Schedulability Analysis of Conditional Parallel Task Graphs in Multicore Systems

Alessandra Melani, Marko Bertogna, Vincenzo Bonifaci,
Alberto Marchetti-Spaccamela, and Giorgio Buttazzo

Abstract—Several task models have been introduced in the literature to describe the intrinsic parallelism of real-time activities, including fork/join, synchronous parallel, DAG-based, etc. Although schedulability tests and resource augmentation bounds have been derived for these task models in the context of multicore systems, they are still too pessimistic to describe the execution flow of parallel tasks characterized by multiple (and nested) conditional statements, where it is hard to decide which execution path to select for modeling the worst-case scenario.

To overcome this problem, this paper proposes a task model that integrates control flow information by considering conditional parallel tasks (cp-tasks) represented by DAGs with both precedence and conditional edges. For this task model, a set of meaningful parameters are identified and computed by efficient algorithms and a response-time analysis is presented for different scheduling policies. A set of simulation results are finally reported to evaluate the efficiency of the proposed schedulability tests and their performance with respect to classic tests based on both conditional and non-conditional existing approaches.

Index Terms—Parallel scheduling, DAG Tasks, Response-Time Analysis, Multiprocessor Systems, Real-Time Systems.

✦

1 INTRODUCTION

The availability of multi-/many-core platforms in the embedded market [1], [2], [3], caused an increasing interest for applications with both high-performance and real-time requirements. Following the same trend, several classic schedulability results have been extended to comply with such new platforms, providing new models and tests to guarantee the timing requirements of parallel task systems. Examples of the task models proposed to capture the parallel structure of an application include the fork/join model [17], the synchronous parallel task model [25], and the DAG-based task model [9]. Each of these models divides a task into smaller computational units, called sub-tasks, which can run simultaneously on different cores.

As noted by Fonseca et al. [16], the problem introduced by conditional statements does not arise in uniprocessor systems, where a single parameter, the worst-case execution time (WCET), is sufficient for deriving an exact schedulability analysis of the task-set. On the other hand, conditional statements become much more detrimental in the schedulability analysis of parallel tasks running on a multicore system. The problem is exacerbated in the presence of multiple conditional statements characterized by branches with very different length. For example, consider a task that conditionally processes an image depending on a particular system state: it may either fork a large number of sub-tasks to process small pixels clusters or run a single sub-task to process a larger set of pixels in a sequential way. Moreover, the level of detail at which each image region is processed may depend on runtime data. As a result, the response-time of such a task, as well as its interference on the other tasks, may vary significantly from instance to instance. Therefore, identifying the worst-case scenario that mostly affects the system schedulability is a challenging issue.

An example of parallel task T_0 specified according to the OpenMP standard is illustrated in Figure 1. The task has a conditional statement at the beginning of its execution. Depending on the conditional clause, the task can take the upper branch, creating a sequential sub-task T_1 of 10 time units, or the lower branch, forking three sub-tasks T_2, T_3, T_4 of 6 time units each. Note that the branch leading to the worst-case response-time depends on the number of cores and the interference from the other tasks. For example, with three or more cores and no interfering tasks, the largest response-time is given by the upper branch, for every work-conserving scheduler[1] (i.e., 10 time units instead of 6). With less cores, the largest response-time is given by the lower branch (i.e., 12 time units with two cores, and 18 time units with one core). If interfering tasks are present, the situation is even more challenging, because the conclusions derived above may be reversed! For example, adding a sequential task of 6 time-units, the worst-case response-time with three cores is now given by the lower branch (12 units instead of 10).

Similarly, it is not easy to predict which branch imposes a larger interference on the other tasks: depending on the characteristics of the other tasks, a higher interference may be produced by a set of parallel sub-tasks or by a longer sequential sub-task. Since applications typically consists

• A. Melani and G. Buttazzo are with the TeCIP Institute, Scuola Superiore Sant'Anna, Pisa, Italy. E-mail: {alessandra.melani, g.buttazzo}@sssup.it.
• M. Bertogna is with the University of Modena and Reggio-Emilia, Modena, Italy. E-mail: m.bertogna@unimore.it.
• V. Bonifaci is with Istituto di Analisi dei Sistemi ed Informatica, CNR, Rome, Italy. E-mail: vincenzo.bonifaci@iasi.cnr.it.
• A. Marchetti-Spaccamela is with Sapienza University of Rome, Rome, Italy. E-mail: alberto@dis.uniroma1.it.

1. A scheduler is work-conserving if it never idles a core whenever there is pending workload to execute.

```
#pragma omp parallel num_threads(N) {
 #pragma omp master {
 #pragma omp task { // T_0
   if (condition) {
     #pragma omp task { // T_1 }
   }
   else {
     #pragma omp task { // T_2 }
     #pragma omp task { // T_3 }
     #pragma omp task { // T_4 }
 }}}}
```
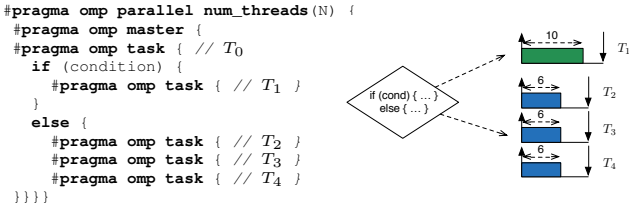
Figure 1: A parallel program with conditional execution.

of several nested conditional statements, the problem of mapping parallel applications to a task model that does not explicitly consider conditional statements is very difficult to solve.

1.1 Contributions and paper organization

This paper extends the parallel DAG model by integrating conditional constructs to provide a tighter analysis to parallel task systems. In the proposed *conditional parallel* task (cp-task) model, each task is represented by a DAG containing both parallel and conditional nodes. To capture the structure of parallel applications, a formal definition of cp-task is provided by specifying the possible connections between the various (conditional and non-conditional) sections of the graph.

For the cp-task model, efficient ways to compute an upper-bound on the response-time of each cp-task are derived using different global scheduling algorithms. The effectiveness of the proposed schedulability analysis is shown by extensive experiments. The paper also shows that the proposed response-time analysis can be efficiently applied to non-conditional task models as well (as the DAG task model [9]). For these latter systems, simulations results show that a significantly higher number of schedulable task-sets is detected at a considerably smaller time complexity, with respect to existing approaches.

The remainder of this paper is organized as follows. Section 2 reviews the related work. Sections 3 introduces the proposed task model and the notation used throughout the paper. Section 4 characterizes the critical interference among tasks, while in Section 5 we present our response-time analysis and describe how to compute certain task parameters of interest. Section 6 reports the simulation results, and Section 7 presents a realistic case study. Finally, Section 8 states the conclusions.

2 RELATED WORK

Several parallel task models have been proposed in the real-time literature, although most of them are limited to non-conditional execution modes. One of the first proposals, the *fork-join* task model, was introduced by Lakshmanan, Kato and Rajkumar [6], [17]. In this model, a task is represented as an alternating sequence of sequential and parallel segments, and every parallel segment has the same degree of parallelism (which is constrained to be less than or equal to the number of available processors). A natural extension of this approach is the *synchronous parallel* model [5], [14], [21], [22], [25], allowing consecutive parallel segments

and an arbitrary degree of parallelism of every segment. Synchronization is still enforced at the boundary of each segment, in the sense that a sub-task in the new segment may start only after all the sub-tasks in the previous segment have completed. A more flexible model of concurrency is the *DAG* model, where a task is represented by a directed acyclic graph (DAG) in which nodes are sequential sub-tasks and arcs represent precedence constraints between sub-tasks [7], [9], [12], [18], [19], [20], [24], [26].

To the best of our knowledge, the first attempt for enriching a parallel task model with control-flow information was proposed by Fonseca et al. [16] through the *multi-DAG* model. This model represents a parallel task as a collection of DAGs, each representing a different execution flow. The authors proposed a method to combine such flows into a single synchronous parallel task that preserves the execution requirements and the precedence constraints of all the execution flows that can possibly occur at runtime, thus reducing the schedulability problem to a simpler problem for synchronous parallel tasks. A disadvantage of this approach is that it is not scalable with respect to the number of sub-tasks, since the number of different flows through a DAG can be exponential in the number of nodes. Moreover, it adds pessimism in the task transformation process and requires server-based synchronization mechanisms that may be difficult to implement.

The same conditional parallel task model proposed in this paper was considered by Baruah et al. [8]. However, the authors focused on global Earliest Deadline First (G-EDF) scheduling, proposing an efficient algorithm that transforms any conditional DAG task into a non-conditional DAG task that is "equivalent", in the sense that it preserves the quantities used by the test of [12] for DAGs without conditional statements. This allows extending the existing tests and speedup bounds for DAG tasks to the G-EDF schedulability of systems of conditional DAG tasks.

3 SYSTEM MODEL AND DEFINITIONS

This paper considers a set $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$ of n sporadic conditional parallel tasks (*cp-tasks*) that execute upon a platform consisting of m identical processors. Each cp-task τ_k releases a potentially infinite sequence of jobs. Each job of τ_k is separated from the next by at least T_k time-units and has a constrained relative deadline $D_k \leq T_k$. Moreover, each cp-task τ_k is represented as a directed acyclic graph $G_k = (V_k, E_k)$, where $V_k = \{v_{k,1}, \ldots, v_{k,n_k}\}$ is a set of nodes (or vertices) and $E_k \subseteq V_k \times V_k$ is a set of directed arcs (or edges), as shown in Figure 2. Each node $v_{k,j} \in V_k$ represents a sequential chunk of execution (or "sub-task") and is characterized by a worst-case execution time $C_{k,j}$. Preemption/migration overheads and cross-core interference (caused by contention for shared hardware components, such as shared caches, memory buses, etc.) are considered to be negligible. In addition, we assume that cp-tasks do not engage in any synchronization. Arcs represent dependencies between sub-tasks, that is, if $(v_{k,1}, v_{k,2}) \in E_k$, then $v_{k,1}$ must complete before $v_{k,2}$ can start executing. A node with no incoming arcs is referred to as a *source*, while a node with no outgoing arcs is referred to as a *sink*. Without loss of generality, each cp-task is assumed to have exactly

one source v_k^{source} and one sink node v_k^{sink}. If this is not the case, a dummy source/sink node with zero WCET can be added to the DAG, with arcs to/from all the source/sink nodes. The subscript k in the parameters associated to the task τ_k is omitted whenever the reference to the task is clear in the discussion.

In the cp-task model, nodes can be of two types: a) *regular* nodes, represented as rectangles, allow *all* successor nodes to be executed in parallel; b) *conditional* nodes, coming in pairs and denoted by diamonds and circles, represent the beginning and the end of a conditional construct, respectively, and require the execution of *exactly one* node among the successors of the start node.

To properly model the possible execution flows, a further restriction is imposed to the connections within a conditional branch. That is, a node belonging to a branch of a conditional statement cannot be connected to nodes outside that branch (including other branches of the same statement). This is formally stated in the following definition.

Definition 3.1. *Let (v_1, v_2) be a pair of conditional nodes in a DAG $G_k = (V_k, E_k)$. The pair (v_1, v_2) is a conditional pair if the following hold:*

1) *If there are exactly q outgoing arcs from v_1 to nodes s_1, s_2, ..., s_q, for some $q > 1$, then there are exactly q incoming arcs into v_2 in E_k, from some nodes $t_1, t_2, \ldots, t_q$.*
2) *For each $\ell \in \{1, 2, \ldots, q\}$, let $V'_\ell \subseteq V_k$ and $E'_\ell \subseteq E_k$ denote all the nodes and arcs on paths reachable from s_ℓ that do not include node v_2. By definition, s_ℓ is the sole source node of the DAG $G'_\ell := (V'_\ell, E'_\ell)$. It must hold that t_ℓ is the sole sink node of G'_ℓ.*
3) *It must hold that $V'_\ell \cap V'_j = \emptyset$ for all ℓ, j, $\ell \neq j$. Additionally, with the exception of (v_1, s_ℓ) there should be no arcs in E_k into nodes in V'_ℓ from nodes not in V'_ℓ, for each $\ell \in \{1, 2, \ldots, q\}$. That is, $E_k \cap ((V_k \setminus V'_\ell) \times V'_\ell) = \{(v_1, s_\ell)\}$ should hold for all ℓ.*

A *chain* or *path* of a cp-task τ_k is a sequence of nodes $\lambda = (v_{k,a}, \ldots, v_{k,b})$ such that $(v_{k,j}, v_{k,j+1}) \in E_k, \forall j \in [a, b)$. The *length* of a chain of τ_k, denoted by $\text{len}(\lambda)$, is the sum of the WCETs of all its nodes, that is, $\sum_{j=a}^{b} C_{k,j}$. The *longest path* of a cp-task is any source-sink path of the task that achieves the longest length.

Definition 3.2. *The* length *of a cp-task τ_k, denoted by L_k, is the length of any longest path of τ_k.*

Note that L_k also represents the minimum worst-case execution time of cp-task τ_k, that is, the time required to execute it when the number of processing units is sufficiently large (potentially infinite) to allow the task to always execute with maximum parallelism. A necessary condition for the feasibility of a cp-task τ_k is that $L_k \leq D_k$.

In the absence of conditional branches, the classical sporadic DAG task model defines the *volume* of the task as the worst-case execution time needed to complete it on a dedicated single-core platform [9], [12], [18], [26]. This quantity can be computed as the sum of the WCETs of all the sub-tasks, that is $\sum_{v_{k,j} \in V_k} C_{k,j}$. In the presence of conditional branches, assuming that all sub-tasks are always executed is overly pessimistic. Hence, the concept of volume
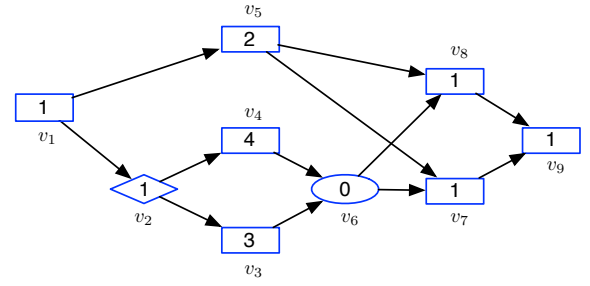

Figure 2: A sample cp-task. Each vertex is labeled with the WCET of the corresponding sub-task.

of a cp-task is generalized by introducing the notion of *worst-case workload*.

Definition 3.3. *The* worst-case workload W_k *of a cp-task τ_k is the maximum time needed to execute an instance of τ_k on a dedicated single-core platform, where the maximum is taken among all possible choices of conditional branches.*

Section 5.3 explains in detail how the worst-case workload of a task can be computed efficiently.

The *utilization* U_k of a cp-task τ_k is the ratio between its worst-case workload and its period, that is, $U_k = W_k/T_k$. For the task-set $\mathcal{T}$, its *total utilization* is defined as $U_{\mathcal{T}} = \sum_{i=1}^{n} U_i$. A simple necessary condition for feasibility is $U_{\mathcal{T}} \leq m$.

Example. Figure 2 illustrates a sample cp-task consisting of nine sub-tasks (nodes) $V = \{v_1, \ldots, v_9\}$ and twelve precedence constraints (arcs). The number inside each node represents its WCET. Two of the nodes, v_2 and v_6, form a conditional pair, meaning that only one sub-task between v_3 and v_4 will be executed (but never both), depending on a conditional clause. The length (longest path) of this cp-task is $L = 8$, and is given by the chain $(v_1, v_2, v_4, v_6, v_7, v_9)$. Its volume is 14 units, while its worst-case workload must take into account that either v_3 or v_4 are executed at every task instance. Since v_4 corresponds to the branch with the largest workload, $W = 11$.

To further clarify the restrictions imposed to the graph structure, note that v_4 cannot be connected to v_5, because this would violate the correctness of conditional constructs and the semantics of the precedence relation. In fact, if they were connected and v_3 were executed, then v_5 would wait forever, since v_4 is not executed. For the same reason, no connection is possible between v_4 and v_3, as they belong to different branches of the same conditional statement.

The notation used throughout the paper is summarized in Table 1.

4 CRITICAL INTERFERENCE OF CP-TASKS

This section presents a schedulability analysis for cp-tasks globally scheduled by any work-conserving scheduler. The analysis is based on the notion of *interference*. In the existing literature for globally scheduled sequential task systems, the interference on a task τ_k is defined as the sum of all intervals in which τ_k is ready, but it cannot execute because all cores are busy executing other tasks. We modify this definition to

Table 1: Notation

$\mathcal{T}$	set of cp-tasks	n	number of tasks in $\mathcal{T}$
τ_k	k-th task of $\mathcal{T}$	D_k	relative deadline of τ_k
T_k	period of τ_k	G_k	DAG associated to τ_k
V_k	node set of G_k	E_k	arc set of G_k
$v_{k,j}$	jth sub-task of τ_k	$C_{k,j}$	WCET of $v_{k,j}$
L_k	length of τ_k's longest chain	W_k	worst-case workload of τ_k
U_k	utilization of τ_k		

adapt it to the parallel nature of cp-tasks, by introducing the concept of critical interference [14], [21].

Fix a set of cp-tasks $\mathcal{T}$ and a work-conserving scheduler. The first useful notion is that of a *critical chain* of a task.

Definition 4.1. *The critical chain λ_k^* of a cp-task τ_k is the chain of nodes of τ_k that leads to its worst-case response-time R_k.*

The critical chain of cp-task τ_k is in principle determined by taking the sink vertex v_k^{sink} of the worst-case instance of τ_k (i.e., the job of τ_k that has largest response-time in the worst-case scenario), and recursively pre-pending the last to complete among the predecessor nodes (whether conditional or not), until the source vertex $v_{k,1}$ has been included in the chain.

A *critical node* of task τ_k is a node that belongs to τ_k's critical chain. Since the response-time of a cp-task is given by the response-time of the sink vertex of the task, the sink node is always a critical node. For deriving the worst-case response-time of a task, it is then sufficient to characterize the maximum interference suffered by its critical chain.

Definition 4.2. *The critical interference I_k on task τ_k is defined as the cumulative time during which some critical nodes of the worst-case instance of τ_k are ready, but do not execute because all cores are busy.*

Lemma 4.1. *Given a set of cp-tasks $\mathcal{T}$ scheduled by any work-conserving algorithm on m identical processors, the worst-case response-time of each task τ_k is*

$$R_k = \text{len}(\lambda_k^*) + I_k. \tag{1}$$

Proof. Let r_k be the release time of the worst-case instance of τ_k. In the scheduling window $[r_k, r_k + R_k]$, the critical chain will require $\text{len}(\lambda_k^*)$ time-units to complete. By Definition 4.2, at any time in this window in which τ_k does not suffer critical interference, some node of the critical chain is executing. Therefore $R_k - I_k = \text{len}(\lambda_k^*)$. □

The difficulty in using Equation (1) for schedulability analysis is that the term I_k may not be easy to compute. An established solution is to express the total interfering workload as a function of individual contributions of the interfering tasks, and then upper-bound such contributions with the worst-case workload of each interfering task τ_i. In the following, we explain how such interfering contributions can be computed, and how they relate to each other to determine the total interfering workload.

Definition 4.3. *The critical interference $I_{i,k}$ imposed by task τ_i on task τ_k is defined as the cumulative workload executed by*
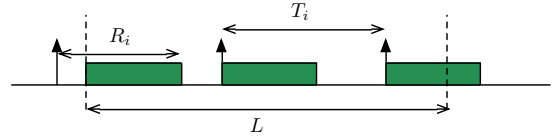

Figure 3: Worst-case scenario to maximize the workload of an interfering task τ_i in the sequential case.

sub-tasks of τ_i while a critical node of the worst-case instance of τ_k is ready to execute but is not executing.

Lemma 4.2. *For any work-conserving algorithm, the following relation holds:*

$$I_k = \frac{1}{m} \sum_{\tau_i \in \mathcal{T}} I_{i,k}. \tag{2}$$

Proof. By the work-conserving property of the scheduling algorithm, whenever a critical node of τ_k is interfered, all m cores are busy executing other sub-tasks. The total amount of workload executed by sub-tasks interfering with the critical chain of τ_k is then mI_k. Hence, $\sum_{\tau_i \in \mathcal{T}} I_{i,k} = mI_k$, and by reordering the terms, the lemma follows. □

Note that when $i = k$, the critical interference $I_{k,k}$ may include the interfering contributions of non-critical sub-tasks of τ_k on itself, that is, the *self-interference* of τ_k.

By combining Equations (1) and (2), the response-time of a task τ_k can be rewritten as

$$R_k = \text{len}(\lambda_k^*) + \frac{1}{m}I_{k,k} + \frac{1}{m} \sum_{\tau_i \in \mathcal{T}, i \neq k} I_{i,k}. \tag{3}$$

5 RESPONSE-TIME ANALYSIS

In this section we derive an upper-bound on the worst-case response-time of each cp-task using Equation (3). To this aim we need to compute the interfering contributions $I_{i,k}$. In the sequel, we first consider the inter-task interference ($i \neq k$) and then the intra-task interference ($i = k$).

5.1 Inter-task interference

We follow the approach adopted in [10], [21], that divides the contribution to the workload of an interfering task τ_i in a window of interest between carry-in, body, and carry-out jobs. The *carry-in* job is the first instance of τ_i that is part of the window of interest and has release time before and deadline within the window of interest. The *carry-out* job is the last instance of τ_i executing in the window of interest, having a deadline after the window of interest. All other instances of τ_i are named *body jobs*.

For sequential task-sets, an upper-bound on the workload of an interfering task τ_i within a window of length L occurs when the first job of τ_i starts executing as late as possible (with a starting time aligned with the beginning of the window of interest) and later jobs are executed as soon as possible [10] (see Figure 3).

For cp-task systems, it is more difficult to determine a configuration that maximizes the carry-in and carry-out contributions. In fact:
1) Due to the precedence constraints and different degree of parallelism of the various execution paths of a cp-task, it
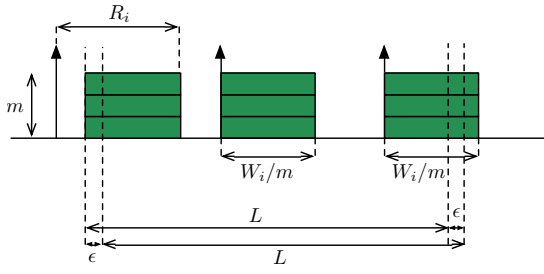
Figure 4: Worst-case scenario to maximize the workload of an interfering cp-task τ_i.

may happen that a larger workload is executed within the window if the interfering task is shifted left, i.e., by decreasing the carry-in and increasing the carry-out contributions. This happens for example when the first part of the carry-in job has little parallelism, while the carry-out part at the end of the window contains multiple parallel sub-tasks.

2) A sustainable schedulability analysis [13] must guarantee that all tasks meet their deadlines even when some of them execute less than the worst-case. For example, one of the sub-tasks of an execution path of a cp-task may execute for less than its WCET $C_{i,j}$. This may lead to larger interfering contributions within the window of interest (e.g. a parallel section of a carry-out job is included in the window due to an earlier completion of a preceding sequential section).

3) The carry-in and carry-out contribution of a cp-task may correspond to different conditional paths of the same task, with different levels of parallelism.

To circumvent the above issues, we consider a scenario in which each interfering job of task τ_i executes for its worst-case workload W_i, i.e., the maximum amount of workload that can be generated by a single instance of a cp-task. We defer the computation of W_i to Section 5.3. The next lemma provides a safe upper-bound on the workload of a task τ_i within a window of interest of length L.

Lemma 5.1. *An upper-bound on the workload of an interfering task τ_i in a window of length L is given by*

$$\mathcal{W}_i(L) = \left\lfloor \frac{L + R_i - W_i/m}{T_i} \right\rfloor W_i +$$
$$+ \min \left(W_i, m \cdot ((L + R_i - W_i/m) \mod T_i) \right).$$

Proof. Consider a situation in which all instances of τ_i execute for their worst-case workload W_i. The highest workload within a window of length L for such a task configuration is produced when the carry-in and carry-out contributions are evenly distributed among all cores, as shown in Figure 4. Note that distributing the carry-in or carry-out contributions on a lesser number of cores may not possibly increase the workload within the window. Moreover, other task configurations with a smaller workload for the carry-in or carry-out instance cannot lead to a higher workload in the window of interest: although a reduced carry-in workload may allow including a larger part of the carry-out (as in shifting right the window of interest by $\epsilon = \Delta W_i/m$ in the figure), the carry-out part that enters the window from the right cannot be larger than the carry-in reduction.

An upper-bound on the number of carry-in and body instances that may execute within the window is

$$\left\lfloor \frac{L + R_i - W_i/m}{T_i} \right\rfloor,$$

each one contributing for W_i. The portion of the carry-out job included in the window of interest is $(L + R_i - W_i/m) \mod T_i$. Since at most m cores may be occupied by the carry-out job within that interval, and the carry-out job cannot execute for more than W_i units, the lemma follows. $\square$

If we schedule according to global EDF, then another upper-bound on the interfering contribution of each task is obtained by noting that the deadline of the interfering jobs cannot be later than that of the interfered task.

Lemma 5.2. *An upper-bound on the interfering workload of a task τ_i on a task τ_k with global EDF is given by*

$$\mathcal{I}_{i,k} = \left(\left\lfloor \frac{D_k - D_i}{T_i} \right\rfloor + 1 \right) W_i +$$
$$+ \min \left(W_i, m \cdot \max \left(0, D_k \mod T_i - D_i + R_i \right) \right).$$

Proof. Consider a window $[r_k, r_k + D_k]$ of a task τ_k. The interfering contribution of a task τ_i is maximized when the deadline of the carry-out job is aligned with $r_k + D_k$. In such a situation, the number of body and carry-out jobs of τ_i is $\lfloor (D_k - D_i)/T_i \rfloor + 1$, each contributing for W_i. Concerning the carry-in job, its scheduling window overlaps with the considered window for $D_k \mod T_i$. Subtracting the slack $(D_i - R_i)$, we obtain that the carry-in job may execute for at most $(D_k \mod T_i) - (D_i - R_i)$ within the considered window, when this term is not negative. Since the carry-in job can occupy at most m cores, and its worst-case workload is bounded by W_i, the lemma follows. $\square$

5.2 Intra-task interference

We now consider the remaining terms of of Equation (3), which take into account the contribution of the considered task to its overall response-time, and we compute an upper-bound on $Z_k \stackrel{\text{def}}{=} \text{len}(\lambda_k^*) + \frac{1}{m} I_{k,k}$.

Lemma 5.3. *For a constrained deadline cp-task system scheduled with any work-conserving algorithm, the following relation holds for any task τ_k:*

$$Z_k = \text{len}(\lambda_k^*) + \frac{1}{m} I_{k,k} \leq L_k + \frac{1}{m}(W_k - L_k). \quad (4)$$

Proof. Since we are in a constrained deadline setting, a job will never be interfered by other jobs of the same task. Being W_k the maximum possible workload produced by a job of cp-task τ_k, the portion that may interfere with the critical chain λ_k^* is $W_k - \text{len}(\lambda_k^*)$. Then, $I_{k,k} \leq W_k - \text{len}(\lambda_k^*)$. Hence,

$$\text{len}(\lambda_k^*) + \frac{1}{m} I_{k,k} \leq \text{len}(\lambda_k^*) + \frac{1}{m}(W_k - \text{len}(\lambda_k^*)). \quad (5)$$

Since $\text{len}(\lambda_k^*) \leq L_k$ and $m \geq 1$, the lemma follows. $\square$

Since Z_k includes only the contribution of task τ_k, one may think that the sum $(\text{len}(\lambda_k^*) + \frac{1}{m} I_{k,k})$ is equal to the worst-case response-time of τ_k when it is executed in isolation on the multi-core system (i.e., the makespan of τ_k).

However, this is not true. For example consider the case of a cp-task τ_k with only one if-then-else statement; assume that when the "if" part is executed, the task executes one sub-task of length 10, otherwise, the task executes two parallel sub-tasks of length 6 each. When τ_k is executed in isolation on a two-core platform, the makespan is clearly given by the "if" branch, i.e., 10. When instead τ_k can be interfered by one job of a task τ_i which executes a single sub-task of length 6, the worst-case response time of τ_k occurs when the "else" branch is executed, yielding a response time of 12. The share of the response time due to the term $\text{len}(\lambda_k^*) + \frac{1}{m}I_{k,k}$ in Equation (3) is $6 + (1/2) \cdot 6 = 9$, which is strictly smaller than the makespan. Note that $\text{len}(\lambda_k^*) + \frac{1}{m}I_{k,k}$ does not even represent a valid lower bound on the makespan. This can be seen by replacing the "if" branch in the above example with a shorter subtask of length 8, giving a makespan of 8. For this reason, one cannot replace the term $\text{len}(\lambda_k^*) + \frac{1}{m}I_{k,k}$ in Equation (4) with the makespan of τ_k.

The righthand side of Equation (4) $(L_k + \frac{1}{m}(W_k - L_k))$ has been therefore introduced to upper-bound the term $\text{len}(\lambda_k^*) + \frac{1}{m}I_{k,k}$. Interestingly, this quantity does also represent a valid upper-bound on the makespan of τ_k, so that it can be used to bound the response time of a cp-task executing in isolation. We omit the proof that is identical to the proofs of the given bounds, considering only the interference due to the task itself.

5.3 Computation of cp-task parameters

The upper-bounds on the interference given by Lemmata 5.1, 5.2 and 5.3 require the computation of two characteristic parameters for each cp-task τ_k: the worst-case workload W_k and the length of the longest chain L_k. The longest path of a cp-task can be computed exactly in the same way as for the longest path of a classical DAG task, since any conditional branch defines a set of possible paths in the graph. For this purpose, conditional nodes can be considered as if they were simply regular nodes. The computation can be implemented in time linear in the size of the DAG by standard techniques, see e.g. Bonifaci et al. [12] and references therein.

The computation of the worst-case workload of a cp-task is more involved. We hereafter show an algorithm to compute W_k for each task τ_k in time quadratic in the DAG size, whose pseudocode is shown in Algorithm 1. The algorithm first computes a topological order of the DAG[2]. Then, exploiting the (reverse) topological order, a simple dynamic program can compute for each node the accumulated workload corresponding to the portion of the graph already examined. The algorithm must distinguish the case when the node under analysis is the head of a conditional pair or not. If this is the case, then the maximum accumulated workload among the successors is selected, otherwise the sum of the workload contributions of all successors is computed.

Algorithm 1 takes as input the graph representation of a cp-task G and outputs its worst-case workload W. In the

---

Algorithm 1 Worst-Case Workload Computation

1: **procedure** WCW(G)
2:     $\sigma \leftarrow \text{TOPOLOGICALORDER}(G)$
3:     $S(v^{\text{sink}}) \leftarrow \{v^{\text{sink}}\}$
4:     **for** $v_i \in \sigma$ from sink to source **do**
5:        **if** $\text{SUCC}(v_i) \neq \emptyset$ **then**
6:           **if** $\text{ISBEGINCOND}(v_i)$ **then**
7:              $v^* \leftarrow \text{argmax}_{v \in \text{SUCC}(v_i)} C(S(v))$
8:              $S(v_i) \leftarrow \{v_i\} \cup S(v^*)$
9:           **else**
10:              $S(v_i) \leftarrow \{v_i\} \cup \bigcup_{v \in \text{SUCC}(v_i)} S(v)$
11:           **end if**
12:        **end if**
13:     **end for**
14:     **return** $C(S(v^{\text{source}}))$
15: **end procedure**

---

algorithm, for any set of nodes S, its total WCET is denoted by $C(S)$. First, at line 2, a topological sorting of the vertices is computed and stored in the permutation σ. Then, the permutation σ is scanned in reverse order, that is, from the (unique) sink to the (unique) source of the DAG. At each iteration of the for loop at line 4, a node v_i is analyzed; a set variable $S(v_i)$ is used to store the set of nodes achieving the worst-case workload of the subgraph including v_i and all its descendants in the DAG. Since the sink node has no successors, $S(v^{\text{sink}})$ is initialized to $\{v^{\text{sink}}\}$ at line 3. Then, the function $\text{SUCC}(v_i)$ computes the set of successors of v_i. If that set is not empty, function $\text{ISBEGINCOND}(v_i)$ is invoked to determine whether v_i is the head node of a conditional pair. If so, the node v^* achieving the largest value of $C(S(v))$, among $v \in \text{SUCC}(v_i)$, is computed (line 7). The set $S(v^*)$ therefore achieves the maximum cumulative worst-case workload among the successors of v_i, and is then used to create $S(v_i)$ together with v_i. Instead, whenever v_i is not the head of a conditional pair, all its successors are executed at runtime. Therefore, the workload contributions of all its successors must be merged into $S(v_i)$ (line 10) together with v_i. The procedure returns the worst-case workload accumulated by the source vertex, that is $C(S(v^{\text{source}}))$.

The complexity of the algorithm is quadratic in the size of the input DAG. Indeed, there are $O(|E|)$ set operations performed throughout the algorithm, and some operations on a set S (namely, the ones at line 7) also require computing $C(S)$, which has cost $O(|V|)$. So the time complexity is $O(|V||E|)$. To implement the set operations, set membership arrays are sufficient.

One may be tempted to simplify the procedure by avoiding the use of set operations, keeping track only of the cumulative worst-case workload at each node, and allowing a linear complexity in the DAG size. However, such an approach would lead to an overly pessimistic result. Consider a simple graph with a source node forking multiple parallel branches which then converge on a common sink. The cumulative worst-case workload of each parallel path includes the contribution of the sink. If we simply sum such contributions to derive the cumulative worst-case workload of the source, the contribution of the sink would be counted multiple times. Set operations are therefore needed to avoid

---

2. A topological order is such that if there is an arc 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 (see any basic algorithm textbook, such as [15]).

accounting multiple times each node's contribution.

We now present refinements of Algorithm 1 in special sub-cases of interest.

5.3.1 Non-conditional DAG tasks

The basic sporadic DAG task model does not explicitly account for conditional branches. Therefore, all vertices of a cp-task contribute to the worst-case workload, which is then equal to the volume of the DAG task: $W_k = \sum_{v_{k,j} \in V_k} C_{k,j}$. In this particular case, the time complexity to derive the worst-case workload of a task (quadratic in the general case), becomes $O(|V|)$, i.e., linear in the number of vertices.

5.3.2 Series-parallel conditional DAG tasks

Some programming languages yield *series-parallel* cp-tasks, that is, cp-tasks that can be obtained from a single edge by series composition and/or parallel composition. For example, the cp-task in Figure 5 is series-parallel, while the cp-tasks in Figures 2 and 6 are not. Such a structure can be detected in linear time [28]. In series-parallel graphs, for every head s_i of a conditional *or parallel* branch there is a corresponding tail t_i. For example, in Figure 5, the tail corresponding to parallel branch head v_2 is v_9. Algorithm 1 can be specialized to series-parallel graphs. For each vertex u, the algorithm will simply keep track of the worst-case workload of the subgraph reachable from u, as follows. For each head vertex s_i of a parallel branch, the contribution from all successors should be added to s_i's WCET, subtracting however the worst-case workload of the corresponding tail t_i a number of times equal to the out-degree of s_i minus 1; for each head vertex s_i of a conditional branch, only the maximum among the successors' worst-case workloads is added to s_i's WCET. Finally, for all non-head vertices add the worst-case workload of their unique successor to their WCET. The complexity of this algorithm reduces then to $O(|E|)$, i.e., it becomes linear in the size of the graph.

5.4 Schedulability condition

Lemmata 5.1 and 5.3 and the bounds previously computed allow proving the following theorem [10].

Theorem 5.1. *Given a cp-task set globally scheduled on m cores, an upper-bound R_k^{ub} on the response-time of a task τ_k can be derived by the fixed-point iteration of the following expression, starting with $R_k^{ub} = L_k$:*

$$R_k^{ub} \leftarrow L_k + \frac{1}{m}\left(W_k - L_k\right) + \left\lfloor \frac{1}{m} \sum_{\forall i \neq k} \mathcal{X}_i^{ALG} \right\rfloor,$$

where, with global FP:

$$\mathcal{X}_i^{ALG} = \mathcal{X}_i^{FP} = \begin{cases} \mathcal{W}_i(R_k^{ub}), & \forall i < k \\ 0, & otherwise \end{cases} ;$$

with global EDF:

$$\mathcal{X}_i^{ALG} = \mathcal{X}_i^{EDF} = \min\left\{\mathcal{W}_i(R_k^{ub}), \mathcal{I}_{i,k}\right\};$$

and $\mathcal{X}_i^{ALG} = \mathcal{W}_i(R_k^{ub})$ for any work-conserving scheduler.

The schedulability of a cp-task system can then be simply checked using Theorem 5.1 to compute an upper-bound
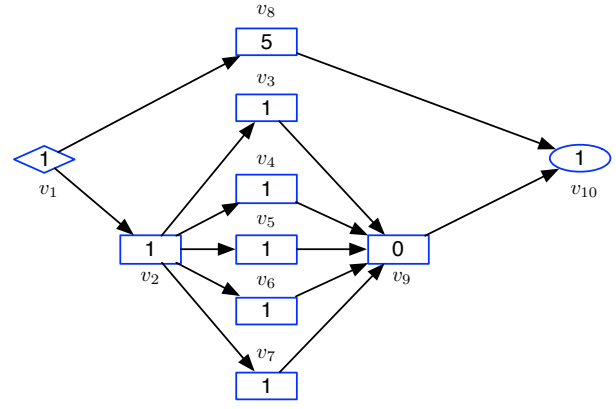

Figure 5: Example of cp-task that shows the pessimism of the upper-bound given in Equation (4).

on the response-time of each task. In the FP case, the bounds are updated in decreasing priority order, starting from the highest priority task. In this case, it is sufficient to apply Theorem 5.1 only once for each task. Instead, in the EDF or general work-conserving cases, multiple rounds may be necessary. All bounds are initially set to $R_k^{ub} = L_k, \forall \tau_k \in \mathcal{T}$. Then, Theorem 5.1 is used to compute a response-time bound for each task τ_k. The procedure continues until either (i) one of the response-time bounds exceeds the corresponding task deadline (returning a negative schedulability result), or (ii) no more update is possible (returning a schedulable condition).

5.5 Improved upper-bounds on intra-task interference

The upper bound given in Lemma 5.3 might be pessimistic. As an example, consider the cp-task τ_k in Figure 5, which executes on a platform composed of $m = 2$ processors. This cp-task has a longest path length of 7 time-units (given by the upper branch), and a worst-case workload $W_k = 8$ time-units (given by the lower branch). When $m = 2$, Equation (4) gives a bound on Z_k of 7.5. However, if the upper branch is taken after the completion of v_1, only the longest path of τ_k would be executed, yielding a value of $Z_k = 7$ time-units. Instead, if the lower branch is taken, only the corresponding portion of the graph would be executed, with an upper-bound of $Z_k \leq 4 + 4/2 = 6$ time-units. Hence, in both cases, the upper-bound computed by (4) would be pessimistic.

This is mainly due to the fact that Equation (4) considers the worst-case situation where, *simultaneously*, i) the critical path of G_k is executed; and ii) the total worst-case workload of τ_k is experienced. However, given the internal structure of the cp-task of Figure 5, this situation can never happen.

The example intuitively suggests that the bound in Equation (4) can be further improved by *jointly* computing the worst-case workload and the longest chain length for each portion of the cp-task, so that both values refer to the same conditional branch. Specifically, for a given chain λ of τ_k, let W_k^λ be the maximum workload attainable by τ_k *when the conditional choices are compatible with λ*. Then, arguing similarly as in Lemma 5.3, we get:

Lemma 5.4.

$$Z_k \leq \mathrm{len}(\lambda_k^*) + \frac{1}{m}(W_k^{\lambda_k^*} - \mathrm{len}(\lambda_k^*))$$

$$\leq \max_{\lambda}\left(\mathrm{len}(\lambda) + \frac{1}{m}(W_k^{\lambda} - \mathrm{len}(\lambda))\right)$$

where λ ranges over all source-sink paths of τ_k.

Algorithm 2 Z_k Bound Computation

1: **procedure** ZBOUND(G, m)
2:     $\sigma \leftarrow$ TOPOLOGICALORDER(G)
3:     $S(v^{\mathrm{sink}}) \leftarrow \{v^{\mathrm{sink}}\}$
4:     $T(v^{\mathrm{sink}}) \leftarrow \{v^{\mathrm{sink}}\}$
5:     $f(v^{\mathrm{sink}}) \leftarrow C^{\mathrm{sink}}$
6:     **for** $v_i \in \sigma$ from sink to source **do**
7:         **if** SUCC$(v_i) \neq \emptyset$ **then**
8:             **if** ISBEGINCOND(v_i) **then**
9:                 $v^* \leftarrow \mathrm{argmax}_{v\in\mathrm{SUCC}(v_i)} C(S(v))$
10:                 $S(v_i) \leftarrow \{v_i\} \cup S(v^*)$
11:                 $u^* \leftarrow \mathrm{argmax}_{u\in\mathrm{SUCC}(v_i)} f(u)$
12:                 $T(v_i) \leftarrow \{v_i\} \cup T(u^*)$
13:                 $f(v_i) \leftarrow C_i + f(u^*)$
14:             **else**
15:                 $S(v_i) \leftarrow \{v_i\} \cup \bigcup_{v\in\mathrm{SUCC}(v_i)} S(v)$
16:                 $u^* \leftarrow \mathrm{argmax}_{u\in\mathrm{SUCC}(v_i)}\big(f(u) +$
17:                     $+\sum_{w\in\mathrm{SUCC}(v_i), w\neq u} C(S(w) \setminus T(u))/m\big)$
18:                 $T(v_i) \leftarrow \{v_i\} \cup T(u^*)$
19:                 $f(v_i) \leftarrow C_i + f(u^*)+$
20:                     $+\sum_{w\in\mathrm{SUCC}(v_i), w\neq u^*} C(S(w) \setminus T(u^*))/m$
21:             **end if**
22:         **end if**
23:     **end for**
24:     **return** $f(v^{\mathrm{source}})$
25: **end procedure**

Algorithm 2 takes as input a given task graph G and m and outputs an upper-bound on the task's Z_k value by computing jointly the worst-case workload and the contributions of different subgraphs of the task. As for Algorithm 1, a topological sorting of the nodes is required (line 2). Three variables for each node v_i are used by the algorithm to store intermediate results: $S(v_i)$ is a set representing the nodes that determine the largest partial workload from v_i till the end of the DAG; $f(v_i)$ stores the bound on the partial Z_k value from node v_i to the end of the DAG, including the full contribution of nodes belonging to the partial longest chain (stored in set $T(v_i)$) and the workload contribution over m cores due to other nodes of the same conditional instance. The computation of the values $S(v_i)$ (lines 3, 10, 15) is exactly as in Algorithm 1.

In the sequel, we focus on the computation of $f(v_i)$ and $T(v_i)$. Since the sink node has no successors, we initialize $T(v^{\mathrm{sink}})$ to $\{v^{\mathrm{sink}}\}$ and $f(v^{\mathrm{sink}})$ to C^{sink}. The algorithm's main loop iterates over the nodes of G in reverse topological order (line 6). If the node under analysis has some successor, different actions are taken depending on whether v_i is the head of a conditional pair or not. In the former case, we compute the successor u^* that maximizes the intermediate upper-bound on Z_k, and set $f(v_i)$ and $T(v_i)$ accordingly. If, instead, a parallel branch is departing from the current node
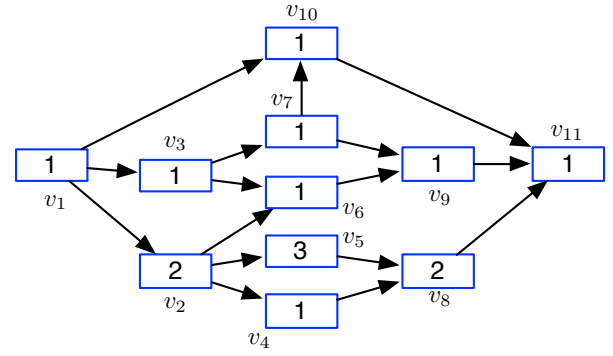

Figure 6: Example of cp-task that shows the two sources of pessimism in the upper-bound of Algorithm 2.

v_i, the workload by all the successors will be transferred to v_i. The goal is to determine the successor u that yields the largest combined value of its partial Z_k bound ($f(u)$) plus the total self-interference from other nodes, which is bounded by

$$\sum_{w\in\mathrm{SUCC}(v_i), w\neq u} C(S(w) \setminus T(u))/m.$$

Note that the set $T(u)$ is subtracted from the set to consider for the self-interfering contribution, because such nodes are already fully accounted for in the term $f(u)$.

The complexity of Algorithm 2 is $O(|V||E|\Delta)$, where Δ denotes the maximum out-degree of a node. In fact, similarly to the analysis of Algorithm 1, the complexity of the algorithm is $O(|V||E|)$, plus the cost of executing the instructions at lines $16 - 17$. The cost of performing such an instruction once is $O(|V|\delta(v_i)^2)$, where $\delta(v_i)$ is the out-degree of node v_i; since $\delta(v_i) \leq \Delta$, it follows that the total cost of the instructions at line $16 - 17$ is

$$O\left(|V|\sum_i \delta(v_i)^2\right) = O\left(|V|\Delta\sum_i \delta(v_i)\right) = O(|V||E|\Delta).$$

This cost dominates, in the worst case, the cost of other operations; hence, the complexity of Algorithm 2 is $O(|V||E|\Delta)$.

5.5.1 A non-redundant upper-bound

The previously introduced upper-bound is still not accurate, because it may lead to account multiple times the interfering contribution of some vertices. In Figure 6 we report an example to intuitively show the two sources of pessimism that affect Algorithm 2. In this example, we assume that the cp-task τ_k in Figure 6 executes on $m = 2$ processors.

When Algorithm 2 examines vertex v_2, it designates v_5 as the vertex u^* that maximizes its partial bound on Z_k plus the intra-task interference from the other successors. In particular, the instruction at lines 19-20 yields

$$f(v_2) = C_2 + f(v_5) + \frac{1}{m}\Big(C(S(v_4) \setminus T(v_5))+$$
$$+ C(S(v_6) \setminus T(v_5))\Big) = 2 + 6 + (1+2)/2 = 9.5. \tag{6}$$

When vertex v_1 is examined, the algorithm selects v_2 as the vertex u^*, and the instruction at lines 19-20 now yields

$$f(v_1) = C_1 + f(v_2) + \frac{1}{m}\Big(C(S(v_3) \setminus T(v_2)+$$
$$+ C(S(v_{10}) \setminus T(v_2))\Big) = 1 + 9.5 + (5+1)/2 = 13.5, \quad (7)$$

which also represents the final output of the algorithm. Note, however, that the contribution of vertices v_6 and v_9 has been accounted twice: first, as intra-task interference on v_2 (i.e., inside set $S(v_6) \setminus T(v_5)$ in Equation (6)), and then as intra-task interference on v_1 (i.e., inside set $S(v_3) \setminus T(v_2)$) in Equation (7). Analogously, also the contribution of v_{10} is accounted twice by Algorithm 2: first, among the successors of v_7, and then, among the successors of v_1. We identify two sources of pessimism in Algorithm 2.

First source of pessimism. When examining vertex u, vertices reachable from successors of u (other than u^*) that have already been considered when examining u^* may be counted multiple times (as v_6 and v_9 in the example above).

This problem can be overcome by replacing the summation at lines 17 and 20 of Algorithm 2 with the expression

$$\sum_{w \in \text{SUCC}(v_i), w \neq u} C(S(w) \setminus S(u))/m.$$

The new expression differs from the one given in Algorithm 2 because it subtracts $S(u)$ (instead of $T(u)$) from the set to consider for the self-interfering contribution, since such vertices have been already fully accounted for in the term $f(u)$, either as part of the partial longest chain or as intra-task interference on u. Since the set $S(u)$ contains vertices from both sets, the new formulation allows discarding all vertices that have already been accounted when examining u^*. Indeed, applying this improvement to the above example, the contribution from vertices in $S(v_3) \setminus S(v_2)$ will be computed (instead of $S(v_3) \setminus T(v_2)$) and added to $f(v_1)$; since $S(v_2)$ also contains v_6 and v_9, their contribution will not be counted multiple times, leading to a tighter bound on Z_k given by

$$f(v_1) = C_1 + f(v_2) + \frac{1}{m}(C(S(v_3) \setminus S(v_2))+$$
$$+ C(S(v_{10}) \setminus S(v_2)) = 1 + 9.5 + (3+1)/2 = 12.5.$$

Second source of pessimism. When examining vertex u, vertices reachable from successors of u (other than u^*) that have been previously examined may be accounted multiple times (as vertex v_{10} in the example above).

This second issue can be prevented by using set operations instead of the sum operand at lines 17 and 20 of Algorithm 2, which leads to the expression

$$\frac{1}{m} \cdot C \left(\bigcup_{w \in \text{SUCC}(v_i), w \neq u} S(w) \setminus S(u) \right) =$$
$$= \frac{1}{m} \cdot C\left(S(v_i) \setminus S(u) \setminus \{v_i\} \right).$$

Applying this refined expression to the example above, the contribution of v_{10} is finally accounted only once. It can be easily verified that the final bound on Z_k is now 12.

The following lemma proves that with the improvements to Algorithm 2 discussed above, the bound on Z_k that we obtain is *non-redundant*, in the sense that it does not account the contribution of any vertex more than once.

Lemma 5.5. *Algorithm 2 with the improvements discussed above yields a non-redundant upper-bound on Z_k.*

Proof. The only steps in the updated algorithm where the contribution of a vertex may be added more than once are line 13 and lines 19-20. At line 13, only the contributions of a vertex v_i and of some of its descendants (those in $S(u^*) \subset S(v_i) \setminus \{v_i\}$) are being added, but $v_i \notin S(u^*)$, so the two sets are disjoint. At lines 19-20, a vertex v_i, and some of its descendants in $S(u^*)$ (line 19) and in $S(v_i) \setminus S(u^*) \setminus \{v_i\}$ (the updated line 20) are being considered. Again, these sets are disjoint and no double counting can occur. □

With a reasoning similar to the one in Section 5.5 one obtains that the complexity of the updated Algorithm 2 is

$$O\left(|V| \sum_i \delta(v_i)|V| \right) = O\left(|V|^2 \sum_i \delta(v_i) \right) = O(|V|^2|E|).$$

6 EXPERIMENTAL RESULTS

In this section, we evaluate the performance of the response-time analysis approach presented in Section 5 in comparison to existing approaches, both for global FP and global EDF scheduling. The metric adopted for the comparison is the number of schedulable task-sets. Our results reveal that our schedulability tests are not only superior to existing approaches targeting conditional parallel tasks, but are also able to outperform state-of-the-art techniques when the task model is restricted to the particular case with no conditional branches, i.e., when classic DAG tasks are considered.

All the algorithms compared in our simulations have been implemented as MATLAB® code. Since a considerable effort was required to design a sufficiently general setting for evaluating the performance of graph-based task systems, we created a public repository [4] where our code can be freely downloaded to test the schedulability performance of conditional/parallel task systems.

6.1 Generation of cp-tasks

We refer to [23] as a baseline for our random cp-task generation. In that work, multiple nested levels of conditional branches are recursively generated by expanding non-terminal vertices (called *blocks*) either to terminal vertices or to conditional subgraphs, until reaching a maximum recursion depth. The derivation rules given in [23] need to be extended by considering that non-terminal vertices can be expanded to either terminal vertices, parallel subgraphs or conditional subgraphs. We specify the probabilities that control such three events as p_{term}, p_{par}, and p_{cond}, respectively, requiring that $p_{term} + p_{par} + p_{cond} = 1$.

Additionally, the maximum number of branches of parallel and conditional subgraphs are indicated as n_{par} and n_{cond}, respectively. Hence, whenever a parallel subgraph is generated, the number of its branches is uniformly selected

in the interval $[2, n_{par}]$. Analogously, whenever a non-terminal vertex is expanded to a conditional subgraph, the number of branches is uniformly selected in $[2, n_{cond}]$.

Unfortunately, this methodology only allows generating series-parallel graphs. Therefore, since in this work we deal with a more general class of task graphs (i.e., cp-DAGs that respect the structural restrictions imposed by conditional nodes, as described in Section 3), we need to randomly add edges between pairs of nodes to obtain cp-DAG tasks. In our methodology, an edge is added between two vertices with a certain probability p_{add}, provided that any of such edges respects the structural restrictions dictated by our cp-task model (see Definition 3.1).

Each cp-task τ_k is generated as follows:

- the WCET $C_{k,j}$ of each vertex $v_{k,j}$ is randomly selected as a positive integer in the interval $[1, 100]$;
- then, L_k and W_k are computed;
- the period T_k is uniformly selected as an integer in the interval $[L_k, W_k/\beta]$, where $\beta \leq 1$ is a parameter that controls the minimum cp-task utilization. In this way, the utilization of each cp-task is uniformly distributed in the interval $[\beta, W_k/L_k]$, where its right endpoint, corresponding to the maximum possible utilization, is given by the average degree of parallelism of the cp-task;
- finally, the relative deadline D_k is uniformly selected as an integer in the interval $[L_k, T_k]$.

Whenever a specific utilization is targeted, we repeatedly add tasks until the desired cumulative utilization is exceeded. Then, the period of the last task is increased to match the desired total system utilization. In all our experiments, we set the maximum recursion depth to 3 for each cp-task.

6.2 Evaluation of cp-tasks

A first comparison is performed to evaluate our response-time analysis approach against the only two works in the literature that target the scheduling of DAG tasks with conditional branches, i.e., [16] and [8].

The first work proposes a transformation of the conditional DAG task into a synchronous parallel task. Existing schedulability tests can then be applied on the transformed task-set. Hence, we evaluate our response-time analysis in a global fixed-priority setting by considering the test for synchronous parallel tasks proposed by Maia et al. [21], which, to the best of our knowledge, outperforms the others. This schedulability test will be referred to as COND-SP, while we will denote as RTA-FP-a (resp., RTA-FP-b) our response-time analysis for global FP when the intra-task interference is bounded as described in Algorithm 2, with (resp., without) the improvements discussed in Section 5.5.1.

For this class of experiments, we set $p_{cond} = 0.4$, $p_{par} = 0.4$, $p_{term} = 0.2$, $p_{add} = 0.1$, $n_{cond} = 2$, $n_{par} = 6$, $\beta = 0.1$. When not explicitly specified, we assume that tasks priorities are assigned according to a Deadline Monotonic (DM) ordering. Also, for each experiment, 1000 task-sets have been generated for each value on the x−axis.

In the first set of experiments, we varied the total system utilization $U_{\mathcal{T}}$ in the range $[0, m]$. The number of

schedulable task-sets obtained when $m = 4$ is reported in Figure 7. The trend observed in the figure, which is representative of the general behavior, shows that RTA-FP clearly outperforms COND-SP for any value of $U_{\mathcal{T}}$.

In the second set of experiments, we varied the number of processors. The results for $U_{\mathcal{T}} = 2$ are reported in Figure 8. For low values of m, RTA-FP significantly outperforms COND-SP, while when the number of processors is high both tests are able to successfully schedule nearly all task-sets.

In the third set of experiments, the number of tasks n is varied in the range $[1, 20]$. Since n has now a fixed value in each experiment, individual cp-task utilizations have been computed using UUnifast [11]. Figure 9 illustrates the results for $m = 4$ and $U_{\mathcal{T}} = 2$. While the performance of the two tests is comparable when n is small, RTA-FP exhibits a substantial improvement over COND-SP when $n \geq 4$. Furthermore, RTA-FP achieves full schedulability for large values of n. This conforms to the intuition that scheduling a large number of "light" tasks is easier than scheduling fewer "heavy" tasks. Instead, COND-SP achieves its maximum schedulability performance for $n = 5$ and then degrades for higher values of n. This stems from the pessimism introduced by the transformation technique in [16], which increases when the number of tasks is higher.

Another drawback of COND-SP concerns its complexity, since it requires enumerating all the conditional flows of each cp-task, which are exponentially many in the nesting level of conditional statements. Instead, our approach relies on efficient algorithms that explicitly deal with conditional branches in psuedo-polynomial time. The main consequence is that the running time of COND-SP is often quite prohibitive, while RTA-FP runs very fast (i.e., in the order of milliseconds).

Other experiments have been performed by varying the connectivity degree of the tasks, i.e., by changing the probability p_{add}, while keeping $U_{\mathcal{T}}$, n and m constant. However, the results do not show any particular trend, as the schedulability ratio remains almost constant for all possible values of p_{add}, hence the corresponding plots are not reported. In other experiments, we have also varied the composition of the cp-tasks, by acting on p_{cond} and p_{par}, while keeping their sum constant. Again, no interesting trend has been identified. Such results can be explained considering that the computation of the interference produced by each cp-task highly depends on its worst-case workload, and its computation (see Algorithm 1) is not very much influenced by the degree of parallelism of the cp-task. Furthermore, the notion of worst-case workload represents an effective way of abstracting from the different conditional flows. This explains why the composition of tasks in terms of conditional/parallel branches does not significantly affect the schedulability performance.

Another class of experiments has been performed to compare our response-time analysis under global EDF against the work by Baruah et al. [8]. That work proposes an efficient method to transform any conditional DAG task into a non-conditional DAG task on which the existing tests in [12] and [7] can be applied. Since the test in [7] analytically dominates the one in [12], we compared our RTA-EDF approach against the test in [7], after performing
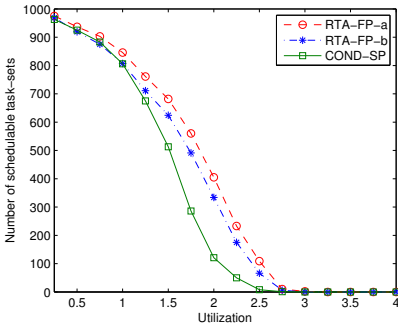
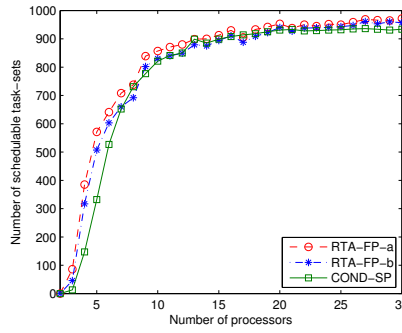Figure 7: RTA-FP as a function of $U_{\mathcal{T}}$, with $m = 4$ and constrained deadlines.

Figure 8: RTA-FP as a function of m, with $U_{\mathcal{T}} = 2$ and constrained deadlines.

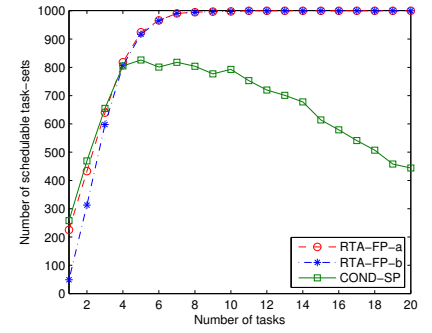
Figure 9: RTA-FP as a function of n, with $m = 4$, $U_{\mathcal{T}} = 2$ and constrained deadlines.

the transformation described in [8]. As before, we will denote as RTA-EDF-a (resp., RTA-EDF-b) our response-time analysis approach under global EDF when the intra-task interference is bounded as described in Algorithm 2, with (resp., without) the improvements discussed in Section 5.5.1.

In Figure 10, we report the number of schedulable task-sets with $m = 4$ when $U_{\mathcal{T}} \in [0, 4]$. At all utilization levels, our approach is able to successfully schedule more task-sets. The same trend can be observed when varying the number of cores. Figure 11 illustrates representative results when $U_{\mathcal{T}} = 2$ and $m \in [2, 30]$. Our approach performs significantly better for any value of m: in particular, while RTA-EDF is able to schedule nearly all task-sets for a large value of m, the approach by Baruah et al. cannot admit a large share of the generated task-sets even when the number of cores significantly increases. Finally, Figure 12 shows the comparison between the two approaches when varying the number of tasks ($n \in [1, 20]$), with $U_{\mathcal{T}} = 2$ and $m = 8$. While RTA-EDF admits almost all task-sets when $n \geq 7$, the approach by Baruah et al. achieves full schedulability only for a very large value of n.

6.3 Evaluation of classic DAG tasks

The following experiments restrict our task model to the case where conditional statements are not modeled, i.e., classic DAG tasks are considered. This particular setting allowed us to evaluate the improvement of our response-time analysis for global EDF over existing schedulability tests targeting systems of sporadic DAG tasks.

The random task generator described above can be simply adapted to generate classical DAG tasks by setting $p_{cond} = 0$ and requiring that $p_{term} + p_{par} = 1$. Specifically, we set: $p_{par} = 0.8$, $p_{term} = 0.2$, $p_{add} = 0.1$, $n_{par} = 6$, $\beta = 0.1$.

We compared our RTA-EDF test against three schedulability tests for systems of sporadic DAG tasks scheduled according to global EDF:

- the test by Baruah [7], which analytically dominates the one in [12];
- the test by Li et al. [18], based on capacity augmentation bound;
- the test by Qamhieh et al. [24] that takes into account the internal structure of the DAG.

Since the test in [18] assumes implicit deadlines, we report all the results under that setting for consistency, although similar results have been obtained also in the general case of constrained deadlines. We do not to plot the results of the test in [24], because its performance was very poor in all observed configurations. This stems from the fact that the analysis in [24] is mainly focused with improving the minimum processor speed that guarantees schedulability under global EDF, rather than ensuring a good schedulability performance.

Figure 13 illustrates the number of schedulable task-sets in the case of $m = 8$ and varying utilization $U_{\mathcal{T}} \in [0, 8]$. While the performance of RTA-EDF drops around $U_{\mathcal{T}} = 5$, the breakdown utilization of the other approaches is significantly lower.

Figure 14 illustrates the performance of RTA-EDF when $U_{\mathcal{T}} = 2$ and m is varied in $[2, 30]$. As evident from the figure, RTA-EDF substantially outperforms the other tests, as it requires a significantly lower number of cores (around 5) to schedule most task-sets, while the test in [7] typically requires twice that number, and it cannot admit any task-set when $m < 5$. The behavior of the test in [18] is even worse, since a large share of the generated task-sets are not deemed schedulable even if a very large number of cores is available. This result indeed reflects the analytical formulation of the test given in [18].

Figure 15 reports the results obtained when varying the number of tasks ($n \in [1, 20]$), with $m = 8$ and $U_{\mathcal{T}} = 2$. As before, our approach substantially outperforms the others for any value of n. The test in [7] shows a slowly degrading trend for high values of n. Instead, the one in [18] is favorably impacted by increasing n, since by keeping the total utilization constant the individual critical path lengths are reduced in average, which is beneficial for the outcome of the test.

This class of experiments clearly shows that the effectiveness of our schedulability analysis goes beyond conditional task structures, as it is able to significantly tighten the schedulability of non-conditional DAG task systems as well.

6.4 Evaluation of RTA-FP vs. RTA-EDF

We now compare the performance of our response-time analysis for global FP and global EDF as a function of
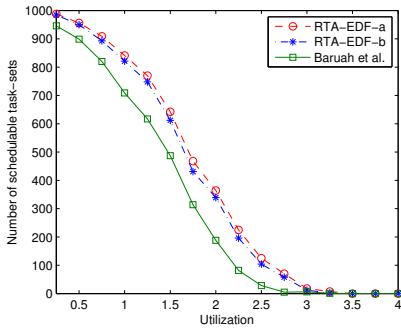
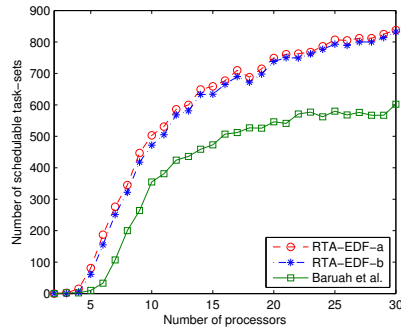Figure 10: RTA-EDF as a function of $U_{\mathcal{T}}$, with $m = 8$ and constrained deadlines.

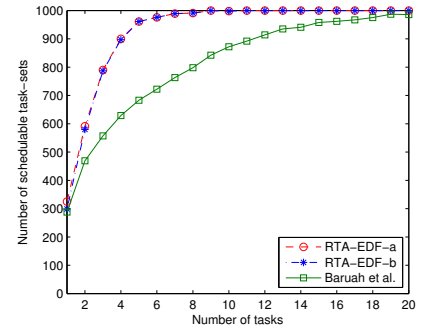Figure 11: RTA-EDF as a function of m, with $U_{\mathcal{T}} = 2$ and constrained deadlines.

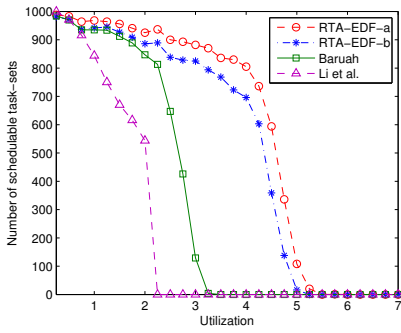Figure 12: RTA-EDF as a function of n, with $m = 8$, $U_{\mathcal{T}} = 2$ and constrained deadlines.

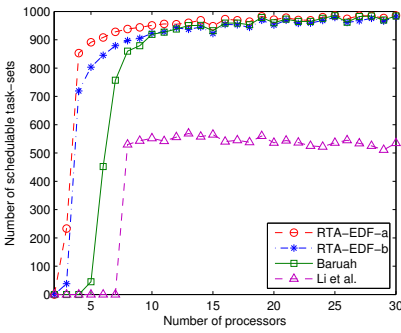Figure 13: RTA-EDF as a function of $U_{\mathcal{T}}$, with $m = 8$ and implicit deadlines.

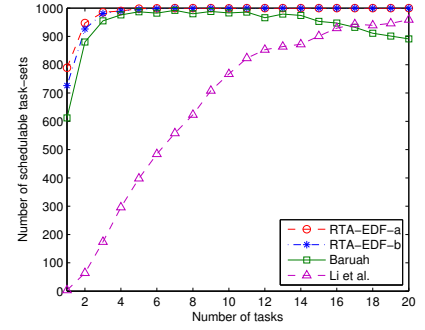Figure 14: RTA-EDF as a function of m, with $U_{\mathcal{T}} = 2$ and implicit deadlines.

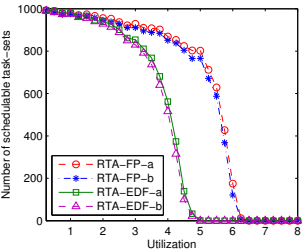Figure 15: RTA-EDF as a function of n, with $m = 8$, $U_{\mathcal{T}} = 2$ and implicit deadlines.

Figure 16: RTA-FP and RTA-EDF as a function of $U_{\mathcal{T}}$ in the case of cp-tasks, with $m = 8$ and implicit deadlines.
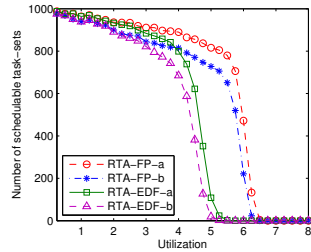
Figure 17: RTA-FP and RTA-EDF as a function of $U_{\mathcal{T}}$ in the case of classic DAG tasks, with $m = 8$ and implicit deadlines.

the system utilization $U_{\mathcal{T}}$. In the following experiments, we consider $m = 8$ and restrict to the case of implicit deadlines. Figure 16 and 17 report the results for the case of cp-tasks and classic DAG tasks, where the parameters for the task generation have been set as in Section 6.2 and 6.3, respectively. In both cases, RTA-FP is able to guarantee ~75% of the total system utilization, being able to detect a significant amount of schedulable task-sets until $U_{\mathcal{T}} = 6$, while the performance of RTA-EDF drops at a smaller utilization level (around $U_{\mathcal{T}} = 5$), which corresponds to ~63% of the total system utilization. As a salient trait, in

both cases the performance of the schedulability test for global FP is significantly superior to the corresponding test for EDF, resembling the behavior observed in the case of multiprocessor response-time analysis for sequential task systems [10]. This is mainly due to the fact that in the case of FP the interference from lower-priority tasks can be neglected, which does not hold when EDF is used. Therefore, the schedulability performance achieved by our RTA-FP test is able to largely compensate the disadvantage of FP vs. EDF in terms of absolute scheduling performance. As a final remark, we observe that the performance of our response-time analysis in the conditional case (Figure 16) is very similar to the case when no conditional vertices are assumed (Figure 17). This fact leads to the conclusion that the concept of worst-case workload is a meaningful characterization of conditional-parallel real-time workload, and, more in general, that the presented analysis is able to effectively integrate conditional statements with parallel execution flows.

7 CASE STUDY

After evaluating the effectiveness of the proposed approach by extensive experiments on synthetically-generated workload, we now turn to a realistic application composed of three OpenMP programs:

- *Wavefront*, a matrix processing algorithm in which the matrix is decomposed into smaller blocks and

Table 2: Case study description

Benchmark	L_k	W_k	D_k	T_k	p_k
Wavefront	1635	3252	2000	2600	High
ESA	5784	48075	17600	22000	Medium
Cholesky	1664	3812	17000	25000	Low

traversed through its diagonals. The program takes as input the block size bs;

- *Cholesky*, a factorization algorithm used for efficient numerical solution of systems of linear equations or Monte Carlo simulation techniques. The program takes as input the number of blocks nb per dimension, and performs the factorization on block submatrices;

- *ESA*, an infrared pre-processing application developed by the European Space Agency (ESA) and used by H2RG sensors to measure the red-shift of galaxies.

Our objective is to determine what is the minimum number of processors required to successfully schedule the considered application. To this aim, we need to determine the cp-task structures corresponding to the considered OpenMP programs. Some recent works [27], [29], [30] explored the connections between the OpenMP tasking model and the existing graph-based task models, to enable the timing analysis of real-time parallel code. In particular, the work in [30] proposed a tool that takes as input an OpenMP program and extracts the corresponding DAG structure. The considered OpenMP benchmarks have been processed by that tool to extract the corresponding cp-DAGs. For the first two benchmarks (*Wavefront* and *Cholesky*), we consider a restricted input set, namely $bs \in \{1, 2, 4\}$ and $nb \in \{1, 2, 4\}$, respectively. Different computation is performed by the two programs depending on the input value, hence their corresponding cp-DAGs have a conditional head node and three branches, each having a different number of vertices. Conversely, *ESA* is a non-conditional (i.e., classic) DAG with a very high level of parallelism.

Table 2 describes the resulting task-set, reporting for each task its critical path length L_k, worst-case workload W_k, relative deadline D_k, period T_k and static priority level p_k.

Under global FP scheduling, the task-set described above can be successfully scheduled on a platform consisting of $m = 6$ processors, being $R_1 = 1904.5 \leq D_1$, $R_2 = 16626 \leq D_2$ and $R_3 = 13286 \leq D_3$. Conversely, when using COND-SP (i.e., the state-of-the-art approach based on the transformation proposed in [16], on which the test in [21] is applied), $m = 11$ processors are required to schedule the given application. Hence, our test is able to almost halve the required computational resources. Also, notice that the task-set in Table 2 would not be deemed schedulable by RTA-FP on $m = 6$ processors if a Deadline Monotonic priority ordering was considered ($m = 7$ processors would be necessary in that case).

If, instead, global EDF scheduling is assumed, our RTA-EDF approach requires $m = 8$ processors to deem the task-set schedulable, which is the same result obtained when using the approach by Baruah et al. (i.e., the transformation proposed in [8] followed by the test in [7]). However, a

significant schedulability advantage over that approach has been observed in the average case (see Figures 10-12).

8 CONCLUSIONS

This paper proposed a new task model, the cp-task model, that generalizes the classic sporadic DAG task model by integrating conditional branches. Such an additional information is exploited by the schedulability analysis to derive a tighter estimation of the interfering contributions, by discriminating their level of parallelism depending on the conditional path undertaken. The topological structure of a cp-task graph has been formally characterized by specifying which connections are allowed between conditional and non-conditional nodes. Then, a schedulability analysis has been derived to compute a safe upper-bound on the response-time of each task in pseudo-polynomial time. Besides its reduced complexity, the proposed analysis has the advantage of requiring only two parameters to characterize the complex structure of the conditional graph of each task: the worst-case workload and the length of the longest path. Algorithms have also been proposed to derive these parameters from the DAG structure in polynomial time.

Simulation experiments carried out with randomly generated cp-task workloads clearly show that the proposed approach does not only improve over previously proposed solutions for conditional DAG tasks, but can also be used to significantly tighten the schedulability analysis of classic (non-conditional) sporadic DAG task systems.

REFERENCES

[1] *Kalray.* http://www.kalrayinc.com/.

[2] *Parallela.* http://www.parallela.org/.

[3] *Texas Instruments. The 66AK2H12 Keystone II Processor.* http://www.ti.com/product/66AK2H12.

[4] *A MATLAB® implementation of schedulability tests for conditional and parallel task systems.* http://retis.sssup.it/~al.melani/downloads/cptasks.zip, 2015.

[5] B. Andersson and D. de Niz. Analyzing global-EDF for multiprocessor scheduling of parallel tasks. In *16th International Conference on Principles of Distributed Systems (OPODIS 2012), Rome, Italy, December 18-20, 2012*, pages 16–30, 2012.

[6] 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 *25th Euromicro Conference on Real-Time Systems (ECRTS 2013), Paris, France*, pages 215–224, July 9-12, 2013.

[7] S. Baruah. Improved multiprocessor global schedulability analysis of sporadic DAG task systems. In *26th Euromicro Conference on Real-Time Systems (ECRTS 2014)*, Madrid, Spain, July 8-11, 2014.

[8] S. Baruah, V. Bonifaci, and A. Marchetti-Spaccamela. The global EDF scheduling of systems of conditional sporadic DAG tasks. In *27th Euromicro Conference on Real-Time Systems (ECRTS 2015)*, Lund, Sweden, July 7-10, 2015.

[9] S. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, L. Stougie, and A. Wiese. A generalized parallel task model for recurrent real-time processes. In *33rd IEEE Real-Time Systems Symposium (RTSS 2012)*, San Juan, Puerto Rico, December 4-7, 2012.

[10] M. Bertogna and M. Cirinei. Response-time analysis for globally scheduled symmetric multiprocessor platforms. In *28th IEEE Real-Time Systems Symposium (RTSS 2007), Tucson, Arizona, USA*, December 3-6, 2007.

[11] E. Bini and G. C. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2):129–154, 2005.

[12] V. Bonifaci, A. Marchetti-Spaccamela, S. Stiller, and A. Wiese. Feasibility analysis in the sporadic DAG model. In *25th Euromicro Conference on Real-Time Systems (ECRTS 2013)*, Paris, France, July 9-12, 2013.

[13] A. Burns and S. K. Baruah. Sustainability in real-time scheduling. *Journal of Computing Science and Engineering*, 2(1):74–97, 2008.

[14] H. S. Chwa, J. Lee, K.-M. Phan, A. Easwaran, and I. Shin. Global EDF schedulability analysis for synchronous parallel tasks on multicore platforms. In *25nd Euromicro Conference on Real-Time Systems (ECRTS 2013)*, Paris, France, July 9-12, 2013.

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

[16] J. C. Fonseca, V. Nélis, G. Raravi, and L. M. Pinho. A multi-DAG model for real-time parallel applications with conditional execution. In *30th ACM/SIGAPP Symposium On Applied Computing (SAC 2015)*, Salamanca, Spain, April 13-17, 2015.

[17] K. Lakshmanan, S. Kato, and R. Rajkumar. Scheduling parallel real-time tasks on multi-core processors. In *31st IEEE Real-Time Systems Symposium*, San Diego, CA, November 30 - December 3, 2010. IEEE.

[18] J. Li, K. Agrawal, C. Lu, and C. Gill. Analysis of global EDF for parallel real-time tasks. In *25th Euromicro Conference on Real-Time Systems (ECRTS 2013)*, Paris, France, July 9-12, 2013.

[19] J. Li, J. Chen, K. Agrawal, C. Lu, C. D. Gill, and A. Saifullah. Analysis of federated and global scheduling for parallel real-time tasks. In *26th Euromicro Conference on Real-Time Systems (ECRTS 2014), Madrid, Spain*, July 8-11, 2014.

[20] J. Li, Z. Luo, D. Ferry, K. Agrawal, C. Gill, and C. Lu. Global EDF scheduling for parallel real-time tasks. *Real-Time Systems*, 51(4):395–439, 2015.

[21] C. Maia, M. Bertogna, L. Nogueira, and L. M. Pinho. Response-time analysis of synchronous parallel tasks in multiprocessor systems. In *22nd International Conference on Real-Time Networks and Systems (RTNS 2014)*, Versailles, France, October 8-10, 2014.

[22] G. Nelissen, V. Berten, J. Goossens, and D. Milojevic. Techniques optimizing the number of processors to schedule multi-threaded tasks. In *24th Euromicro Conference on Real-Time Systems (ECRTS 2012), Pisa, Italy*, July 11-13, 2012.

[23] B. Peng, N. Fisher, and M. Bertogna. Explicit preemption placement for real-time conditional code. In *26th Euromicro Conference on Real-Time Systems (ECRTS 2014)*, Madrid, Spain, July 8-11, 2014.

[24] M. Qamhieh, F. Fauberteau, L. George, and S. Midonnet. Global EDF scheduling of directed acyclic graphs on multiprocessor systems. In *21st International Conference on Real-Time Networks and Systems (RTNS 2013)*, Sophia Antipolis, France, October 16-18, 2013.

[25] A. Saifullah, K. Agrawal, C. Lu, and C. D. Gill. Multi-core real-time scheduling for generalized parallel task models. In *32nd IEEE Real-Time Systems Symposium (RTSS 2011)*, Vienna, Austria, November 29 - December 2, 2011.

[26] A. Saifullah, D. Ferry, J. Li, K. Agrawal, C. Lu, and C. D. Gill. Parallel real-time scheduling of DAGs. *IEEE Trans. Parallel Distrib. Syst.*, 25(12):3242–3252, 2014.

[27] M. A. Serrano, A. Melani, R. Vargas, A. Marongiu, M. Bertogna, and E. Quiñones. Timing characterization of OpenMP4 tasking model. In *10th ACM/IEEE International Conference on Compilers, Architectures and Synthesis of Embedded Systems (CASES)*, Amsterdam, The Netherlands, October 4-9, 2015.

[28] J. Valdes, R. E. Tarjan, and E. L. Lawler. The recognition of series parallel digraphs. *SIAM Journal on Computing*, 11(2):298–313, 1982.

[29] R. Vargas, E. Quiñones, and A. Marongiu. OpenMP and timing predictability: A possible union? In *18th Design, Automation and Test in Europe Conference (DATE)*, Grenoble, France, March 9-13, 2015.

[30] R. Vargas, S. Royuela, M. A. Serrano, X. Martorell, and E. Quiñones. A lightweight OpenMP4 run-time for embedded systems. In *21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, Macao, China, January 25-28, 2016.

Alessandra Melani is a PhD student at the ReTiS Lab of the Scuola Superiore Sant'Anna, Pisa (Italy). She received a B.S. in Computer Engineering with honors from University of Pisa in 2011, and a double M.S. in Computer Science and Engineering with honors from University of Trento and Scuola Superiore Sant'Anna in 2013. In 2015, she visited the Department of Computer Science of the University of Illinois at Urbana-Champaign, working on co-scheduling algorithms for multi-core real-time systems. Her research interests are in scheduling algorithms, schedulability analysis and optimization of single- and multi-processor systems.

Marko Bertogna is Associate Professor at the University of Modena, Italy. He previously was Assistant Professor at the Scuola Superiore Sant'Anna of Pisa, Italy, where he also received (cum laude) a Ph.D. in Computer Engineering. He has authored over 60 papers in international conferences and journals in the field of real-time and multiprocessor systems, receiving seven Best Paper Awards and one Best Dissertation Award. He served in the Program Committees of the major international conferences on real-time systems. He is Senior Member of the IEEE.

Vincenzo Bonifaci received the Laurea degree in computer engineering from Sapienza University of Rome, Italy, in 2003 and the PhD degree from Sapienza University of Rome and from the Technische Universiteit Eindhoven, the Netherlands, in 2007. He is a permanent researcher at the Institute for Systems Analysis and Informatics of the Italian National Research Council (IASI-CNR). His research interests are in graph algorithms, scheduling and optimization.

Alberto Marchetti-Spaccamela received the degree in electronic engineering from Sapienza Università di Roma in 1977. Since 1991 he is professor at Sapienza Università di Roma. Previously he was a visiting scholar at UC Berkeley and professor at Università dell'Aquila. His research and teaching interests are in scheduling algorithms, approximation and online algorithms, graph and combinatorial algorithms, and bioinformatics.

Giorgio Buttazzo is full professor of Computer Engineering at the Scuola Superiore Sant'Anna of Pisa. He graduated in Electronic Engineering at the University of Pisa in 1985, received a M.S. degree in Computer Science at the University of Pennsylvania in 1987, and a Ph.D. in computer engineering at the Scuola Superiore Sant'Anna of Pisa in 1991. He has authored 7 books on real-time systems and over 200 papers in the field of real-time systems, robotics, and neural networks.