$), and τ_2 and τ_3 execute on 1 core each ($\mu_2[1]=4$ and $\mu_3[1]=6$).

3) Lower-priority interference: Finally, given the overall worst-case workload for each scenario $\rho_k[s_l]$, the lower-priority interference of $lp(k)$ presented in Equation (5), can be reformulated as the maximum overall worst-case workload among all scenarios:

$$\Delta_k^m = \max_{s_l \in e^m} \rho_k[s_l]$$
$$\Delta_k^{m-1} = \max_{s_l \in e^{m-1}} \rho_k[s_l] \quad (8)$$

where $\max_{s_l \in e^m}$ and $\max_{s_l \in e^{m-1}}$ provide the maximum worst-case workload among e^m and e^{m-1} scenarios.

The lower-priority interference of $lp(k)$, shown in Figure 1, is given by the maximum $\rho_k[s_l]$ from Table III, i.e., $\Delta_k^4 = 19$. On the contrary, the pessimistic approach given by Equation (5) selects the sum of the m largest NPRs among all lower-priority tasks, i.e. $\Delta_k^4 = C_{3,1}+C_{4,1}+C_{4,4}+C_{2,2} = 20$. The pessimism comes from the fact that nodes $v_{4,1}$ and $v_{4,4}$ cannot be executed in parallel. Similarly, $\Delta_k^3 = 15$ according to Equation (8), while $\Delta_k^3 = 16$ according to Equation (5).

Clearly, LP-ILP allows computing a tighter lower-priority interference, at the cost of increasing the complexity of deriving it, compared to the LP-max approach presented in Equation (5).

V. COMPUTATION OF RESPONSE-TIME FACTORS OF LP-ILP

Equation (4) shows that the schedulability of a DAG-based task-set under LP-ILP can be checked in pseudo-polynomial time if, beside deadline and period, we can derive: (1) the worst-case workload generated by each lower-priority task τ_i (i.e., μ_i, see Equation (6)), and (2) the overall worst-case workload of lower-priority tasks for each execution scenario $s_l \in e^m$ (i.e., $\rho_k[s_l]$, see Equation (7)). The former can be computed at compile-time for each task, and is independent from the task-set; the latter requires the complete task-set knowledge, and is computed at system integration time. In this section, we present the algorithms to compute these factors.

A. Worst-case workload of τ_i executing in c cores: $\mu_i[c]$

$\mu_i[c]$ is determined by the set of c NPRs of τ_i that can potentially execute in parallel. As a first step, we identify for each NPR the set of potential parallel NPRs; then, we compute the interference of parallel execution when different number of cores are used.

1) Computing the set of parallel NPRs: Given the DAG $G_i = (V_i, E_i)$, Algorithm 1 computes, for each NPR $v_{i,j} \in V_i$, the set of NPRs that can execute in parallel with it.

The algorithm takes as input the DAG of task τ_i, the topological order[2] of G_i, and, for each node $v_{i,j}$, the sets: (1) SIBLING($v_{i,j}$), which contains the nodes which have a common predecessor with $v_{i,j}$; (2) SUCC($v_{i,j}$), which contains the nodes reachable from $v_{i,j}$ and (3) PRED($v_{i,j}$), which contains the nodes from which $v_{i,j}$ can be reached. It outputs, for each $v_{i,j}$, the set $Par(v_{i,j})$, containing the nodes that can execute in parallel with it.

[2]A topological order is such that if there is an edge from u to v in the DAG, then u appears before v in the topological order. A topological order can be easily computed in time linear in the size of the DAG [23].

Algorithm 1 Parallel NPRs of τ_i

Input: *(1)* $G_i = (V_i, E_i)$*; (2)* TOPOLOGICAL-ORDER(G_i)*; (3)* SIBLING($v_{i,j}$), SUCC($v_{i,j}$), PRED($v_{i,j}$) $\forall v_{i,j} \in V_i$
Output: $Par(v_{i,j}), \forall v_{i,j} \in V_i$

```
1:  procedure PARALLEL-NPR
2:      for each v_{i,j} ∈ V_i do
3:          Par(v_{i,j}) ← ∅
4:          for each v_{i,l} ∈ SIBLING(v_{i,j}) do
5:              if (v_{i,j}, v_{i,l}) ∉ E_i and (v_{i,l}, v_{i,j}) ∉ E_i then
6:                  Succ ← SUCC(v_{i,l}) \ SUCC(v_{i,j})
7:                  Par(v_{i,j}) ← Par(v_{i,j}) ∪ {v_{i,l}} ∪ Succ
8:              end if
9:          end for
10:     end for
11:     for each v_{i,j} ∈ TOPOLOGICAL-ORDER(G_i) do
12:         for each v_{i,l} ∈ PRED(v_{i,j}) do
13:             Pred ← Par(v_{i,l}) \ PRED(v_{i,j})
14:             Par(v_{i,j}) ← Par(v_{i,j}) ∪ Pred
15:         end for
16:     end for
17: end procedure
```

The algorithm iterates twice over all nodes in V_i. The first loop (lines 2-10) adds to $Par(v_{i,j})$ (line 7) the set of sibling nodes $v_{i,l}$ that are not connected to $v_{i,j}$ by an edge (line 5), and the nodes reachable from $v_{i,l}$ (SUCC($v_{i,l}$)), discarding those connected to $v_{i,j}$ by an edge (line 6). The second loop (lines 11-15), which traverses V_i in topological order, adds to $Par(v_{i,j})$ (line 14) the set of nodes $Par(v_{i,l})$ computed at line 7, being $v_{i,l}$ a node from which $v_{i,j}$ can be reached ($v_{i,l} \in$ PRED($v_{i,j}$)). From $Par(v_{i,l})$ we discard the nodes from which $v_{i,j}$ can be reached (line 13).

As an example, consider node $v_{1,3}$ of τ_1 in Figure 1. The first loop iterates over the sibling nodes $v_{1,2}$, $v_{1,4}$ and $v_{1,5}$. None of them is connected to $v_{1,3}$ by an edge (lines 4 and 5); also, SUCC($v_{1,2}$) $= \{v_{1,6}, v_{1,8}\}$, SUCC($v_{1,4}$) $= \{v_{1,7}, v_{1,8}\}$ and SUCC($v_{1,5}$) $= \{v_{1,7}, v_{1,8}\}$. The algorithm discards from SUCC($v_{1,2}$) nodes $\{v_{1,6}, v_{1,8}\}$, since they are already included in SUCC($v_{1,3}$) (line 6). This is not the case of $v_{1,7} \in$ SUCC($v_{1,4}$) and SUCC($v_{1,5}$). Hence, we obtain $Par(v_{1,3}) = \{v_{1,2}, v_{1,4}, v_{1,5}, v_{1,7}\}$. The second loop does not add new nodes to $Par(v_{1,3})$ because the unique node from which $v_{1,3}$ can be reached is $v_{1,1}$, and $Par(v_{1,1}) = \emptyset$. When the second loop examines node $v_{1,7}$, the two sets $Par(v_{1,4})$ and $Par(v_{1,5})$ are considered, since $v_{1,4}, v_{1,5} \in$ PRED($v_{1,7}$). Then, nodes $v_{1,2}$, $v_{1,3}$ and $v_{1,6}$ are included in $Par(v_{1,7})$, since none of them belongs to PRED($v_{1,7}$).

2) Impact of parallel NPRs on c cores: For any task τ_i, we present an ILP formulation to compute $\mu_i[c]$, i.e., the sum of the c largest NPRs in V_i that, when executed in parallel, generate the worst-case workload.

Parameters: (1) c, i.e., the maximum number of cores used by τ_i; (2) $v_{i,j} \in V_i$; (3) $q_i + 1$, i.e., the number of NPRs; (4) $C_{i,j}$; and (5) $IsPar_{i,j,k} \in (0,1)$, i.e., a binary variable that takes 1 if $v_{i,j}$ and $v_{i,k}$ can execute in parallel, 0 otherwise.

Problem variables: (1) $b_j \in (0,1)$, i.e., a binary variable that takes the value 1 if $v_{i,j}$ is one of the selected parallel NPRs, 0 otherwise, and (2) $b_{j,k} = b_j \wedge b_k, b_{j,k} \in (0,1), j \neq k$, i.e., an auxiliary binary

variable.

Constraints: (1) $\sum_{j=1}^{q_i+1} b_j = c$, i.e., only c NPRs can be selected; (2) $\sum_{j=1}^{q_i+1} \sum_{k=j+1}^{q_k+1} b_{j,k} IsPar_{i,j,k} = c$, i.e., the selected NPRs can be executed in parallel; and (3) $b_{j,k} \geq b_j + b_k - 1; b_{j,k} \leq b_j; b_{j,k} \leq b_k$, i.e., auxiliary constraints used to model the logical *and*.

Objective function: $max \sum_{j=1}^{q_i+1} C_{i,j} b_j$

B. Overall worst-case workload of $lp(k)$ per execution scenario s_l: $\rho_k[s_l]$

Given the set $lp(k)$ and an execution scenario $s_l \in e^m$, we present an ILP formulation to derive $\rho_k[s_l]$, that is, the overall worst-case workload generated by $lp(k)$ under s_l.

Parameters: (1) $lp(k)$; (2) m; (3) s_l; and (4) $\mu_i[c], \forall \tau_i \in lp(k), \forall c = 1, \ldots, m$.

Problem variable: w_i^c, i.e., a binary variable that takes the value 1 on the selected $\mu_i[c]$ that contributes to the worst-case workload, 0 otherwise.

Constraints: (1) $\sum_{c=1}^{m} \sum_{\forall \tau_i \in lp(k)} w_i^c = |s_l|$, i.e., the number of tasks contributing to the worst-case workload must be equal to the size of the execution scenario; (2) $\forall \tau_i \in lp(k), \sum_{c=1}^{m} w_i^c \leq 1$, i.e., one task can be considered at most in one scenario; (3) $\sum_{\forall \tau_i \in lp(k)} w_i^c \geq 1, c \in s_l$, i.e., for each number of cores considered in s_l, there exist at least one $\mu_i[c]$ that is selected; and (4) $\sum_{c=1}^{m} \sum_{\forall \tau_i \in lp(k)} w_i^c \cdot c = m$, the number of cores considered is m.

Objective function: $max \sum_{c=1}^{m} \sum_{\forall \tau_i \in lp(k)} w_i^c \mu_i^c$

C. Complexity

The schedulability analysis upon which our approach is built, i.e. [10] and [15], have been proven to run in pseudo-polynomial time. This section discusses the complexity of the LP-ILP analysis presented in this paper.

Algorithm 1 (Section V-A1) requires to specify for each node in V_i the sets SIBLING, SUCC and PRED, which can be computed in quadratic time in the number of nodes. Similarly, the complexity of Algorithm 1 is quadratic in the size of the DAG task, i.e., $O(|V_k|^2)$. The ILP formulation to compute $\mu_i[c]$ (Section V-A2) is performed for each task (except for the highest-priority one), and the number of cores ranges from 2 to m (when $c = 1, \mu_i[1] = max_{1 \leq j \leq q_{i+1}} C_{i,j}$), hence the complexity cost is $O(nm) \cdot O(ilp_A)$. It is important to remark that Algorithm 1 (as well as its inputs) and the ILP that computes $\mu_i[c]$ are executed at compile-time for each task and are independent of the task-set and the system where they execute.

$\rho_k[s_l]$ (Section V-B) is computed for the execution scenarios e^m and e^{m-1}, and for each task τ_k (except for the lowest-priority task τ_n), hence the complexity cost is: $O(n \cdot p(m)) \cdot O(ilp_B) + O(n \cdot p(m-1)) \cdot O(ilp_B)$. The cost of solving both ILP formulations is pseudo-polynomial, if the number of constraints is fixed [24]. Our ILP formulations have fixed constraints, with a function cost of $O(ilp_A)$ and $O(ilp_B)$ depending on $|V_k|$ and $(m \cdot n)$ respectively.

Therefore, the cost of computing $\rho_k[s_l]$ for e^m dominates the cost of other operations; hence, the complexity of computing the lower-priority interference is pseudo-polynomial in the number of tasks and execution scenarios, i.e., cores (Section VI analyses the computational time required by our response time analysis).

VI. Experimental Results

This section evaluates our schedulability analysis for global fixed-priority scheduling with LP, computing the lower-priority interferences considering the two methods, i.e., LP-max and LP-ILP, presented in Section IV. The schedulability analysis, as well as the algorithm presented in Section V, have been implemented in MATLAB ®. The ILP formulation has been coded by using IBM ILOG OPL and solved by IBM ILOG/CPLEX [25].

A. Generation of tasks-sets and scheduling parameters

We randomly generate the DAG task-sets using the simulation environment presented in [10]. In order to properly exercise the two lower-priority interference methods, we generate two groups of DAG task-sets. The first group is composed of DAGs with different levels of parallelism, i.e., containing tasks spawning a high number of parallel NPRs, and task with very-limited parallelism (or even sequential). This group is very common in the embedded domain, representing systems incorporating data-flow tasks (typically with a high level of parallelism) and control-flow tasks (typically with a low level of parallelism). The second group is composed of DAGs with a high level of parallelism in which the number of parallel NPRs spawned is similar among tasks. This group is very common in high-performance domain, representing systems with only data-flow tasks.

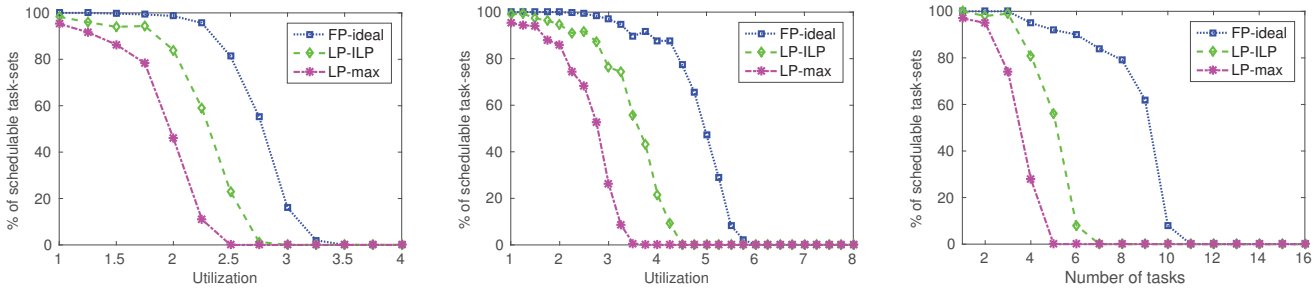
Highly parallel DAGs have been derived with the following parameters: the probabilities of creating a terminal node or keeping the expansion of the graph are $p_{term} = 0.4$ and $p_{par} = 0.6$, respectively; $n_{par} = 6$ is the maximum number of successors a node can have; and $\beta = 0.5$ is used to define the minimum DAG-task utilization. Moreover, the longest path of the DAGs is at most 7, and the WCET $C_{i,j}$ of each node is uniformly selected in the interval $[1, 100]$. We define an additional parameter that is the maximum number of nodes (NPRs) per DAG, which is set to 30. For each experiment, we generated 300 task-sets for each target utilization value (x-axis in Figure 2) , considering the implicit deadline case ($D_k = T_k$).

B. Evaluation

Our experiments aims to compare our proposed response-time analysis using the two lower-priority interference methods (labeled as *LP-max* and *LP-ILP*) against an ideal FP analysis (labeled as *FP-ideal*) in which the impact of lower priority interfere is discarded ($I_k^{lp} = 0$, see Equation (1)). It is important to remark that the performance of a real FP approach in which the preemption overheads would be included in the analysis may significantly decrease compared to LP. Accurately accounting for preemption overheads in FP is very difficult (if not impossible) since the execution can be preempted at any execution point of the task. Preemption overheads in the case of LP-max and LP-ILP have not been considered as well.

Figure 2 presents the percentage of schedulable DAG task-sets of the first group, on $m = 4, 8$ and 16 cores. In all cases, LP-ILP outperforms LP-max. The reason is because LP-max considers the sequential NPRs from tasks with lower parallelism. The LP-ILP instead, selects only NPRs that can actually execute in parallel.

Figure 2(a) shows the case in which $m = 4$ cores are used, ranging the utilization from 1 to 4. The three approaches are able to schedule nearly all the task-sets until the utilization reaches 2. From this point on, the performance of the LP-max approach drops earlier than the others, e.g., when the overall task-set utilization is 2.25, the scheduling task-sets rate is 11%, 59% and 95% for the LP-max, LP-ILP and FP-ideal, respectively. Figure 2(b) shows the schedulable task-set on $m = 8$ cores, ranging the utilization from 1 to 8. Assuming an utilization of 3.25, LP-max is able to schedule only the 8.67% of the task-sets while LP-ILP and FP-ideal achieve a scheduling rate



(a) m = 4 (b) m = 8 (c) m = 16

Fig. 2. Number of schedulable task sets running in (a) 4 cores, (b) 8 cores and (c) 16 cores as a function of the utilization.

of 74% and 94% respectively. LP-max cannot schedule any task-set when the utilization is higher than 3.5. In case of LP-ILP and FP-ideal, such a point is reached when utilization is higher than 4.5 and 5.75 respectively. Finally, Figure 2(c) shows the schedulable task-set on $m = 16$ cores, where the trend is maintained, although the distance between LP-ILP and the ideal FP case is slightly higher.

When considering the second group of DAG task-sets, the LP-max and the LP-ILP perform very similar on $m = 4, 8$ and 16 cores (results are not shown due to space constraints). The reason is that, when the parallelism increases, many more NPRs per task are allowed to be executed in parallel, so that the pessimism of LP-max with respect to LP-ILP is reduced.

Regarding the complexity of the LP-ILP approach, we compute the execution time of the response time analysis on an Intel(R) Core(TM) i7-3740QM processor. On average, the schedulability test takes 0.45, 4.75 seconds and 43 minutes when considering $m = 4$, $m = 8$ and $m = 16$, respectively, to provide a positive scheduling answer of a random generated DAG-based task-set.

VII. Conclusions

LP scheduling is an effective strategy to quantify and reduce the preemption-related overheads compared to FP. This paper proposes a new schedulability analysis of global fixed-priority scheduling with LP for task-sets composed of DAGs. Two methods have been proposed to compute the lower-priority interference: (1) a pessimistic but easy-to-compute method, named LP-max, which upper bounds the interference by selecting the NPRs with the longest worst-case execution time; and (2) a tighter but computationally-intensive method, named LP-ILP, which also takes into account precedence constraints among DAGs nodes in the analysis. Our results demonstrate that LP-ILP increases the accuracy of the schedulability test with respect to LP-max when considering DAG-based task-sets with different levels of parallelism. In the future, we intend to improve the LP analysis by (i) refining the estimation of the number of times a task can be preempted (and therefore blocked by lower priority tasks), and (ii) tighten the response-time analysis by taking into account the last non-preemptive region of each task.

References

[1] B. D. de Dinechin, D. van Amstel, M. Poulhies, and G. Lager, "Time-critical computing on a single-chip massively parallel processor," in *DATE*, 2014.

[2] E. Stotzer, A. Jayraj, M. Ali, A. Friedmann, G. Mitra, et al., "OpenMP on the Low-Power TI Keystone II ARM/DSP SoC," in *IWOMP*, 2013.

[3] P. Axer, S. Quinton, M. Neukirchner, R. Ernst, B. Döbel, and H. Härtig, "Response-time analysis of parallel fork-join workloads with real-time constraints," in *ECRTS*, July 2013.

[4] A. Saifullah, J. Li, K. Agrawal, C. Lu, and C. D. Gill, "Multi-core real-time scheduling for generalized parallel task models," *Real-Time Systems*, no. 4, 2013.

[5] S. Baruah, "Improved multiprocessor global schedulability analysis of sporadic DAG task systems," in *ECRTS*, July 2014.

[6] "OpenMP API, Version 4.0," OpenMP ARB, Tech. Rep., July 2013.

[7] P. Burgio, G. Tagliavini, A. Marongiu, and L. Benini, "Enabling fine-grained OpenMP tasking on tightly-coupled shared memory clusters," in *DATE*, 2013.

[8] R. Vargas, E. Quinones, and A. Marongiu, "OpenMP and timing predictability: a possible union?" in *DATE*, 2015.

[9] R. Vargas, S. Royuela, M. A. Serrano, X. Martorell, and E. Quinones, "A Lightweight OpenMP4 Run-time for Embedded Systems," in *asp-DAC*, 2016.

[10] A. Melani, M. Bertogna, V. Bonifaci, A. Marchetti-Spaccamela, and G. Buttazzo, "Response-time analysis of conditional DAG tasks in multiprocessor systems," in *ECRTS*, July 2015.

[11] J. Lee and K. G. Shin, "Improvement of real-time multi-coreschedulability with forced non-preemption," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 5, pp. 1233–1243, 2014.

[12] M. Bertogna, G. Buttazzo, M. Marinoni, G. Yao, F. Esposito, and M. Caccamo, "Preemption points placement for sporadic task sets," in *ECRTS*, July 2010.

[13] G. Buttazzo, M. Bertogna, and G. Yao, "Limited preemptive scheduling for real-time systems. a survey," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 1, pp. 3–15, March 2012.

[14] M. A. Serrano, A. Melani, R. Vargas, A. Marongiu, M. Bertogna, and E. Quinones, "Timing characterization of OpenMP4 tasking model," in *CASES*, 2015.

[15] A. Thekkilakattil, R. I. Davis, R. Dobrin, S. Punnekkat, and M. Bertogna, "Multiprocessor fixed priority scheduling with limited preemptions," in *RTNS*, November 2015.

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

[17] M. Bertogna, O. Xhani, M. Marinoni, F. Esposito, and G. Buttazzo, "Optimal selection of preemption points to minimize preemption overhead," in *ECRTS*, July 2011.

[18] B. Peng, N. Fisher, and M. Bertogna, "Explicit preemption placement for real-time conditional code," in *ECRTS*, July 2014.

[19] A. Block, H. Leontyev, B. Brandenburg, and J. Anderson, "A flexible real-time locking protocol for multiprocessors," in *RTCSA*, 2007.

[20] R. Davis, A. Burns, J. Marinho, V. Nelis, S. Petters, and M. Bertogna, "Global and partitioned multiprocessor fixed priority scheduling with deferred preemption," *ACM TECS*, no. 3, May 2015.

[21] J. Marinho, V. Nelis, S. Petters, M. Bertogna, and R. Davis, "Limited pre-emptive global fixed task priority," in *RTSS*, December 2013.

[22] P. Audibert, *Mathematics for informatics and computer science*. John Wiley & Sons, 2013.

[23] S. Dasgupta, C. H. Papadimitriou, and U. Vazirani, *Algorithms*. McGraw-Hill, Inc., 2006.

[24] C. H. Papadimitriou, "On the complexity of integer programming," *Journal of the ACM (JACM)*, vol. 28, 1981.

[25] IBM ILOG, "Cplex optimization studio," *URL: http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer*, 2014.