Response-Time Analysis of Bundled Gang Tasks Under Partitioned FP Scheduling

Veronica Rispo, *Student Member, IEEE,* Federico Aromolo, Daniel Casini, *Member, IEEE,* and Alessandro Biondi *Member, IEEE*

Abstract—The study of parallel task models for real-time systems has become fundamental due to the increasing computational demand of modern applications. Recently, gang scheduling has gained attention for improving performance in tightly synchronized parallel applications. Nevertheless, existing studies often overestimate computational demand by assuming a constant number of cores for each task. In contrast, the bundled model accurately represents internal parallelism by means of a string of segments demanding for a variable number of cores. This model is particularly relevant to modern real-time systems, as it allows transforming general parallel tasks into bundled tasks while preserving accurate parallelism. However, it has only been analyzed for global scheduling, which carries analytical pessimism and considerable run-time overheads. This paper introduces two response-time analysis techniques for parallel real-time tasks under partitioned, fixed-priority gang scheduling under the bundled model, together with a set of specialized allocation heuristics. Experimental results compare the proposed methods against state-of-the-art approaches.

Index Terms—Real-time systems, multiprocessor scheduling, parallel tasks, gang scheduling, partitioned scheduling.

1 Introduction

The study of parallel real-time task models has gained attention due to the increased computational power required by cyber-physical systems. Notable examples are emerging applications in autonomous driving [1, 2], Industry 4.0 [3], robotics [4], and artificial intelligence [5]. These applications are often characterized by tasks composed of parallel computations and exhibit a varying degree of parallelism during execution while requiring to meet real-time constraints. Different approaches have been proposed over the years to schedule real-time parallel workloads, both in terms of task models and scheduling algorithms.

Among them, the gang scheduling paradigm is particularly interesting as it has been shown to improve overall performance by reducing communication and context-switch delays [6]. Essentially, it consists of grouping tasks in "gangs", where each gang needs to be co-scheduled simultaneously on a set of cores. Gang scheduling also showed benefits in achieving less pessimistic worst-case execution times (WCETs) by managing shared hardware resource contention [7]. Moreover, it is supported by widely used parallel computing paradigms such as MPI and OpenMP.

However, existing works on real-time gang partitioned scheduling often assume the so-called *rigid* model where the number of cores required by a task remains constant during the task execution [8, 9], disregarding the parallel graph structure of the task. This leads to overestimating the demand for computational resources, as several applications require a variable degree of parallelism during execution.

V. Rispo, F. Aromolo, D. Casini, and A. Biondi are with the Tecip Institute, Scuola Superiore Sant'Anna, Pisa, Italy. F. Aromolo, D. Casini and A. Biondi are also with the Department of Excellence in Robotics and AI, Scuola Superiore Sant'Anna, Pisa, Italy.
E-mail:{v.rispo, f.aromolo, d.casini, al.biondi}@santannapisa.it
Manuscript received April 19, 2021; revised August 16, 2021.

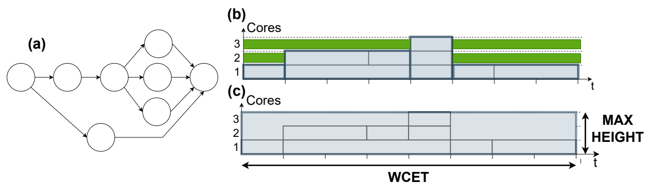

Fig. 1: Example of a parallel task (inset (a)) executed under the bundled (inset (b)) and rigid (inset (c)) models. (c) shows how the rigid model does not account for the varying degrees of parallelism of the task, modeling the execution by relying on the maximum number of cores only. Instead, (b) faithfully models the parallelism of the task, allowing the potential scheduling of other tasks in the time intervals in which some of the cores are not needed (highlighted in green).

These limitations are overcome in the *bundled* model [10], where tasks consist of bundles (or segments), each requiring a different number of cores (the *height* of the bundle), thus more accurately accounting for the varying degrees of parallelism exhibited by tasks over time, as shown in Fig. 1.

When scheduling tasks on a multicore platform, a key decision is whether to follow a *global* or *partitioned* approach [11]. In global scheduling, tasks can migrate across cores, while partitioned scheduling statically assigns tasks to cores, disallowing migration.

Both approaches have advantages and disadvantages: global scheduling offers transparency to users without requiring upfront allocation and tends to have shorter average-case response times. On the other hand, partitioned scheduling provides better timing predictability, control over microarchitectural effects like memory contention [12], and reduced context-switch and migration overheads [13], but requires dealing with the design-time allocation problem of finding a suitable assignment of each computational activity to the available processing cores [11]. Furthermore, for sequential tasks, it has been formally proven that popular analysis techniques for global scheduling are analytically dominated by partitioned scheduling under fixed-priority [14], earliest deadline first (EDF), and first-in-first-out (FIFO) scheduling [15]. These negative results for global scheduling renewed interest in partitioned scheduling. Overall, although global scheduling can improve the average-case timing performance thanks to its dynamic load balancing capabilities, partitioned scheduling has been shown to provide enhanced worst-case behavior and improved predictability by enabling a more precise timing analysis. However, partitioned scheduling is generally less studied in the literature for parallel task models.

Overall, to the best of the authors' knowledge, no previous work addressed the problem of deriving a schedulability analysis for a set of gang-scheduled parallel tasks executing under the bundled model in the case of partitioned scheduling, despite the aforementioned benefits and the enhanced modeling accuracy offered by the bundled execution model.

The case of gang tasks scheduled under partitioned scheduling has only been investigated by Ueter et al. [9]; however, their work only applies to the more restrictive rigid execution model. On the other hand, the bundled execution model has been studied only for global scheduling [10]. With respect to the problems addressed in the aforementioned works, the schedulability analysis problem for the case of bundled gang tasks under partitioned scheduling introduces an additional layer of complexity due to the difficulty in precisely identifying the interfering workload for each bundle of the task under analysis. Specifically, the main challenges in effectively solving the analysis problem consist in **(i)** precisely determining the set of interfering higher-priority bundles for each bundle of the analyzed task and **(ii)** ensuring that the interference generated by a higher-priority bundle is not accounted for multiple times when analyzing each of the bundles composing the analyzed task. Moreover, no specialized allocation strategy has been proposed to efficiently assign each bundle to a suitable set of cores under the partitioned bundled model.

Therefore, we highlight a significant gap in the state of the art related to the analysis of gang tasks.

Contribution. To fill this gap, this paper derives two schedulability analyses for partitioned gang scheduling of parallel tasks under fixed priorities. Contrary to the state-of-the-art for partitioned gang scheduling, which follows the rigid model [9], this work adopts a more accurate representation of the internal parallelism of the tasks by means of the *bundled* task model [10]. In particular, we first present a closed-form schedulability analysis where each bundle of a task is separately analyzed, thus obtaining a set of response-time bounds for each task in the system. The closed-form analysis solves challenge **(i)** by providing a precise characterization of the interfering workload for each bundle of the analyzed task. Then, a second, more precise solution is proposed by bounding inter-task interference with an optimization problem formulated with Mixed-Integer Linear Programming (MILP). This enhanced design-time schedulability analysis better accounts for non-trivial scheduling effects that are difficult to capture with a closed-form solution, as a way to avoid to account for the same interference source multiple times, thus addressing challenge **(ii)**. Specialized partitioning heuristics for the bundled model are also proposed to find an efficient mapping of each bundle to the available cores, leveraging specific properties of the model. Finally, the results of an extensive experimental evaluation are presented, comparing the proposed approach with other approaches [9, 10], showing improvements up to 34%.

2 System Model

We consider a symmetric multicore platform equipped with M identical cores $P = \{p_1, p_2, \dots, p_M\}$. The application workload is modeled as a task set of N sporadic bundled parallel tasks $\Gamma = \{\tau_1, \tau_2, \dots, \tau_N\}$. The bundled task set can be derived from sporadic parallel directed acyclic graph (DAG) tasks, as recalled in Section 3.

Each task τ_i consists of a sequence of b_i bundles denoted by $\tau_i = (\tau_{i,1}, \tau_{i,2}, \dots, \tau_{i,b_i})$, and is characterized by a minimum inter-arrival time T_i, a relative deadline $D_i \leq T_i$ (constrained deadline), and a unique fixed priority π_i. The symbol τ_i is also used to denote the set of bundles of a task.

Given an arbitrary task τ_i, the set of tasks with higher priorities than τ_i is denoted by $hp(\tau_i)$. We assume tasks in Γ are ordered by decreasing priority.

A task τ_i generates an infinite sequence of jobs (or instances) separated by at least T_i time units. Each job of τ_i executes the b_i bundles in order. Bundle instances are referred to as bundle jobs or, simply, jobs. The priority π_i of each task τ_i is inherited by each job of the task and all the bundle instances in that job.

Each bundle $\tau_{i,j} \in \tau_i$ is characterized by two parameters $(h_{i,j}, l_{i,j})$, where $h_{i,j} \leq M$ is the number of cores needed to gang-schedule the bundle and $l_{i,j}$ is the worst-case execution time (WCET) of the bundle on each core. We define the WCET of the entire task τ_i as $L_i = \sum_{j=1}^{b_i} l_{i,j}$.

Tasks are scheduled according to a partitioned, preemptive, fixed-priority gang scheduling scheme. Therefore, each bundle needs to be statically allocated to a set of available cores. From the requirement of $h_{i,j}$ cores of each bundle, the partitioning phase decides the actual set of cores that are allocated for the bundle. To this end, we define the concept of *partitioned bundled gang assignment*. A partitioned bundled gang assignment $A_{i,j} \subseteq \{p_1, p_2, \dots, p_M\}$ of a bundle $\tau_{i,j}$ of a task τ_i is a subset of cores of size $|A_{i,j}| = h_{i,j} \leq M$, which are assigned to execute jobs of bundle $\tau_{i,j}$.

The utilization per assigned core p_k of a single bundle $\tau_{i,j}$ of a task τ_i is defined as $U^{\star}_{i,j} = l_{i,j}/T_i$, while its total utilization (over multiple cores) is defined as $U_{i,j} = (l_{i,j} \cdot h_{i,j})/T_i$. The utilization of a task τ_i is defined as $U_i = \sum_{j=1}^{b_i} (l_{i,j} \cdot h_{i,j})/T_i$.

The worst-case response time (WCRT) bound R_i of a task τ_i is an upper-bound on the longest time span from when any of its jobs is released to when it completes. Analogously, the WCRT bound $R_{i,j}$ of a bundle $\tau_{i,j}$ is an upper-bound on the longest time span from when any instance of bundle $\tau_{i,j}$ is released to when it completes. A task is said to be *schedulable* if $R_i \leq D_i$. A task set is said to be schedulable if all its tasks are schedulable.

We say that a task τ_i is *active* at time t if a job of τ_i is released at or before t and has not finished yet. A bundle $\tau_{i,j}$ is said to be *active* at time t if and only if **(1)** the task τ_i

TABLE 1: Table of the main symbols used in this paper.

Sym.	Description	Sym.	Description
p_k	k-th core	τ_i	i-th task
L_i	total WCET of τ_i	D_i	relative deadline of τ_i
T_i	min. inter-arrival time	b_i	number of bundle of τ_i
π_i	fixed priority of τ_i	$\tau_{i,j}$	j-th bundle of τ_i
$l_{i,j}$	WCET of $\tau_{i,j}$	$h_{i,j}$	# of cores of $\tau_{i,j}$
$A_{i,j}$	gang assign. of $\tau_{i,j}$	$R_{i,j}$	WCRT of $\tau_{i,j}$
$\mathcal{B}_{i,j}$	set of bundles that can interfere with $\tau_{i,j}$	$\mathcal{T}_{i,j}$	set of tasks that can interfere with $\tau_{i,j}$

is active, **(2)** the current instance of $\tau_{i,j}$ is not completed yet, and **(3)** when $j > 1$, the bundle $\tau_{i,j-1}$ has finished executing. For each job of τ_i, a corresponding job of $\tau_{i,j}$ is released at the first time instant after the release of τ_i in which all the conditions (1), (2), and (3), hold. Assuming that the task set is schedulable, for any time instant t, there is at most one active bundle for each task τ_i, since $R_i \leq D_i \leq T_i$.

To help the reader follow the adopted notation, Table 1 reports the main symbols introduced in the system model.

3 Background

This section summarizes the key pillars on which the paper builds. First, Section 3.1 reviews the dynamic self-suspending task model. Then, Section 3.2 reviews the algorithm proposed in [10] to convert both fork-join and DAG tasks to the bundled model.

3.1 The Dynamic Self-Suspending Model

In the *dynamic self-suspending* task (DSS) model, each task τ_i is characterized by a tuple $(C_i, S_i, D_i, T_i, \pi_i)$, where C_i is the cumulative WCET of τ_i, S_i is an upper bound on the self-suspension time of τ_i across all of its suspension intervals, T_i is the minimum inter-arrival time between jobs of τ_i, $D_i \leq T_i$ is the relative deadline of each job of τ_i, and π_i is the priority of τ_i. The utilization of a self-suspending task τ_i is defined as $U_i = C_i/T_i$. Tasks are assigned to a unique priority level such that task τ_i has a higher priority than task τ_j if $\tau_i \in hp(\tau_j)$, and are scheduled on a single processor under a fixed-priority scheduler.

Following [16], under this model an upper bound on the WCRT R_i of a dynamic self-suspending task τ_i is given by the minimum positive value for t such that

$$C_i + S_i + \sum_{\tau_p \in hp(\tau_i)} \left\lceil \frac{t + \sum_{j=p}^{i-1} S_j x_j + y_p (R_p - C_p)}{T_p} \right\rceil C_p \leq t, \tag{1}$$

where $y_p = (1 - x_p)$, $x_p = 1$ if $U_p(R_p - C_p) > S_p \times \sum_{\ell=1}^{p} U_\ell$, and $x_p = 0$ otherwise, provided that $R_p \leq D_p$ holds for each $\tau_p \in hp(\tau_i)$. Note that a different WCRT upper bound for the task under analysis τ_i can be obtained for every possible assignment of x_p, with $\tau_p \in hp(\tau_i)$. The computation in Eq. (1) considers the interference on τ_i generated by the set of higher-priority tasks $hp(\tau_i)$ by explicitly accounting for their maximum suspension times. This unifying analysis dominates existing techniques where the suspensions of interfering tasks are accounted for either with a release jitter term or with a blocking time term. In particular, if all values of x_p are set to 0, then this analysis is equivalent to modeling the suspensions as release jitter, while if all values are set to 1 then the analysis is equivalent to modeling the suspensions as blocking time. Other configurations of the x_p values allow to account for the suspension of each higher-priority task in different ways, according to the values of the corresponding x_p parameters. Note that, when setting all values of x_p to 0, we obtain an analysis that does not explicitly depend on the suspension times of the higher-priority tasks, where the WCRT upper bound is simply given by the minimum positive value for t such that

$$C_i + S_i + \sum_{\tau_p \in hp(\tau_i)} \left\lceil \frac{t + (R_p - C_p)}{T_p} \right\rceil C_p \leq t. \tag{2}$$

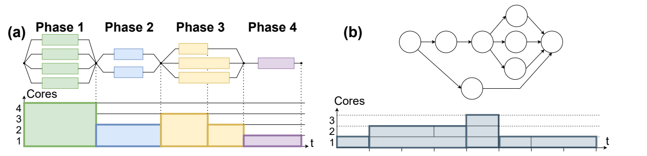

Fig. 2: Examples of transformations from the fork-join (inset (a)) and DAG (inset (b)) task models to the bundled model.

3.2 Model Transformation to Bundled Tasks

To make the paper self-consistent, we briefly describe the model transformations proposed in [10].

Fork-Join. In the fork-join model [17, 18] (Fig. 2(a)), tasks are composed of an interleaved sequence of parallel phases of computation with multiple subtasks, subject to precedence constraints. In each phase, if all subtasks have the same WCET, the phase is mapped to only one bundle with WCET $l_{i,j}$, i.e., equal to the WCET of the phase, and a required number of cores $h_{i,j}$ equal to the number of subtasks of the phase. If the subtasks in a phase have different WCETs, the phase is mapped to one bundle for each different WCET, as for phase 3 of Fig. 2. In this case, the first bundle has WCET equal to the shorter subtask and includes all the subtasks, while the following bundle includes only the subtasks with higher WCET. The WCET of this bundle is set to be equal to the total WCET minus the WCET of the previous bundles obtained by transforming the phase under analysis.

DAG. In this case, the execution of the program is simulated, with a scheduling simulator, according to any work-conserving application scheduler using M cores where each subtask executes in a non-preemptive way for its WCET, as shown in Fig. 2(b). If the parallelism degree of the DAG task is larger than the number of cores, the maximum time interval in which the number of used cores remains constant is identified in the simulation. For this interval, a bundle $\tau_{i,j}$ is created such that $h_{i,j}$ is equal to the number of cores and $l_{i,j}$ is equal to the length of that interval.

4 Closed-Form Schedulability Analysis

This section proposes a closed-form schedulability analysis for bundled gang tasks executed under partitioned fixed-priority scheduling. The proposed analysis follows the response-time analysis approach, where a bound R_i on the WCRT is first derived for all tasks $\tau_i \in \Gamma$, and then the task set is deemed schedulable if $R_i \leq D_i$ holds $\forall \tau_i \in \Gamma$.

We assume that a partitioned gang assignment $A_{i,j}$ is given for each bundle $\tau_{i,j}$ of each task $\tau_i \in \Gamma$. Techniques to suitably partition bundled tasks are discussed in Section 6. We focus on analyzing whether an arbitrary task under analysis $\tau_i \in \Gamma$ can meet its deadline, assuming that all the tasks in $\mathrm{hp}(\tau_i)$ are already deemed schedulable[1]. Therefore, the schedulability test iterates from the highest-priority bundled task to the lowest-priority bundled task in the task set Γ. To this end, we derive individual response-time bounds $R_{i,j}$ for each bundle $\tau_{i,j}$ of the task τ_i under analysis, and we deem it schedulable if $R_i = \sum_{j=1}^{b_i} R_{i,j} \leq D_i$ holds.

An individual WCRT bound $R_{i,j}$ for a bundle $\tau_{i,j}$ is obtained using a transformation to the DSS task model in two different ways, both yielding a valid WCRT upper bound for $\tau_{i,j}$. In both methods, the bundle under analysis is transformed into a DSS task $\tau'_{i,j}$ with suspension time $S'_{i,j} = 0$ and WCET $C'_{i,j} = l_{i,j}$. The difference between the two techniques is that, in the former, referred to as *bundle-level transformation*, each interfering *bundle* is transformed into a DSS task, whereas, in the second method, referred to as *task-level transformation*, each interfering *task* is transformed into a DSS task. In both methods, a WCRT upper bound is obtained for the bundle under analysis by applying the state-of-the-art reviewed in Section 3.1 to the resulting sets of DSS tasks, yielding two WCRT upper bounds $R^B_{i,j}$ and $R^T_{i,j}$ for the first and second method, respectively. The final WCRT upper bound $R_{i,j}$ is computed as the minimum of the two bounds, i.e., $R_{i,j} = \min\{R^B_{i,j}, R^T_{i,j}\}$.

Next, definitions are introduced to characterize the interference suffered by an arbitrary bundle $\tau_{i,j}$ under analysis. The set $\mathcal{B}_{i,j}$ of bundles that can interfere with $\tau_{i,j}$ is:

$$\mathcal{B}_{i,j} = \{\tau_{p,l} \in \tau_p \mid \tau_p \in \Gamma \wedge (A_{i,j} \cap A_{p,l} \neq \emptyset) \wedge \pi_p > \pi_i\}, \quad (3)$$

which is the set of high-priority bundles that share at least one core with $\tau_{i,j}$. The set of the tasks $\mathcal{T}_{i,j}$ corresponding to bundles in $\mathcal{B}_{i,j}$ and that can hence interfere with $\tau_{i,j}$ is:

$$\mathcal{T}_{i,j} = \{\tau_p \in \Gamma \mid \exists \tau_{p,l} \in \tau_p \text{ such that } \tau_{p,l} \in \mathcal{B}_{i,j}\}. \quad (4)$$

Bundle-Level Transformation for Interfering Bundles. Next, we discuss how to model each interfering bundle as a DSS task. To obtain a characterization of the interference suffered by $\tau_{i,j}$, we study the timing behavior of the bundles $\tau_{p,l} \in \mathcal{B}_{i,j}$ that can interfere with $\tau_{i,j}$ from the point of view of the bundle under analysis.

Lemma 1. Consider a bundle under analysis $\tau_{i,j}$ and an interfering bundle $\tau_{p,l} \in \mathcal{B}_{i,j}$ that is the highest priority active bundle in $\mathcal{B}_{i,j}$ at a given time t. From the point of view of $\tau_{i,j}$, the bundle $\tau_{p,l}$ exhibits a suspension behavior at time t (i.e., it is the highest priority active bundle in $\mathcal{B}_{i,j}$ but it is not executing at time t) if and only if at least one bundle $\tau_{k,q}$ in $\mathcal{B}_{p,l}$ but not in $\mathcal{B}_{i,j}$ is executed at t.

Proof. If the bundle $\tau_{p,l}$ is the highest-priority active bundle in $\mathcal{B}_{i,j}$ and it is not executing at time t, then it is interfered by a bundle $\tau_{k,q}$ in $\mathcal{B}_{p,l}$; hence τ_k has higher priority than τ_p. Clearly $\tau_{k,q} \notin \mathcal{B}_{i,j}$, otherwise $\tau_{p,l}$ would not be the highest

1. This is also a required assumption to apply the analysis by Chen et al. [16] in Eq. (1), which we ultimately use to derive the WCRT bounds after leveraging the task-set transformation proposed next.

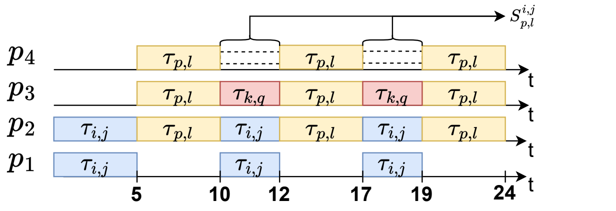

Fig. 3: Example of self-suspending behavior of the higher-priority bundle $\tau_{p,l} \in \mathcal{B}_{i,j}$ as seen from the point of view of the bundle under analysis $\tau_{i,j}$. The self-suspending behavior of $\tau_{p,l}$ is represented in the figure by the dotted boxes and is due to a bundle $\tau_{k,q}$ with higher priority than $\tau_{p,l}$ that executes on p_3, which is one of the cores needed by $\tau_{p,l}$ but not needed by $\tau_{i,j}$.

priority active bundle in $\mathcal{B}_{i,j}$ at time t, contradicting the hypothesis. Therefore, $\tau_{k,q}$ must be in $\mathcal{B}_{p,l} \setminus \mathcal{B}_{i,j}$.

Now, suppose that a bundle $\tau_{k,q} \in \mathcal{B}_{p,l} \setminus \mathcal{B}_{i,j}$ is executing at time t. By contradiction, assume that $\tau_{p,l}$ is not suspended at t. Since $\tau_{p,l}$ has the highest priority in $\mathcal{B}_{i,j}$, then it must be in execution at the same time instant t. However, $\tau_{k,q}$ is also executing at time t and is in the set $\mathcal{B}_{p,l} \setminus \mathcal{B}_{i,j}$, thus $\tau_{k,q}$ has higher priority than $\tau_{p,l}$ and $A_{p,l} \bigcap A_{k,q} \neq \emptyset$. This is not possible because it implies that both $\tau_{k,q}$ and $\tau_{p,l}$ are executing at the same time on the same cores. Thus, $\tau_{p,l}$ must be in a suspended state at time t and the lemma follows. □

Example. An example of this behavior is shown in Fig. 3. In the figure, a bundle under analysis $\tau_{i,j}$ suffers interference from a higher-priority bundle $\tau_{p,l} \in \mathcal{B}_{i,j}$. From the point of view of $\tau_{i,j}$, $\tau_{p,l}$ exhibits a suspension behavior when $\tau_{p,l}$ itself is interfered by another bundle $\tau_{k,q} \in \mathcal{B}_{p,l} \setminus \mathcal{B}_{i,j}$. Specifically, assume that **(i)** $\tau_{i,j}$, the lowest priority bundle, needs to execute on cores p_1 and p_2; **(ii)** $\tau_{p,l}$ needs to execute on cores p_2, p_3, and p_4; and **(iii)** $\tau_{k,q}$, the highest priority bundle, needs to execute on p_3. The bundle $\tau_{i,j}$ starts to execute on p_1 and p_2, then, at $t = 5$, one instance of $\tau_{p,l}$ becomes active and preempts $\tau_{i,j}$ because $\tau_{p,l}$ has higher priority and needs to execute on p_2. At $t = 10$, a job of $\tau_{k,q}$ becomes active and preempts $\tau_{p,l}$, since it needs to execute on p_3 and has higher priority. As seen in Fig. 3, from the point of view of $\tau_{i,j}$, $\tau_{p,l}$ is self-suspended at this time. This is because $\tau_{i,j}$ is not assigned to p_3; therefore, from the perspective of $\tau_{i,j}$, $\tau_{p,l}$ is ready to execute on the core p_2, which it shares with $\tau_{i,j}$, but it is not executing. This is equivalent to a situation where $\tau_{p,l}$ self-suspends between times 10 to 12. The same self-suspending behavior is also observed between times 17 and 19.

Lemma 2 derives an upper bound on the suspension time of a bundle $\tau_{p,l} \in \mathcal{B}_{i,j}$ as seen from the point of view of $\tau_{i,j}$.

Lemma 2. Consider a bundle $\tau_{i,j}$ under analysis and a higher-priority bundle $\tau_{p,l} \in \mathcal{B}_{i,j}$ belonging to the task τ_p. When analyzing $\tau_{i,j}$, the suspension time of bundle $\tau_{p,l}$, as seen from the point of view of the bundle under analysis $\tau_{i,j}$, is upper bounded by

$$S^{i,j}_{p,l} = \min\left\{R_{p,l} - l_{p,l}, \sum_{\tau_{k,q} \in \{\mathcal{B}_{p,l} \setminus \mathcal{B}_{i,j}\}} \left\lceil \frac{R_{p,l} + \hat{R}_{k,q}}{T_k} \right\rceil l_{k,q}\right\}, \quad (5)$$

where $\hat{R}_{k,q} = \min\left\{R_k - \sum_{z=q+1}^{b_k} l_{k,z}, \sum_{z=1}^{q} R_{k,z}\right\}$[2].

Proof. First, the suspension of the bundle $\tau_{p,l}$, as seen from the point of view of the bundle under analysis $\tau_{i,j}$, cannot exceed $R_{p,l} - l_{p,l}$; otherwise the definition of WCRT would be violated. Then, by Lemma 1, the maximum amount of suspension of $\tau_{p,l}$, as seen from the point of view of $\tau_{i,j}$, is equal to the interference suffered by $\tau_{p,l}$ by bundles in $\mathcal{B}_{p,l} \setminus \mathcal{B}_{i,j}$. Consider a bundle $\tau_{k,q} \in \mathcal{B}_{p,l} \setminus \mathcal{B}_{i,j}$. The WCRT of the interfering bundle $\tau_{k,q}$ with respect to the release of the task τ_k and up to the completion of $\tau_{k,q}$ can be upper bounded by the sum of the WCRTs $R_{k,z}$ of the preceding bundles in τ_k and the WCRT $R_{k,q}$ of $\tau_{k,q}$ itself, since $\tau_{k,q}$ cannot start executing before all of the preceding bundles in τ_k are completed. Another upper bound to the WCRT of $\tau_{k,q}$ is given by the WCRT R_k of τ_k minus the WCET $l_{k,z}$ of all the bundles following $\tau_{k,q}$ in τ_k, since $\tau_{k,q}$ will always terminate within that time. A tighter WCRT upper bound, denoted by $\hat{R}_{k,q}$, can be obtained by considering the minimum of the two WCRT bounds for $\tau_{k,q}$. Next, consider that the number of instances of a task τ_x with sporadic release pattern that can exist in an interval of length t is upper bounded by $\left\lceil \frac{t+R_x}{T_x} \right\rceil$ [19]. Thus, the number of instances of $\tau_{k,q}$ that can exist in a time interval of length $R_{p,l}$ is upper bounded by $\left\lceil \frac{R_{p,l}+\hat{R}_{k,q}}{T_k} \right\rceil$. Each of these instances cannot execute and cause interference on $\tau_{p,l}$ for more than $l_{k,q}$ time units. Hence the lemma follows. □

To analyze $\tau_{i,j}$, we transform the bundles in $\mathcal{B}_{i,j}$ into DSS tasks, as reported in Theorem 1.

Theorem 1. *Consider a bundle $\tau_{i,j}$ under analysis and an interfering higher-priority bundle $\tau_{p,l} \in \mathcal{B}_{i,j}$. When analyzing $\tau_{i,j}$, $\tau_{p,l}$ can be safely modeled by a DSS task $\tau_{p,l}^{sus} = (C'_{p,l}, S'_{p,l}, D_p, T_p, \pi_p)$, where $C'_{p,l} = l_{p,l}$ and $S'_{p,l} = S_{p,l}^{i,j} + \sum_{k=1}^{l-1} R_{p,k}$, with $S_{p,l}^{i,j}$ defined as in Lemma 2.*

Proof. Each job of the higher-priority bundle $\tau_{p,l}$ cannot cause more interference than $C'_{p,l}$ on $\tau_{i,j}$. By Lemma 2, each job of the bundle $\tau_{p,l}$ can be considered as self-suspended for at most $S_{p,l}^{i,j}$ units of time from the point of view of $\tau_{i,j}$. The bundle $\tau_{p,l}$ is released when the preceding bundle $\tau_{p,l-1}$ terminates its execution. Thus, the interfering workload generated by a job of $\tau_{p,l}$ on $\tau_{i,j}$ is delayed by at most $\sum_{k=1}^{l-1} R_{p,k}$ units of time since the release of the corresponding job of τ_p. This delay is modeled as an additional self-suspension interval for the job of $\tau_{p,l}$. □

From Theorem 1, the WCRT upper bound $R_{i,j}^{B}$ is obtained by analyzing the transformed task $\tau'_{i,j}$ equivalent to $\tau_{i,j}$, subject to the interference of the DSS tasks in $\mathcal{B}_{i,j}^{\text{sus}} = \bigcup_{\tau_{p,l} \in \mathcal{B}_{i,j}} \tau_{p,l}^{\text{sus}}$.

Task-Level Transformation for Interfering Tasks. Next, we discuss how to model each interfering task (instead of each bundle of each interfering task) as a DSS task.

By definition of $\mathcal{T}_{i,j}$, only tasks $\tau_p \in \mathcal{T}_{i,j}$ can interfere with the bundle under analysis $\tau_{i,j}$, therefore, in the following, we study the timing behavior of the tasks in this set from the point of view of $\tau_{i,j}$.

2. Note that if $q+1 > b_k$ then $\sum_{z=q+1}^{b_k} l_{k,z} = 0$.

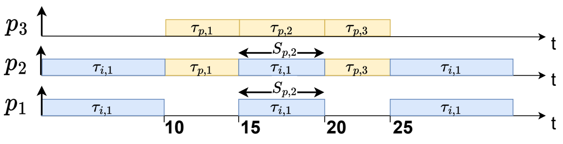

Fig. 4: Self-suspending behavior of the higher priority bundle $\tau_{p,2} \in \tau_p$, where $\tau_p \in \mathcal{T}_{i,1}$ with respect to a bundle under analysis $\tau_{i,1}$. The highlighted interval between time instants 15 and 20 shows the self-suspending behavior of $\tau_{p,2}$ that executes on core p_3 and does not need cores necessary to $\tau_{i,1}$, i.e., p_1 and p_2.

Consider an interfering task $\tau_p \in \mathcal{T}_{i,j}$ that is the highest priority active task in $\mathcal{T}_{i,j}$ at a given time instant t. From the point of view of the bundle under analysis $\tau_{i,j}$, τ_p can exhibit a behavior at time t that is analogous to a suspension, meaning that τ_p is the highest priority active task in $\mathcal{T}_{i,j}$ at t but it is not executing on any cores in $A_{i,j}$ at t. Note that, at any time instant, at most one bundle of each job of a bundled task τ_p can be active. Consequently, the job of τ_p is suspended at time t if and only if the current active bundle is suspended. This possible suspension behavior for τ_p at time t can be one of two types, depending on whether the bundle of τ_p that is active at time t belongs to $\mathcal{B}_{i,j}$ or not. In the former case, i.e., if $\tau_{p,l} \in \mathcal{B}_{i,j}$, the resulting behavior for the active bundle of τ_p is equivalent to that observed in the bundle-level transformation approach, thus its suspension behavior is characterized as in Lemma 1. In the latter case, i.e., if $\tau_{p,l} \notin \mathcal{B}_{i,j}$, a different type of suspension is observed, which is specific to the task-level transformation. Specifically, if $\tau_{p,l} \notin \mathcal{B}_{i,j}$, then $\tau_{p,l}$ is not using any of the cores necessary to the bundle under analysis $\tau_{i,j}$, i.e., $A_{p,l} \bigcap A_{i,j} = 0$. Therefore, from the point of view of $\tau_{i,j}$, $\tau_{p,l}$ exhibits a suspension behavior for all the time it is active. This latter behavior is illustrated in the following example.

Example. Consider a bundle $\tau_{i,1}$ under analysis with partitioned bundled assignment $A_{i,1} = \{p_1, p_2\}$. Furthermore, consider an interfering bundle task $\tau_p = (\tau_{p,1}, \tau_{p,2}, \tau_{p,3})$, such that $A_{p,1} = \{p_2, p_3\}$, $A_{p,2} = \{p_3\}$, and $A_{p,3} = \{p_2, p_3\}$, as shown in Fig. 4. At $t = 0$, bundle $\tau_{i,1}$ starts executing, but is then preempted by $\tau_{p,1}$ at $t = 10$. $\tau_{p,1}$ executes on p_2 and p_3 up to $t = 15$, when $\tau_{p,1}$ completes and $\tau_{p,2}$ starts executing on p_3, which is not in the the set of cores $A_{i,1}$ required by $\tau_{i,1}$. Therefore, $\tau_{p,2}$ exhibits the self-suspension behavior described above from the point of view of $\tau_{i,1}$ due to the fact that interfering tasks are composed of multiple bundles and not every bundle executes on cores in the set of cores $A_{i,1}$.

Lemma 3 provides an upper bound of the suspension of $\tau_{p,l}$ as seen from the point of view of $\tau_{i,j}$ in case $\tau_{p,l} \notin \mathcal{B}_{i,j}$.

Lemma 3. Consider a bundle $\tau_{i,j}$ under analysis and a higher-priority bundle $\tau_{p,l} \notin \mathcal{B}_{i,j}$ belonging to the bundled task $\tau_p \in \mathcal{T}_{i,j}$. When analyzing $\tau_{i,j}$, the suspension of the bundle $\tau_{p,l}$ as seen from the point of view of $\tau_{i,j}$ is upper bounded by $S_{p,l}^{i,j} = R_{p,l}$.

Proof. Suppose by contradiction that $S_{p,l}^{i,j} > R_{p,l}$. This means that the bundle is active for more than $R_{p,l}$, thus violating the definition of WCRT. □

Theorem 2 shows how to safely transform the set of interfering tasks $\tau_p \in \mathcal{T}_{i,j}$ into an equivalent DSS task.

Theorem 2. *Consider a bundle $\tau_{i,j}$ under analysis and a higher-priority bundled task $\tau_p \in \mathcal{T}_{i,j}$ that can interfere with $\tau_{i,j}$. When analyzing $\tau_{i,j}$, the bundled task τ_p can be safely modeled by a dynamic self-suspending task $\tau_p^{sus} = (C'_p, S'_p, D_p, T_p, \pi_p)$, with*

$$C'_p = \sum_{\tau_{p,l} \in \mathcal{B}_{i,j} \cap \tau_p} l_{p,l} \quad \text{and} \quad S'_p = \sum_{\tau_{p,l} \in \tau_p} S^{i,j}_{p,l}, \tag{6}$$

where, if $\tau_{p,l} \in \mathcal{B}_{i,j}$, $S^{i,j}_{p,l}$ is defined as in Lemma 2, while, if $\tau_{p,l} \notin \mathcal{B}_{i,j}$, $S^{i,j}_{p,l}$ is defined as in Lemma 3.

Proof. We first prove C'_p. By definition of WCET, each of the bundles $\tau_{p,l} \in \tau_p$ that can interfere with the bundle $\tau_{i,j}$ under analysis, i.e., such that $\tau_{p,l} \in \mathcal{B}_{i,j}$, cannot execute more than its WCET $l_{p,l}$. Consequently, the corresponding WCET of the DSS task τ_p^{sus} built from the bundles of τ_p contained in $\mathcal{B}_{i,j}$ cannot execute more than the sum of their WCETs. We then prove S'_p. The task τ_p exhibits a suspension behavior from the point of view of $\tau_{i,j}$ at time instant t if and only if its active bundle at the time instant t is exhibiting a suspension behavior. From the point of view of $\tau_{i,j}$, there are two mutually exclusive conditions for each bundle $\tau_{p,l}$ of the task τ_p: either $\tau_p \in \mathcal{B}_{i,j}$ or $\tau_p \notin \mathcal{B}_{i,j}$. In the former case, the amount of suspension exhibited by the bundle $\tau_{p,l}$ from the point of view of $\tau_{i,j}$ is upper bounded by the term $S^{i,j}_{p,l}$ defined in Lemma 2. In the latter case, the suspension of $\tau_{p,l}$ is upper bounded by the term $S^{i,j}_{p,l}$ defined as in Lemma 3. □

Therefore, the WCRT upper bound $R^T_{i,j}$ is computed using as interfering task set $\mathcal{T}^{\text{sus}}_{i,j} = \bigcup_{\tau_p \in \mathcal{T}_{i,j}} \tau_p^{\text{sus}}$, where τ_p^{sus} is the equivalent dynamic self-suspending task obtained transforming the task τ_p using Theorem 2.

Observation. The WCRT bound obtained with the proposed closed-form analysis can account for the same interference source multiple times, possibly leading to pessimistic WCRT estimations, since the bundles of the task under analysis are analyzed separately. This is similar to the so-called *pay-bursts-only-once* problem [20]. To overcome this limitation, Section 5 proposes an MILP formulation to refine the previous bound and reduce the analytical pessimism.

5 Optimization-Based Analysis

Next, we present an MILP formulation to provide a more accurate upper bound on the worst-case response time of a bundled task. In this formulation, the WCRT of the task under analysis is maximized, subject to a set of constraints that reduce the search space to obtain a tighter WCRT estimation. The MILP formulation is applied at design time to extract refined WCRT upper bounds. The closed-form WCRT bounds derived in Section 4 are leveraged to define the constraints.

The optimization-based analysis works by bounding the WCRTs using an iterative algorithm that uses, at each iteration, the optimal solution of the MILP to bound the inter-task interference that the task under analysis can experience.

Overview. Consider a task τ_i under analysis. Following standard response-time analysis principles, the WCRT of τ_i can be bounded by the smallest non-negative fixed point of the following recurrence, computed starting from $R_i = L_i$:

$$R_i \leftarrow L_i + \overline{I}_i(R_i), \tag{7}$$

provided that $\overline{I}_i(R_i)$ is a safe upper bound on the interference suffered by τ_i if R_i is its tentative WCRT bound. The iterative search of the fixed point stops in two cases: **(1)** when a fixed point is found, in which case R_i can be deemed as a safe WCRT bound; or **(2)** when $R_i > D_i$, in which case no WCRT bound can be claimed.

The MILP formulation presented next computes $\overline{I}_i(R_i)$.

In the following, we assume that WCRT bounds of higher-priority tasks are already available when analyzing a given task, which can be easily achieved by analyzing tasks in priority order, and that $R_h \leq D_h$ holds for each task τ_h with higher priority than τ_i.

The MILP formulation for a bundled task τ_i models an arbitrary, tentative schedule σ of a single job of task τ_i and a set of higher-priority tasks that can interfere with τ_i, which starts with the release of the job of τ_i and ends with its completion.

In addition to the sets $\mathcal{B}_{i,j}$ (Eq. (3)) and $\mathcal{T}_{i,j}$ (Eq. (4)), we also define the set $\mathcal{T}_i$ of tasks that can interfere with an arbitrary task τ_i under analysis as:

$$\mathcal{T}_i = \{\tau_p \mid \exists \tau_{p,l} \in \tau_p \text{ such that } \tau_{p,l} \in \mathcal{B}_i\}, \tag{8}$$

where

$$\mathcal{B}_i = \{\tau_{p,l} \mid \exists \tau_{i,j} \in \tau_i \text{ such that } \tau_{p,l} \in \mathcal{B}_{i,j}\} \tag{9}$$

is the set of bundles that can interfere with τ_i.

The arbitrary schedule modeled by the optimization problem is characterized by variables encoding the response times of the bundles of τ_i and the inter-task interference suffered by each of such bundles, defined as follows.

Definition 1. The inter-task interference $I^{i,j}_{p,l}$ imposed by a higher-priority bundle $\tau_{p,l} \in \mathcal{B}_{i,j}$ on a bundle $\tau_{i,j}$ is the maximum cumulative time in which $\tau_{i,j}$ is active but not executing because $\tau_{p,l}$ is executing on one or more cores required by $\tau_{i,j}$.

MILP Variables. The following variables are defined:

- For each bundle $\tau_{i,j}$ of τ_i, its individual response time $R_{i,j} \in [l_{i,j}, \overline{R}_{i,j}]$ in σ is encoded as a real variable, where $\overline{R}_{i,j}$ is the WCRT upper bound given by the closed-form analysis (Section 4);
- For each bundle $\tau_{i,j}$ of τ_i and for each higher-priority $\tau_{p,l} \in \mathcal{B}_{i,j}$, $I^{i,j}_{p,l} \in \mathcal{R}^{\geq 0}$ is a real variable encoding the inter-task interference due to jobs of $\tau_{p,l}$ on $\tau_{i,j}$ in σ.

MILP Formulation. For the sake of clarity, the candidate WCRT bound for τ_i is denoted by $R_i^\dagger$ in the presentation of the following constraints. The MILP has the objective of maximizing the inter-task interference that the task under analysis τ_i can suffer in the schedule σ of length $R_i^\dagger$. Thus, the objective function is defined as:

$$\textbf{maximize } \overline{I}_i(R_i^\dagger) = \sum_{j=1}^{b_i} \sum_{\tau_{p,l} \in \mathcal{B}_{i,j}} I^{i,j}_{p,l}. \tag{10}$$

The constraints presented next limit the search space of the optimizer; in fact, an unconstrained maximization of the interference would lead to an infinite value of the objective function, which is a safe WCRT bound but is also useless.

Constraints Overview. Given the tentative WCRT upper bound for the task under analysis τ_i, Constraint 1 bounds the search space for the WCRT upper bound variables based on the tentative WCRT of τ_i. Furthermore, the WCRT upper bounds produced by the closed-form analysis presented in Section 4, although potentially pessimistic, can be used to limit the search space for the interference variables in the MILP. Therefore, Constraint 2 is given to bound the interference on each bundle $\tau_{i,j}$ of the task under analysis τ_i, using the WCRT upper bounds obtained with the closed-form analysis. Note that this constraint also makes the MILP formulation safe by construction, since the WCRT upper bound obtained with the optimization cannot exceed the upper bound obtained with the closed-form analysis.

Constraints 3-6 were introduced to limit the pessimism of the analysis with respect to the closed-form approach by trying to minimize the number of times that each interfering source (bundle or task) is accounted for in the computation of the WCRT of the task under analysis and of its bundles. In particular, Constraint 3 limits the interference that each interfering task τ_p as a whole could cause on the bundle $\tau_{i,j}$ of the task τ_i under analysis, while Constraint 4 limits the interference by τ_p on the whole task τ_i under analysis. Moreover, Constraint 6 limits the interference that each bundle $\tau_{p,l}$ of the interfering task τ_p could cause on the bundle $\tau_{i,j}$ of the task τ_i under analysis, while Constraint 5 limits the interference on the whole task τ_i under analysis. The above constraints leverage the fact that only one bundle of each task can be active at any time t to limit the amount of interference suffered by the task under analysis and its bundles. Constraints 3-6 leverage the interference term from the analysis for self-suspending tasks in Eq. (2) to obtain upper bounds on the interference suffered by the bundles of the task under analysis or by the task as a whole. This approach is possible without performing an explicit transformation to self-suspending tasks because the interference term does not require an upper bound of the suspension of higher-priority tasks. Additionally, note that Constraints 3 and 6 obtain upper bounds on the interference suffered by the bundles of the task under analysis caused respectively by the interfering task as a whole and by its bundles, similar to the computation of $R^T_{i,j}$ and $R^B_{i,j}$ in the closed-form analysis.

Some of the constraints presented in the following are not reported in a linear form to simplify the presentation. [3]

Next, we start presenting the constraints. Constraint 1 enforces the definition of the individual response times $R_{i,j}$ with respect to the tentative WCRT bound $R^\dagger_i$.

Constraint 1. Given the tentative response time $R^\dagger_i$ of τ_i, the WCRT of bundles $\tau_{i,j} \in \tau_i$ in σ are constrained as follows:

$$\sum_{j=1}^{b_i} R_{i,j} \leq R^\dagger_i. \tag{11}$$

3. A ceiling operator of the form $\lceil f(x) \rceil$ can be safely linearized using an auxiliary integer variable y that is constrained as $f(x) \leq y \leq f(x) + 1 - \epsilon$, where ϵ is a small real number compatible with the integrality tolerance used by the solver. Note that linearization is only required whenever the ceiling operator is applied to a function containing an MILP variable.

Proof. The constraint follows by the definition of $R^\dagger_i$. □

Let $R^*_{i,j}$ represent the WCRT upper bound for $\tau_{i,j}$ obtained with the closed-form analysis. Constraint 2 bounds the interference on each bundle $\tau_{i,j}$ due to all the interfering bundles in $\mathcal{B}_{i,j}$ using the WCRT upper bound $R^*_{i,j}$.

Constraint 2. For each $\tau_{i,j} \in \tau_i$,

$$\sum_{\tau_{p,l} \in \mathcal{B}_{i,j}} I^{i,j}_{p,l} \leq R^*_{i,j} - l_{i,j}. \tag{12}$$

Proof. The set of higher-priority bundles that can cause interference on $\tau_{i,j}$ is given by $\mathcal{B}_{i,j}$. The interference caused by the bundles in this set is upper bounded by the cumulative interference suffered by $\tau_{i,j}$, which is itself upper bounded by $R^*_{i,j} - l_{i,j}$, given the values of the WCRT upper bounds $R^*_{i,j}$ obtained using the closed-form analysis. □

The following constraints leverage the fact that WCRT bounds are computed in priority order, so that the bounds of tasks with higher priority than τ_i are known. For each higher-priority task τ_p, let R^*_p represent a WCRT bound of τ_p. In addition, let $R^*_{p,l}$ represent a WCRT bound for each bundle of τ_p, obtained with the closed-form analysis.

Constraint 3 bounds the interference on each bundle $\tau_{i,j} \in \tau_i$ due to all bundles of τ_p.

Constraint 3. For each $\tau_{i,j} \in \tau_i$, for each $\tau_p \in hp(\tau_i)$,

$$\sum_{\tau_{p,l} \in \mathcal{B}_{i,j} \cap \tau_p} I^{i,j}_{p,l} \leq \left\lceil \frac{R_{i,j} + R^*_p - C'_p}{T_p} \right\rceil \cdot C'_p, \tag{13}$$

where C'_p is defined as in Theorem 2.

Proof. The left-hand side (LHS) of the inequality represents the interference suffered by $\tau_{i,j}$ generated by the bundles of τ_p that can interfere with $\tau_{i,j}$, i.e., those in $\mathcal{B}_{i,j} \cap \tau_p$. The right-hand side (RHS) of the inequality is constructed by noting that the interference due to all the bundles of τ_p is upper bounded by the interference that task τ_p as a whole can generate on $\tau_{i,j}$. By Theorem 2, task τ_p can be modeled as a DSS task τ^{sus}_p for the purpose of evaluating the interference of τ_p on $\tau_{i,j}$. The bundle $\tau_{i,j}$ does not perform any kind of self-suspension during its execution, and can thus be modeled as a self-suspending task $\tau^{sus}_{i,j}$ with priority π_i and suspension length 0. Hence, an upper bound on the number of jobs of τ_p that can interfere with $\tau_{i,j}$ in an interval of length $R_{i,j}$ can be obtained by considering the interference term in Eq. (2). □

Constraint 4 bounds the overall interference on task τ_i due to each interfering task τ_p.

Constraint 4. For each $\tau_p \in hp(\tau_i)$,

$$\sum_{j=1}^{b_i} \sum_{\tau_{p,l} \in \mathcal{B}_{i,j} \cap \tau_p} I^{i,j}_{p,l} \leq \left\lceil \frac{R^\dagger_i + R^*_p - C'_p}{T_p} \right\rceil \cdot C'_p, \tag{14}$$

where C'_p is defined as in Theorem 2.

Proof. Let $I^i_{p,l}$ represent the interference on τ_i caused by $\tau_{p,l}$, and I^i_p represent the interference on τ_i caused by τ_p. At any point in time t, at most one bundle of τ_i can be active; therefore, we have $I^i_{p,l} = \sum_{j=1}^{b_i} I^{i,j}_{p,l}$ and for the same reason we have that $I^i_p = \sum_{\tau_{p,l} \in \mathcal{B}_{i,j} \cap \tau_p} I^i_{p,l}$. As a result, the LHS of the constraint represents the total interference I_p^i that a task τ_p can cause on the task under analysis τ_i. Next, we show that the number of interfering jobs in Eq. (2) is valid in the context of this constraint (RHS). Consider the transformation of τ_i into a rigid gang task τ_i^R, with WCET equal to the sum of the lengths $l_{i,j}$ of the bundles $\tau_{i,j}$ in τ_i, and height given by the cardinality of the union of the cores $A_{i,j}$ assigned to each bundle $\tau_{i,j}$, i.e., $\left|\bigcup_{j=1}^{b_i} A_{i,j}\right|$. The interference caused by the higher-priority task τ_p on τ_i^R cannot be smaller than the interference I_p^i. A rigid gang task can be considered as a particular case of bundled task with only one bundle. Therefore, Theorem 2 can be applied to bound the interference caused by τ_p on the transformed rigid task τ_i^R, which is equivalent to a single bundle of a bundled task. In particular, task τ_p can be modeled as a DSS task τ_p^{sus}, whereas the task τ_i^R does not perform any self-suspension during its execution, and can thus be modeled as a DSS task with suspension length 0 and priority π_i. Therefore, an upper bound on the number of jobs of τ_p that can cause interference on τ_i^R, and thus on τ_i, in an interval of length $R_i^\dagger$ can be obtained by considering the simplified interference term in Eq. (2). □

Constraint 5 bounds the interference on task τ_i due to each interfering bundle $\tau_{p,l}$.

Constraint 5. For each $\tau_{p,l} \in \mathcal{B}_i$,

$$\sum_{j=1}^{b_i} I_{p,l}^{i,j} \leq \left\lceil \frac{R_i^\dagger + \hat{R}_{p,l} - C'_{p,l}}{T_p} \right\rceil \cdot C'_{p,l}, \tag{15}$$

where $\hat{R}_{p,l} = \min\left\{R_p^* - \sum_{q=l+1}^{b_p} l_{p,q}, \sum_{q=1}^{l} R_{p,q}^*\right\}$ and $C'_{p,l}$ is defined as in Lemma 1.

Proof. The LHS of the constraint represents the total interference that a bundle $\tau_{p,l}$ can cause on the task under analysis τ_i. Let us now focus on the RHS. First, we need to show that the number of interfering jobs in Eq. (2) is valid in the context of this constraint. This is analogous to the proof of Constraint 4, with the only difference of modeling each individual bundle $\tau_{p,l}$ as a DSS task $\tau_{p,l}^{sus}$, by leveraging Theorem 1 instead of Theorem 2 because this constraint uses a bundle-level transformation instead of a task-level transformation, which is used in Constraint 4. Then, we prove the validity of the bound on the term $\hat{R}_{p,l} - C'_{p,l}$ used as a release jitter term in the RHS to leverage Eq. (2). Term $C'_{p,l}$ directly follows from Eq. (2) after the model transformation. The response time of the interfering bundle $\tau_{p,l}$ with respect to the release of the task τ_p is upper bounded by $\hat{R}_{p,l}$ that is computed as in Lemma 2. □

Constraint 6 bounds the interference on each bundle $\tau_{i,j}$ of τ_i due to each interfering bundle $\tau_{p,l}$ in $\mathcal{B}_{i,j}$.

Constraint 6. For each $\tau_{i,j} \in \tau_i$, for each $\tau_{p,l} \in \mathcal{B}_{i,j}$,

$$I_{p,l}^{i,j} \leq \left\lceil \frac{R_{i,j} + \hat{R}_{p,l} - C'_{p,l}}{T_p} \right\rceil \cdot C'_{p,l}, \tag{16}$$

where $\hat{R}_{p,l} = \min\left\{R_p^* - \sum_{q=l+1}^{b_p} l_{p,q}, \sum_{q=1}^{l} R_{p,q}^*\right\}$ and $C'_{p,l}$ is defined as in Theorem 1.

Proof. Similarly to the proof of Constraint 5, the response time of the interfering bundle $\tau_{p,l}$ with respect to the release of the task τ_p is upper bounded by $\hat{R}_{p,l}$. By Lemma 1, the bundle $\tau_{p,l}$ can be modeled as a DSS task $\tau_{p,l}^{sus}$ for the purpose of evaluating the interference of $\tau_{p,l}$ on $\tau_{i,j}$. The bundle $\tau_{i,j}$ does not perform any self-suspension during its execution, and can thus be modeled as a DSS task $\tau_{i,j}^{sus}$ with suspension length 0 and priority π_i. Therefore, an upper bound on the number of jobs of $\tau_{p,l}$ that can cause interference on $\tau_{i,j}$ in an interval of length $R_{i,j}$ can be obtained by considering the interference term in Eq. (2). □

The fixed-point iterative algorithm described in Eq. (7) terminates in a finite number of steps. Specifically, since the inter-task interference $\overline{I}_i$ (computed using the MILP formulation) is a monotonically non-decreasing function, either **(1)** the algorithm converges to a fixed point $R_i \leq D_i$ when two consecutive iterations of the algorithm produce the same value for the interference $\overline{I}_i$, or **(2)** the tentative WCRT R_i becomes $R_i > D_i$, in which case the algorithm terminates and no valid WCRT bound is produced for τ_i.

6 Partitioned Bundle Assignment

The assignment procedure determines a suitable set of cores $A_{i,j}$ for each bundle $\tau_{i,j}$ of all tasks $\tau_i \in \Gamma$, as part of the offline design phase. Determining an optimal allocation is considered a highly intractable problem (NP-hard in the strong sense), even for the less general case of rigid gang tasks [9]. In the following, we describe a set of viable allocation techniques for bundled gang tasks.

We first provide an outline of the allocation procedure, with reference to Algorithm 1. Variants and specialized enhancements of the baseline approach for the case of bundled gang tasks are presented later in Section 6.1. In every considered technique, tasks are considered in decreasing priority order (Lines 3-22 of Algorithm 1). For each task τ_i, the bundles $\tau_{i,j}$ are each allocated to a set of cores $A_{i,j}$ according to their precedence constraints (Lines 4-22). The allocation under partitioned scheduling considers each core as a container with unitary capacity, with reference to the overall utilization of the core and in an analogy to the bin-packing problem.

Variants of Allocation Algorithms. For each allocation technique presented next, three possible variants are considered (selected with the Variant input variable), each inspired by a corresponding bin-packing heuristic algorithm (worst fit, best fit, and first fit). These common heuristics, typically also used for partitioning sequential tasks, are extended to the case of bundles, which may require more than one core. To extend the *worst fit* approach to the bundled model, for each bundle $\tau_{i,j}$, the cores are sorted in increasing order of their current utilization by considering all the previously considered bundles already allocated to a set of cores (Line 6). Then, we search for a set of $h_{i,j}$ cores where the current bundle fits (Lines 7-20). In particular, the fitting of each bundle $\tau_{i,j}$ (Line 9) is determined by considering a sliding window of length $h_{i,j}$ of consecutive cores given their ordering by increasing current utilization (Line 8). Note that, when considering a sliding window of cores, the *contiguity is not determined based on the index of the corresponding cores* but

Algorithm 1 Baseline allocation methods (utilization-based and schedulability-based allocation).

```
1:  procedure ALLOCATION($\Gamma, P$, Variant, Method)
2:      UProcs $\leftarrow \{0, \dots, 0\}_{|P|}$
3:      for all $\tau_i \in \Gamma$ in decreasing priority order do
4:          for all $\tau_{i,j} \in \tau_i$ do
5:              Allocated $\leftarrow$ FALSE
6:              ProcOrder $\leftarrow$ processor order by Variant
7:              for all $p_k \in P$ ordered by Variant do
8:                  $W \leftarrow$ ProcOrder$[k, (k + h_{i,j})\%M]$
9:                  if $\exists w \in W \mid$ UProcs$(w) + U^\star_{i,j} > 1$ then
10:                     continue
11:                 if Method = Utilization-based then
12:                     Allocated $\leftarrow$ TRUE
13:                 if Method = Schedulability-based then
14:                     $\Gamma' \leftarrow$ task set including all allocated
                            tasks and the allocated bundles of
                            $\tau_i$, assuming $A_{i,j} = W$
15:                     if SchedAnalysis$(\Gamma')$ = TRUE then
16:                         Allocated $\leftarrow$ TRUE
17:                 if Allocated = TRUE then
18:                     $A_{i,j} \leftarrow W$
19:                     for all $p_a \in A_{i,j}$ do
20:                         UProcs$(p_a) \leftarrow$ UProcs$(p_a) + U^\star_{i,j}$
                        break
21:             if Allocated = FALSE then
22:                 return Failure (task set not allocated)
23:     return Success (task set allocated)
```

Algorithm 2 Specialized allocation methods (specialized schedulability-based and WCRT-based allocation).

```
1:  procedure SPECALLOCATION($\Gamma, P$, Variant, Method)
2:      UProcs $\leftarrow \{0, \dots, 0\}_{|P|}$
3:      for all $\tau_i \in \Gamma$ in decreasing priority order do
4:          $R_{i,j} \leftarrow +\infty$
5:          for all $\tau_{i,j} \in \tau_i$ do
6:              Allocated $\leftarrow$ FALSE
7:              ProcOrder $\leftarrow$ processor order by Variant
8:              for all $p_k \in P$ ordered by Variant do
9:                  $W \leftarrow$ ProcOrder$[k, (k + h_{i,j})\%M]$
10:                 if $\exists w \in W \mid$ UProcs$(w) + U^\star_{i,j} > 1$ then
11:                     continue
12:                 $\Gamma' \leftarrow$ task set including all allocated
                        tasks and the allocated bundles of
                        $\tau_i$, assuming $A_{i,j} = W$
13:                 Sched, $R'_{i,j} \leftarrow$ SchedAnalysis$(\Gamma', i, j)$
14:                 if Sched = TRUE then
15:                     Allocated $\leftarrow$ TRUE
16:                     if Method = Schedulability-based then
17:                         $A_{i,j} \leftarrow W$
18:                         break
19:                     if Method = WCRT-based then
20:                         if $R'_{i,j} \le R_{i,j}$ then
21:                             $A_{i,j} \leftarrow W$
22:                             $R_{i,j} \leftarrow R'_{i,j}$
23:             if Allocated = FALSE then
24:                 return Failure (task set not allocated)
25:         for all $p_k \in P$ do
26:             MaxUtask $\leftarrow \max_{\tau_{i,j} \in \tau_i | p_k \in A_{i,j}} U^\star_{i,j}$
27:             UProcs$(p_k) \leftarrow$ UProcs$(p_k)$ + MaxUtask
28:     return Success (task set allocated)
```

according to their current utilization. As a result, the core ordering is recomputed multiple times and can potentially change for each bundle allocation. This has the effect of reducing the number of combinations of cores to be tested while still considering the configurations that are most likely to provide a schedulable result according to the heuristic. The window corresponding to the tentative assignment of cores for the bundle $\tau_{i,j}$ is shifted over the ordered set of cores until the first element of the window reaches the last core. In all positions where part of the sliding window exceeds the maximum core index, the window is wrapped around the set of cores as in a circular vector structure (Line 8).

Each of the allocation techniques described next is based on this overall approach and is applied by considering each of the three variants resulting from the different ordering heuristics in sequence (worst fit, which we already discussed, *best fit*, which sorts cores in decreasing order of their current utilization, and *first fit*, which simply considers cores in order of index) on the complete task set Γ, stopping at the first variant producing a valid allocation. If none of the variants returns a valid allocation for Γ (Lines 21-22), then the allocation procedure fails, although it might still be possible that a feasible allocation exists due to the non-optimal nature of the algorithm.

6.1 Allocation Methods

We now present four allocation techniques.

1) Utilization-based allocation. This is the baseline approach (described in Algorithm 1 when the Method input

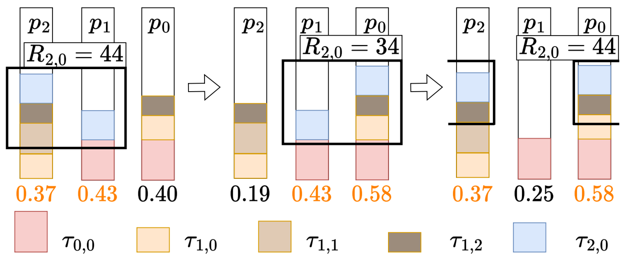

Fig. 5: Example of allocation for a bundle $\tau_{2,0}$ within the *specialized schedulability- and WCRT-based allocation* procedure (worst-fit variant).

variable is set to Utilization-based). For the allocation of cores $A_{i,j}$ for a bundle $\tau_{i,j}$, a tentative assignment is considered valid if the bundles fit in all the cores in terms of utilization in the tentative allocation window (Lines 11-12). Schedulability is only checked at the end once all bundles of all tasks have been assigned to cores.

2) Schedulability-based allocation. This technique (described in Algorithm 1 when Method is set to Schedulability-based) extends the previous approach by determining the fitting of each bundle $\tau_{i,j}$ to a set of cores not only based on the utilization of the target cores, but also by running the schedulability test for any tentative assignment of the bundle (Lines 13-16).

3) Specialized schedulability-based allocation. The above allocation methodologies are suitable for both rigid and bundled gang tasks. It is possible to define a specialized technique (described in Algorithm 2 when Method is set to Schedulability-based) for the case of bundled gang tasks by considering that, for any time instant t, at most one bundle can be active for each task $\tau_i \in \Gamma$ if the task set is schedulable. Leveraging this observation, during allocation, the contribution of each task τ_i to the utilization of core p_k is determined as the maximum utilization among the bundles of τ_i that are assigned to p_k, rather than accumulating the utilization contributed by all the bundles of τ_i (Lines 25-27). This influences the core ordering when using worst fit and best fit. As in the previous approach, schedulability is checked for each tentative assignment.

4) Specialized WCRT-based allocation. In all the previous approaches, each bundle is assigned to the first set of cores that meets the utilization-based test or the schedulability test. In contrast, this heuristic (described in Algorithm 2 when Method is set to WCRT-based) evaluates all the schedulable tentative assignments after allocating the current bundle $\tau_{i,j}$ according to the previously discussed sliding window approach and selects the one characterized by the *lowest* WCRT upper bound for $\tau_{i,j}$ (Lines 19-22).

Example. Fig. 5 depicts how the allocation for a bundle $\tau_{2,0}$ is selected within the *specialized schedulability and WCRT-based allocation*, in its worst-fit variant in a system with 3 cores. Consider that the bundled tasks $\tau_0 = \{\tau_{0,0}\}$ and $\tau_1 = \{\tau_{1,0}, \tau_{1,1}, \tau_{1,2}\}$ were already allocated, such that $A_{0,0} = \{p_0, p_1\}$, $A_{1,0} = \{p_0, p_2\}$, $A_{1,1} = \{p_2\}$, and $A_{1,2} = \{p_0, p_2\}$. The utilization for the bundles $\tau_{0,0}$, $\tau_{1,0}$, $\tau_{1,1}$, $\tau_{1,2}$, and $\tau_{2,0}$ are 0.25, 0.15, 0.19, 0.12, and 0.18, respectively. Therefore, the utilization of each of the three cores p_0, p_1, and p_2, computed by considering the maximum bundle utilization for each task in each given core, is, respectively, 0.40, 0.25, and 0.19. Thus, the order of cores considered when allocating $\tau_{2,0}$ is p_2, p_1, p_0. We then slide a window of length 2 (equal to the height of $\tau_{2,0}$) and we record the WCRT bound $R_{2,0}$ for each feasible allocation. The fitting is assessed based on the maximum utilization which is reported under the cores in Fig. 5. Finally, we select the allocation with the lowest WCRT, that is, $A_{2,0} = \{p_1, p_0\}$.

Complexity Analysis. The time complexity of the utilization-based allocation approach is $\mathcal{O}(NBM^2)$, where B is an upper bound on the number of bundles across the tasks. The complexity of the other allocation approaches is $\mathcal{O}(NBM^2 + NBM\mathcal{O}(\text{SchedAnalysis}))$, where $\mathcal{O}(\text{SchedAnalysis})$ is the complexity of the selected schedulability analysis. The complexity $\mathcal{O}(\text{SchedAnalysis})$ is $\mathcal{O}(NB(\mathcal{O}(\text{DSStransf}) + \mathcal{O}(\text{DSSanalysis})))$ when using the closed-form analysis and $\mathcal{O}(NB(\mathcal{O}(\text{DSStransf}) + \mathcal{O}(\text{DSSanalysis})) + N\mathcal{O}(\text{MILP}))$ when using the optimization-based analysis, where $\mathcal{O}(\text{MILP})$ is the complexity of solving an instance of the proposed MILP formulation, $\mathcal{O}(\text{DSStransf}) = \mathcal{O}(N^2B^3)$ is the complexity of the DSS transformation, and $\mathcal{O}(\text{DSSanalysis})$ is the complexity of the DSS analysis reported in Equation (1), which is pseudo-polynomial [16].

The complexity of the MILP formulation in terms of the number of constraints and number of variables per constraint is described in Table 2. The overall number of variables in the MILP is $\mathcal{O}(NB^2)$, whereas the number of constraints is $\mathcal{O}(NB^2)$.

TABLE 2: Complexity of the MILP formulation in terms of the number of constraints and number of variables per constraint.

Constraint	Eq.	# of constraints	# of variables per constraint
1	(11)	$\mathcal{O}(1)$	$\mathcal{O}(\lvert b_i \rvert)$
2	(12)	$\mathcal{O}(\lvert b_i \rvert)$	$\mathcal{O}(N \times B)$
3	(13)	$\mathcal{O}(\lvert b_i \rvert \times N)$	$\mathcal{O}(B)$
4	(14)	$\mathcal{O}(N)$	$\mathcal{O}(\lvert b_i \rvert \times B)$
5	(15)	$\mathcal{O}(N \times B)$	$\mathcal{O}(\lvert b_i \rvert)$
6	(16)	$\mathcal{O}(\lvert b_i \rvert \times B \times N)$	$\mathcal{O}(B)$

7 Evaluation

This section presents a selection of representative results of an extensive experimental evaluation campaign that was performed to evaluate the performance of the proposed analysis and allocation techniques. The aim of the experiments is to: **(i)** evaluate and compare the proposed allocation techniques, **(ii)** compare the performance of the closed-form and the optimization-based analyses, **(iii)** compare the proposed approach with the state-of-the-art analysis for global scheduling of bundled gang tasks by Wasly and Pellizzoni [10] and the state-of-the-art analysis for partitioned scheduling under the rigid model [9], and **(iv)** compare the runtime of the algorithms considered in the evaluation.

Experimental Setup. The task sets evaluated in the experiments were generated following the approach in [10], which we recap in the following. Each task τ_i was generated by selecting the total WCET L_i from the discrete uniform distribution $[10, 150]$, and the number of bundles b_i from the discrete uniform distribution $[2, 5]$. T_i was selected from the discrete uniform distribution $[L_i, 10 \cdot L_i]$, resulting in a task utilization U_i between $V_i/(10 \cdot L_i)$ and V_i/L_i, where $V_i = \sum_{j=1}^{b_i} l_{i,j} \cdot h_{i,j}$ is the volume of the bundled gang task τ_i. As a result, the minimum and the maximum possible values of each T_i were 10 and 1500, respectively. The relative deadline was set to $D_i = T_i$ (implicit deadlines). To obtain a task set Γ with a given system utilization value $U = \sum_{\tau_i \in \Gamma} C_i/T_i$, tasks were generated iteratively and added until the desired utilization U was reached. Finally, three types of task sets were considered to generate the height $h_{i,j}$ and the length $l_{i,j}$ of each bundle $\tau_{i,j} \in \Gamma$. For *lightly-parallel* task sets, each bundle was randomly categorized as either a short or tall bundle, with a probability of 50%. The height of short and tall bundles was selected in the discrete uniform distributions $[1, \lceil 0.3 \cdot m \rceil]$ and $[m - \lceil 0.3 \cdot m \rceil + 1, m]$, respectively. To generate $l_{i,j}$, 80% of the WCET L_i was randomly distributed among the short bundles, while the remaining 20% was distributed among tall bundles. For *heavily-parallel* task sets, generation is similar to the lightly-parallel task sets, except that 80% of the WCET is assigned to tall bundles and 20% to short ones to obtain the length of the bundles. For *mixed* task sets, the height of each bundle is selected from the discrete uniform distribution $[1, m]$, and L_i is uniformly distributed among all bundles.

Task priorities were assigned according to rate monotonic (i.e., priorities inversely proportional to periods). In some of the experiments, we considered a multicore platform with $m \in \{4, 8, 16\}$, and the system utilization U was varied from 0 to m in increments of 0.5. In other experiments, we varied the number of cores M between

4 and 16 and fixed the system utilization U to a given percentage of the total maximum system utilization (i.e., of the number of cores in the system). Then, for each system configuration point to be evaluated (system utilization or number of cores), a total of 100 task sets were generated and analyzed with the analyses under evaluation.

We evaluated the schedulability ratio, i.e., the ratio of schedulable task sets over the total number of generated task sets, for each tested system configuration. The average and maximum runtimes of the analysis techniques were collected as part of the experiments. All the experiments were executed on a computer equipped with an Intel Core i9-9900 with 8 multithreaded cores with a base frequency of 3.10 GHz, and 32 GiB of main memory.

7.1 Evaluation of Partitioned Analyses

Fig. 6(a) shows the schedulability achieved under a relevant system configuration with the two proposed analyses: the closed-form analysis (Sec. 4) and the optimization-based analysis (Sec. 5). The runtime times required by both approaches in the same configuration are reported in Fig. 6(h). When the two approaches are applied to mixed task sets running on 8 cores, a significant performance gap, up to 57%, is exhibited between the closed-form (**CF**) and the optimization-based (**MILP**) analyses (Fig. 6(a)). On the other hand, the performance gain of the optimization-based analysis comes at a significant cost in terms of analysis runtime (Fig. 6(h)). Fig. 6(h) reports the maximum (**max**) and average (**avg**) runtimes of the analyses observed when running the experiments in Fig. 6(a). When considering the average runtimes, the MILP analysis only takes thirty times more than the closed-form analysis. Overall, the runtimes are largely compatible with offline system design workflows.

7.2 Evaluation of Allocation Techniques

Fig. 6(b) compares the schedulability performance achieved with different allocation methods and with an *optimal*, exhaustive search algorithm that tests all the possible combinations of allocation for every bundle (**Exact**). Since **Exact** is characterized by an exponential algorithmic complexity, the experiment involving a comparison with the **Exact** algorithm is performed on a simplified experimental setup. In particular, we used a fixed number of tasks equal to $N = 3$, while the number of bundles b_i is selected from the reduced discrete uniform distribution $[2, 4]$. The utilization is divided among the tasks using the UUnifast algorithm [21], which generates a set of uniformly-distributed utilization values U_i for each task such that $\sum_{\tau_i \in \Gamma} U_i = U$, where U is the target system utilization. In this simplified setup, we considered a platform with $m = 4$ cores, and the system utilization U was varied from 0.4 to 2.2 in increments of 0.2. Then, for each value of U, a total of 100 task sets were generated and analyzed with the allocation techniques under evaluation, using the optimization-based analysis to assess schedulability.

In the comparison, we also considered the utilization-based allocation (**Util-based**), the schedulability-based allocation (**Sched-based**), the specialized schedulability-based allocation (**Spec-based**), and the specialized WCRT-based allocation (**RSpec-based**).

Fig. 6(b) considers the simplified experimental setup when applied to mixed task sets and reports the performance of **Exact**, while Fig. 6(c) considers the general task-set generation setup (discussed at the beginning of the section) and a higher number of cores (8), but excluding **Exact** (for scalability reasons due to the exhaustive search).

Fig. 6(j) reports the schedulability ratio achieved with the proposed allocation methods when varying the number of cores in the system between 4 and 16 and fixing the system utilization to a value equal to 60% of the number of cores (i.e., of the maximum possible system utilization).

The results in Fig. 6(b,c,j) show that **RSpec-based** performs better than the other allocation techniques. In particular, in Fig. 6(c), we highlight a significant performance gap (of up to 15% in terms of schedulability ratio) in favor of **RSpec-based**, with respect to **Spec-based** and up to 31% with respect to **Util-based**. Therefore, **RSpec-based** was selected as the allocation technique to be used in the other experiments. Moreover, we observe (in Fig. 6(b)) that **RSpec-based** achieves similar performance to the algorithm that tests all the possible combinations of allocation. Specifically, **RSpec-based** exhibits a loss of schedulability ratio of at most 3% only with respect to **Exact**, while offering drastic improvements in terms of algorithmic complexity since it does not require an exhaustive evaluation of all the possible combinations of allocation for each bundle. The results in Fig. 6(j) confirm again the RSpec-based method as the best allocation strategy, with a slight decline in performance for all methods when the number of cores increases.

7.3 Comparison with the State of the Art

The third set of experiments compares the schedulability ratio attained by the proposed MILP-based analysis for bundled gang tasks executing under partitioned scheduling with the state-of-the-art MILP-based analysis for bundled gang tasks executing under global scheduling [10] (**gl**) and the state-of-the-art closed-form analysis for rigid gang tasks executing under partitioned scheduling [9] (**rig**).

When evaluating the performance achieved by the global scheduling approach, in addition to the results of the analysis performed using the nominal values of the WCETs of each task obtained by the task set generation procedure, we report the results of the analysis on modified task sets where the WCETs of each task were inflated from 0% to 20% (with steps of 5%) of their nominal value. These results aim at evaluating the performance of global scheduling under more fair conditions with respect to partitioned scheduling approach to account for well-known phenomena such as migration costs [13], loss of cache affinity [12], and the difficulty in deriving tight WCET bounds. In particular, the experiments report the results of the global scheduling with no WCET inflation (**gl**) and with WCETs inflated by 5% (**gl+5**), 10% (**gl+10**), 15% (**gl+15**), and 20% (**gl+20**).

Fig. 6(d) compares the various approaches considering mixed-parallel task sets on 4 cores. The proposed approach (**our**) performs better than global scheduling, with a performance gain in terms of schedulability ratio of up to 13% in the case with no WCET inflation, 23% with 5% inflation, and up to 39% in the case of 20% inflation. With respect to **rig**, **our** provides an improvement of up to 18%.

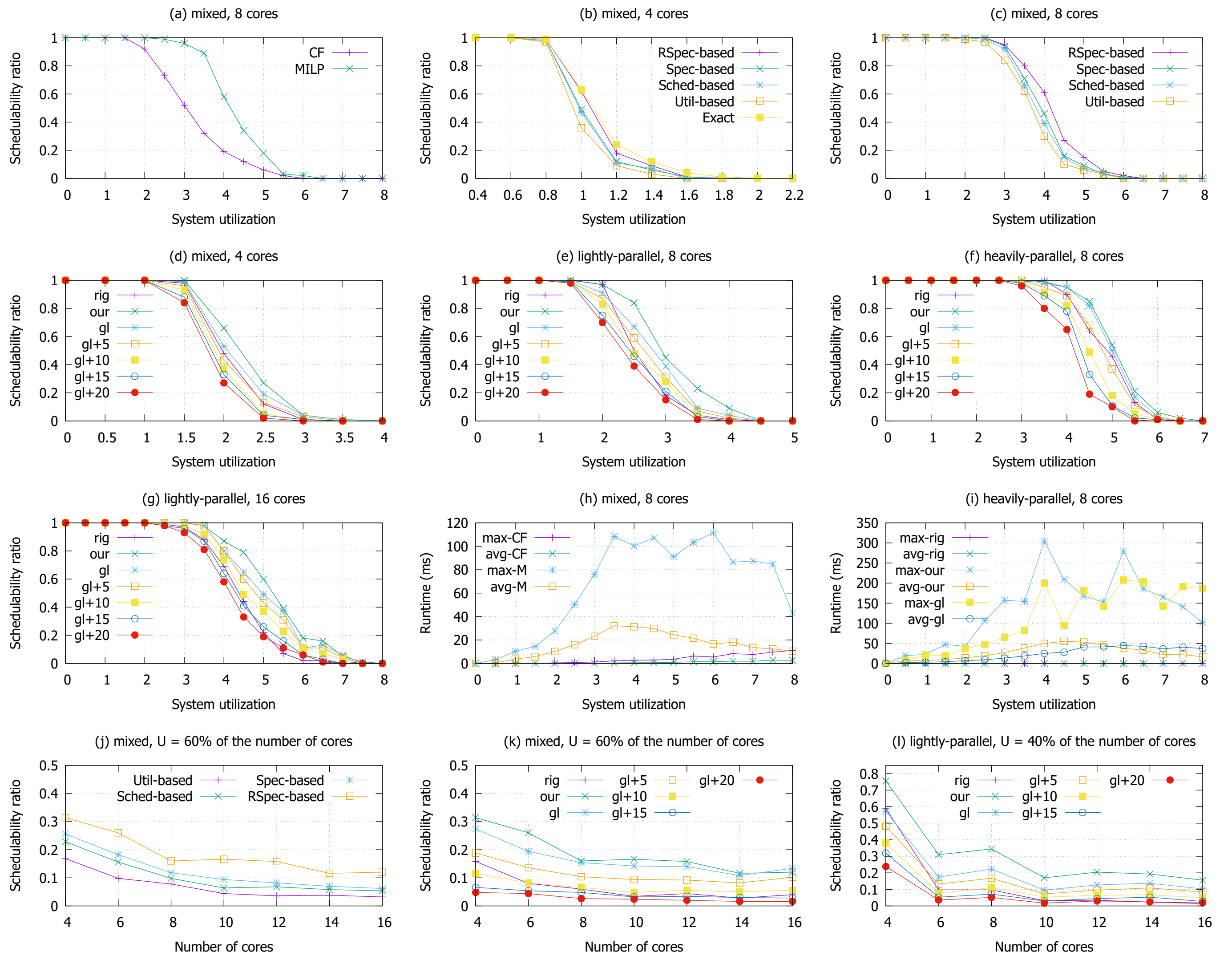

Fig. 6: Comparison in terms of schedulability ratio (a-g) or required runtime (h,i) between: the two proposed analyses (a,h), the exact and heuristic allocation techniques (b), heuristic allocation techniques (c,j), and of the proposed MILP-based analysis with global scheduling [10] and partitioned rigid scheduling [9] (d-g,i,k,l).

An even better improvement can be observed when considering lightly-parallel task sets on 8 cores (Fig. 6(e)). Here, the performance gain with respect to global scheduling reaches up to 17% without WCET inflation, 25% with 5% inflation, and 45% with 20% inflation. Furthermore, **our** exhibits a significant perform gain with respect to **rig**, up to 34%. Fig. 6(f) considers heavily-parallel tasks. In this case, **our** provides a slight performance improvement to global scheduling without WCET inflation (**gl**), up to at most 4%. However, the improvement with respect to **gl+5** and **gl+20** is significant, and equal to up to 17% and 44%, respectively. The improvement of **our** with respect to **rig** is up to 21%. Fig. 6(g) reports the results for 16 cores and lightly-parallel tasks, showing a similar trend to other experiments.

Fig. 6(k,l) presents the comparison when varying the number of cores in the system from 4 to 16, and fixing the system utilization to a fixed percentage of the number of cores. Note that Fig. 6(k) considers the same randomly generated dataset of mixed task sets utilized in Fig. 6(j). Instead, Fig. 6(l) refers to lightly parallel task sets with a percentage of utilization equal to 40% of the number of cores. The results in Fig. 6(k,l) show a similar descendent trend in performance across all the evaluated analyses which stabilizes when considering 10 or more cores, with a larger gain in performance with respect to global scheduling observed in Fig. 6(l). Overall, **our** consistently outperforms the other approaches, especially when considering inflated WCETs for global scheduling due to the additional overheads incurred by the approach.

Finally, Fig. 6(i) reports the maximum (**max**) and average (**avg**) runtimes of the analyses observed when running the experiments in Fig. 6(d). The fastest method is the analysis for partitioned rigid scheduling [9] (**rig)**, since it is based on a closed-form approach. In this case, the average and maximum runtimes are in the order of tenths of milliseconds. On the other hand, the analyses for partitioned bundled scheduling and global scheduling (**our** and **gl**) are based on an optimization approach, thus requiring higher runtimes, in the order of hundreds of milliseconds for maximum runtimes and less than 60 milliseconds for average runtimes.

Overall, all the evaluated approaches have runtimes compatible with typical offline design workflows.

8 Related Work

The timing behavior of parallel tasks has been extensively studied in the literature, according to global [22, 23], federated [24]–[26], and partitioned scheduling [27, 28]. The latter category is closer to this research. The analysis techniques for these models are either based on decomposition-based scheduling [27, 29] or on direct response-time analysis techniques [28, 30]–[32]. Similar to this paper, approaches based on response-time analysis leverage self-suspending task theory [33, 34]. However, they do not target gang scheduling, allowing subtasks to be scheduled independently.

Works on gang scheduling (and parallel tasks in general) are further classified on how cores are assigned to the parallel application. The rigid model [9, 35] assigns the number of cores a priori and does not change it during its execution; the moldable model [36] still does not change during the execution the number of cores, but determines it at runtime; the malleable model [37] allows changing the number of cores allocated to an application at runtime; and finally, the bundled model. To the best of our knowledge, only this paper and [10] considered the bundled scheduling paradigm. Concerning the analysis of gang tasks, most works considered global scheduling algorithms [8, 35], with the only recent work by Ueter et al. [9] considering partitioned scheduling, but under the rigid model, hence not accounting for the varying degree of parallelism that parallel programs exhibit during their execution.

Overall, none of the previous work considered the gang scheduling of parallel real-time tasks under partitioned scheduling and the bundled model.

9 Conclusions and Future Work

This paper presented two response-time analysis techniques for bundled gang tasks under fixed-priority partitioned scheduling: a closed-form analysis based on a transformation to a set of self-suspending tasks, and an optimization-based approach that significantly improves the bounds obtained with the closed-form analysis. Specialized allocation strategies were also proposed. Experimental results were presented to evaluate the schedulability of the proposed allocation and analysis techniques, showing performance improvements up to 34% and 17%, with respect to rigid partitioned gang scheduling [9] and global gang scheduling [10], respectively.

Future work includes deriving an analysis for bundled gang tasks under partitioned Earliest Deadline First scheduling, e.g., by leveraging existing results for self-suspending tasks under EDF [38]–[40], and devising more advanced partitioning heuristics for bundled gang tasks.

Acknowledgments

This work has been partially supported by the SERICS project funded by the Italian Ministry of University and Research (PNRR), under Grant PE00000014.

References

[1] L. Belluardo, A. Stevanato, D. Casini, G. Cicero, A. Biondi, and G. Buttazzo, "A multi-domain software architecture for safe and secure autonomous driving," in *2021 IEEE 27th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2021, pp. 73–82.

[2] S. Kato, S. Tokunaga, Y. Maruyama, S. Maeda, M. Hirabayashi, Y. Kitsukawa, A. Monrroy, T. Ando, Y. Fujii, and T. Azumi, "Autoware on board: Enabling autonomous vehicles with embedded systems," in *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, 2018, pp. 287–296.

[3] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund, "Industrial internet of things: Challenges, opportunities, and directions," *IEEE transactions on industrial informatics*, vol. 14, no. 11, pp. 4724–4734, 2018.

[4] Y. Yang and T. Azumi, "Exploring real-time executor on ros 2," in *2020 IEEE International Conference on Embedded Software and Systems (ICESS)*. IEEE, 2020, pp. 1–8.

[5] D. Casini, "A theoretical approach to determine the optimal size of a thread pool for real-time systems," in *2022 IEEE Real-Time Systems Symposium (RTSS)*, 2022, pp. 66–78.

[6] M. A. Jette, "Performance characteristics of gang scheduling in multiprogrammed environments," in *Proceedings of the 1997 ACM/IEEE conference on Supercomputing*, 1997, pp. 1–12.

[7] W. Ali and H. Yun, "Rt-gang: Real-time gang scheduling framework for safety-critical systems," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2019, pp. 143–155.

[8] Z. Dong and C. Liu, "Analysis techniques for supporting hard real-time sporadic gang task systems," *Real-Time Systems*, vol. 55, pp. 641–666, 2019.

[9] N. Ueter, M. Günzel, G. von der Brüggen, and J.-J. Chen, "Hard Real-Time Stationary GANG-Scheduling," in *33rd Euromicro Conference on Real-Time Systems (ECRTS 2021)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), B. B. Brandenburg, Ed., vol. 196. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, pp. 10:1–10:19. [Online]. Available: https://drops.dagstuhl.de/opus/volltexte/2021/13941

[10] S. Wasly and R. Pellizzoni, "Bundled scheduling of parallel real-time tasks," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2019, pp. 130–142.

[11] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM computing surveys (CSUR)*, vol. 43, no. 4, pp. 1–44, 2011.

[12] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha, "Memguard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms," in *2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2013, pp. 55–64.

[13] B. B. Brandenburg and M. Gül, "Global scheduling not required: Simple, near-optimal multiprocessor real-time scheduling with semi-partitioned reservations," in *2016 IEEE Real-Time Systems Symposium (RTSS)*, 2016, pp. 99–110.

[14] Y. Sun and M. Di Natale, "Pessimism in multicore global schedulability analysis," *Journal of Systems Architecture*, vol. 97, pp. 142–152, 2019.

[15] A. Biondi and Y. Sun, "On the ineffectiveness of 1/m-based interference bounds in the analysis of global EDF and FIFO scheduling," *Real-Time Systems*, vol. 54, no. 3, pp. 515 – 536, 2018.

[16] J.-J. Chen, G. Nelissen, and W.-H. Huang, "A unifying response time analysis framework for dynamic self-suspending tasks," in *28th Euromicro Conference on Real-Time Systems (ECRTS)*, 2016.

[17] K. Lakshmanan, S. Kato, and R. Rajkumar, "Scheduling parallel real-time tasks on multi-core processors," in *2010 31st IEEE Real-Time Systems Symposium*, Nov 2010.

[18] 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*, July 2012.

[19] B. B. Brandenburg, "Scheduling and locking in multiprocessor real-time operating systems," Ph.D. dissertation, The University of North Carolina at Chapel Hill, 2011.

[20] S. Schliecker and R. Ernst, "A recursive approach to end-to-end path latency computation in heterogeneous multiprocessor systems," in *Proceedings of the 7th international conference on Hardware/software codesign and system synthesis*, 2009, pp. 433–442.

[21] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-time systems*, vol. 30, no. 1-2, pp. 129–154, 2005.

[22] S. Baruah, "Improved multiprocessor global schedulability analysis of sporadic DAG task systems," in *2014 26th Euromicro Conference on Real-Time Systems*, July 2014.

[23] J. Fonseca, G. Nelissen, and V. Nélis, "Improved response time analysis of sporadic DAG tasks for global FP scheduling," in *Proceedings of the 25th International Conference on Real-Time Networks and Systems*, ser. RTNS '17, 2017.

[24] J. Li, J. J. Chen, K. Agrawal, C. Lu, C. Gill, and A. Saifullah, "Analysis of federated and global scheduling for parallel real-time tasks," in *2014 26th Euromicro Conference on Real-Time Systems*. IEEE, 2014, pp. 85–96.

[25] X. Jiang, N. Guan, H. Liang, Y. Tang, L. Qiao, and Y. Wang, "Virtually-federated scheduling of parallel real-time tasks," in *IEEE Real-Time Systems Symposium (RTSS)*, 2021, pp. 482–494.

[26] N. Ueter, G. Von Der Brüggen, J.-J. Chen, J. Li, and K. Agrawal, "Reservation-based federated scheduling for parallel real-time tasks," in *IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2018.

[27] X. Jiang, X. Long, N. Guan, and H. Wan, "On the decomposition-based global edf scheduling of parallel real-time tasks," in *IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2016, pp. 237–246.

[28] D. Casini, A. Biondi, G. Nelissen, and G. Buttazzo, "Partitioned fixed-priority scheduling of parallel tasks without preemptions," in *IEEE Real-Time Systems Symposium (RTSS)*, 2018, pp. 421–433.

[29] M. Qamhieh, F. Fauberteau, L. George, and S. Midonnet, "Global edf scheduling of directed acyclic graphs on multiprocessor systems," in *Proceedings of the 21st International conference on Real-Time Networks and Systems*, 2013, pp. 287–296.

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

[31] F. Aromolo, A. Biondi, G. Nelissen, and G. Buttazzo, "Event-driven delay-induced tasks: Model, analysis, and applications," in *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2021, pp. 53–65.

[32] F. Aromolo, G. Nelissen, and A. Biondi, "Replication-based scheduling of parallel real-time tasks," in *35th Euromicro Conference on Real-Time Systems, ECRTS 2023*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2023, p. 18.

[33] J.-J. Chen, G. Nelissen, W.-H. Huang, M. Yang, B. Brandenburg, K. Bletsas, C. Liu, P. Richard, F. Ridouard, N. Audsley, R. Rajkumar, D. de Niz, and G. von der Brüggen, "Many suspensions, many problems: a review of self-suspending tasks in real-time systems," *Real-Time Systems*, pp. 55(1):144–207, 2019. [Online]. Available: https://doi.org/10.1007/s11241-018-9316-9

[34] J.-J. Chen, G. von der Brüggen, W.-H. Huang, and C. Liu, "State of the art for scheduling and analyzing self-suspending sporadic real-time tasks," in *IEEE 23rd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2017.

[35] S. Lee, S. Lee, and J. Lee, "Response time analysis for real-time global gang scheduling," in *2022 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2022, pp. 92–104.

[36] G. Nelissen, J. Marcè i Igual, and M. Nasri, "Response-time analysis for non-preemptive periodic moldable gang tasks," in *34th Euromicro Conference on Real-Time Systems (ECRTS 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.

[37] J. Błażewicz, M. Machowiak, J. Węglarz, M. Y. Kovalyov, and D. Trystram, "Scheduling malleable tasks on parallel processors to minimize the makespan," *Annals of Operations Research*, vol. 129, pp. 65–80, 2004.

[38] F. Aromolo, A. Biondi, and G. Nelissen, "Response-time analysis for self-suspending tasks under edf scheduling," in *34th Euromicro Conference on Real-Time Systems (ECRTS 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.

[39] M. Günzel, G. von der Brüggen, and J.-J. Chen, "Suspension-aware earliest-deadline-first scheduling analysis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 4205–4216, 2020.

[40] M. Günzel, G. von der Brüggen, K.-H. Chen, and J.-J. Chen, "Edf-like scheduling for self-suspending real-time tasks," in *2022 IEEE Real-Time Systems Symposium (RTSS)*, 2022, pp. 172–184.

10 Biography Section

Veronica Rispo received the graduate (cum laude) degree in embedded computing systems engineering, the joint master's degree from the Scuola Superiore Sant'Anna of Pisa and the University of Pisa. She is currently working toward the PhD degree with the Real-Time Systems (ReTiS) Laboratory of the Scuola Superiore Sant'Anna of Pisa. Her current research interests include the modeling and analysis of distributed real-time applications.

Federico Aromolo received the M.Sc. degree (cum laude) in embedded computing systems from the Scuola Superiore Sant'Anna of Pisa and the University of Pisa, and the Ph.D. degree (cum laude) in computer engineering from the Scuola Superiore Sant'Anna. He is assistant professor with the Real-Time Systems (ReTiS) Laboratory of the Scuola Superiore Sant'Anna. His research interests include real-time scheduling algorithms, schedulability analysis, and real-time operating systems.

Daniel Casini received the graduate (cum laude) degree in embedded computing systems engineering, the joint master's degree from the Scuola Superiore Sant'Anna of Pisa and University of Pisa, and the PhD degree in computer engineering from the Scuola Superiore Sant'Anna of Pisa (with honors). He is assistant professor with the Real-Time Systems (ReTiS) Laboratory of the Scuola Superiore Sant'Anna of Pisa. In 2019, he has been visiting scholar with the Max Planck Institute for Software Systems (Germany). His research interests include software predictability in multi-processor and distributed systems, middleware frameworks, and schedulability analysis.

Alessandro Biondi received the graduate (cum laude) degree in computer engineering from the University of Pisa, Italy, within the excellence program, and the PhD degree in computer engineering from the Scuola Superiore Sant'Anna. In 2016, he has been visiting scholar with the Max Planck Institute for Software Systems (Germany). He is associate professor with the Real-Time Systems (ReTiS) Laboratory of the Scuola Superiore Sant'Anna. His research interests include design and implementation of realtime operating systems and hypervisors, schedulability analysis, cyberphysical systems, synchronization protocols, and component-based design for real-time multiprocessor systems. He was recipient of six best paper awards, one Outstanding Paper Award, the IEEE TCCPS Early Career Award 2023, the ACM SIGBED Early Career Award 2019, and the EDAA Dissertation Award 2017.