Cache-aware response time analysis for real-time tasks with fixed preemption points

Filip Marković
School of Innovation, Design and Engineering (IDT)
Mälardalen University
Västerås, Sweden
filip.markovic@mdh.se

Jan Carlson
IDT
Mälardalen University
Västerås, Sweden
jan.carlson@mdh.se

Radu Dobrin
IDT
Mälardalen University
Västerås, Sweden
radu.dobrin@mdh.se

***Abstract*—In real-time systems that employ preemptive scheduling and cache architecture, it is essential to account as precisely as possible for cache-related preemption delays in the schedulability analysis, as an imprecise estimation may falsely deem the system unschedulable. In the current state of the art for preemptive scheduling of tasks with fixed preemption points, the existing schedulability analysis considers overly pessimistic estimation of cache-related preemption delay, which eventually leads to overly pessimistic schedulability results. In this paper, we propose a novel response time analysis for real-time tasks with fixed preemption points, accounting for a more precise estimation of cache-related preemption delays. The evaluation shows that the proposed analysis significantly dominates the existing approach by being able to always identify more schedulable tasksets.**

***Index Terms*—Real-time Systems, Cache-related preemption delay (CRPD), Fixed-Priority scheduling, Preemptive Scheduling, Cache Memory**

I. Introduction

For many years it has been debated whether the non-preemptive real-time scheduling or the fully-preemptive real-time scheduling offers better schedulability bound and, if so, under which circumstances. Eventually, it was shown by Yao et al. [1], and Butazzo et al. [2] that Limited-Preemptive Scheduling (LPS) can provide even better schedulability results than the two since it can combine and mimic both of the scheduling paradigms.

Under the LPS paradigm, several approaches emerged, such as Deferred Preemption Scheduling (LP- DPS) proposed by Sanjoy Baruah [3], Preemption Thresholds Scheduling (LP-PTS) proposed by Wang and Saksena [4], and Fixed Preemption Points (LP-FPP) Scheduling, proposed by Alan Burns [5]. Among all of them, it has been shown by Yao et al. [1], and Butazzo et al. [2] that LP-FPP allows for the most precise estimation of the taskset schedulability since the possibly preempted points of a task are limited and known in advance, prior the system's runtime.

It is shown by Pellizoni et al. [6] that preemptions can introduce a significant preemption related delay in preemptive real-time systems, even up to 33% of the task's worst-case execution time. Also, it is shown by Bastoni et al. [7] that in real-time systems which employ a cache-based architecture, cache-related preemption delay (CRPD) is the major part of the preemption related delay. Those are a few reasons why it can be important to limit the number of preemptions, which is possible by using LP-FPP scheduling, and why such systems may benefit from a precise CRPD estimation since the preemption still may occur and a pessimistic estimation may deem a schedulable taskset being unschedulable.

The existing work on cache-related preemption estimation, integrated with the schedulability analysis, has, so far, considered only fully-preemptive real-time systems, and has been shown by Altmeyer et al. [8], [9] to be highly beneficial in determining the schedulability of real-time systems. However, concerning the LP-FPP scheduling, the existing feasibility analysis proposed by Yao et al. [10], [11] accounts for overly pessimistic estimation of preemption related delay, without seizing many of the properties of the LP-FPP task model which facilitate the CRPD estimation of a task.

In this paper, we propose a novel cache-aware response time analysis for sporadic fixed-priority real-time tasks with fixed preemption points and sequential non-preemptive regions. We first identify reloadable cache blocks, by defining the maximum number of times each memory cache block can be reloaded upon all possible preemptions, which is an improvement compared to the existing methods. Secondly, we define the CRPD bounds accounting for the individual preemptions on the non-preemptive regions, finally combining the two approaches in order to define a safely upper-bounded CRPD estimation which is integrated into the schedulability analysis. Also, we identify and propose a computation of the maximum time interval during which any of the released preempting jobs may affect the CRPD of a task. Evaluation results show that the proposed analysis outperforms the existing feasibility analysis for the LP-FPP task model in all of the evaluated cases, regardless of the variation in different task and cache parameters.

In the remainder of the paper, we first define the system model in Section II. In Section III we discuss the related work and describe the existing feasibility analysis for LP-FPP scheduling. The proposed response time analysis is motivated in Section IV and defined in Section V, which is followed by the evaluation results, shown in Section VI. The conclusions are discussed in Section VII.

II. System Model and notation

In this paper, we consider a sporadic task model, under a preemptive scheduler. In more detail, a taskset Γ consists of n tasks with preassigned fixed and disjunct priorities, where each task τ_i generates an infinite number of jobs, characterised by the following tuple of parameters: $\langle P_i, C_i, T_i, D_i \rangle$. For a task τ_i $(1 \leq i \leq n)$, its priority is denoted with P_i, its worst-case execution time (WCET) without considering for preemptions is denoted with C_i. The minimum inter-arrival time between the consecutive task jobs is T_i, and the relative deadline is denoted with D_i. The assumed deadlines are constrained, i.e. $D_i \leq T_i$. A j-th job of τ_i is denoted with $\tau_{i,j}$.

We also consider that a task consists of a sequence of l_i non-preemptive regions (NPR), separated by $l_i - 1$ preemption points (see Fig. 1). Non-preemptive regions are also called runnables in some real-time system domains, e.g. the automotive industry [12], [13], [14]. The k-th NPR of τ_i is denoted with $\delta_{i,k}$, $(1 \leq k \leq l_i)$, and its worst-case execution time, assuming no preemptions during the execution of $\delta_{i,1}, ..., \delta_{i,k-1}$, is denoted with $q_{i,k}$. Thus C_i can be expressed as $\sum_{k=1}^{l_i} q_{i,k}$, and the worst-case execution time of the first $l_i - 1$ NPRS is denoted with E_i and is equal to $E_i = \sum_{k=1}^{l_i - 1} q_{i,k}$. The k-th preemption point of τ_i is denoted with $PP_{i,k}$.

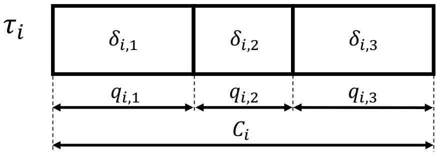

Fig. 1: Task model

In this paper, we consider single-core systems with single-level direct-mapped cache, while in Section V-D we enlist the adjustments for using the proposed method in case of LRU set-associative caches. For each NPR, we assume that the information about all the cache-sets whose blocks may be accessed throughout the execution of the NPR is derived. In this paper, we formally represent cache-sets as integers.

Formally, for each non-preemptive region $\delta_{i,k}$ we define:

◇ a set of evicting cache blocks $ECB_{i,k}$ such that a cache-set s is in $ECB_{i,k}$ if τ_i may access cache block in cache-set s throughout the execution of $\delta_{i,k}$.

For each preemption point $PP_{i,k}$ we define a notion of a useful memory block, building on the formulation originaly proposed by Lee et al. [15], and later superseded by Altmeyer et al. [16].

◇ A memory block m is useful at $PP_{i,k}$ if and only if:

a) m must be cached at program point $PP_{i,k}$, and

b) m may be reused on at least one control flow path starting from $PP_{i,k}$ without self-eviction of m before the reuse.

Then, we define a set $UCB_{i,k}$ of useful cache blocks at $PP_{i,k}$ such that a cache-set s is in $UCB_{i,k}$ if τ_i has a useful cache block in cache-set s at $PP_{i,k}$.

For each task τ_i we define a set ECB_i of evicting cache blocks such that $ECB_i = \bigcup_{k=1}^{l_i} ECB_{i,k}$.

We consider that a cache-related preemption delay is computed as the upper bound on number of cache block reloads, multiplied by a constant BRT, which is the longest time needed for a single memory block to be reloaded into cache memory, i.e. memory block reload time.

In the paper we also use the following notations:

◇ $hp(i)$: the set of tasks with priority higher than P_i.

◇ $hpe(i)$: $hp(i) \cup \tau_i$.

◇ $lp(i)$: the set of tasks with priority lower than P_i.

Regarding the mathematical notations, in some cases, we use standard integer sets to denote cache block sets, but we also use multisets. A multiset is a set modification that can consist of multiple instances of the same element, unlike a set where only one instance of an element can be present. For this reason, besides the standard set operations such as set union ($\cup$), and set intersection ($\cap$), we also use the multiset union ($\uplus$). The result of the multiset union over two multisets is a multiset that consists of the summed number of instances of each element that is present in either of the unified multisets.

III. Related Work

The majority of the cache-aware schedulability analyses are proposed in the context of fully-preemptive scheduling. Among those, the current state of the art schedulability analyses are: ECB-Union Multiset, and UCB-Union Multiset, proposed by Altmeyer et al. [8]. Multiset approaches account for a more precise estimation of the nested preemptions than the prior proposed analyses. Prior to the multiset approaches, the two most precise approaches were: UCB-Union, proposed by Tan and Mooney [17], and ECB-Union, proposed by Altmeyer et al. [18]. The union approaches considered for the first time that CRPD can be computed by intersecting the evicting cache blocks and useful cache blocks when considering task interactions, using however different principles. The very first ECB-based analyses were proposed by Tomiyama et al. [19], and Busquets-Mataix et al. [20]. In these works, it was considered that the upper bound on CRPD can be derived by computing the maximum number of ECBs. Contrary to this, the very first UCB-based analysis was proposed by Lee et al. [15], and it considered that the upper bound can be derived by computing the maximum number of UCBs. Ramaprasad and Mueller [21] accounted for the CRPD bounds by investigating the feasible preemption points and preemption patterns in fully-preemptive periodic tasks, and also in periodic tasks with a single NPR [22]. In contrast, in this work, we account for sporadic tasks, self-pushing phenomenon [23], and multiple possible NPRs, for which the above analyses are not applicable.

Considering the feasibility analysis of tasks with fixed preemptions, the seminal and the most relevant work for this paper was proposed by Yao et al. [10], [11] and this feasibility analysis is formally described at the end of this section. Prior to this work, a very important contribution was made by Bril et al. [23] who for the first time identified the scheduling anomaly known as *self-pushing phenomenon* (also described later).

The benefits of using a fixed preemption model is described by Butazzo et al. [2], among which are the increased timing predictability and the facilitation of the mutual exclusion problems. Butazzo et al. [2] and Yao et al. [1] showed that LP-FPPS can in the majority of the cases outperform many of the existing scheduling algorithms. Becker et al. [13] showed the benefits of limited-preemptiveness in automotive systems.

Other works in the area of fixed preemptions based task models included the preemption point selection algorithms, proposed by Bertogna et al. [24], Peng et al. [25], and Cavicchio et al. [26]. Cavicchio et al. [26] identified for the first time that a detailed view on CRPD is important for the tasks with fixed preemption points, especially a possibility of cache block to be reloaded. Also, in our previous work [27], [28] considering the tasks with fixed preemptions, we proposed the constraint satisfaction model for analysing CRPD at a task level. However, the existing feasibility analysis proposed by Yao et al. [10], [11] accounts for the overly pessimistic preemption delay estimation, without seizing many of the properties of the LP-FPP task model which may facilitate the CRPD estimation of a task.

In this paper, we propose a novel cache-aware schedulability analysis for fixed-priority preemptive scheduling for sporadic tasks with fixed preemption points. In the following subsection, we first describe the existing feasibility methods.

A. Overview of the feasibility analysis for tasks with non-preemptive regions

The seminal feasibility analysis for tasks with fixed preemption points was proposed by Yao et al. [10], [11], and here we briefly describe their work using Butazzo's formulations [29].

1) Self-Pushing phenomenon: A self-pushing phenomenon is a scheduling anomaly which may occur in scheduling tasks with non-preemptive regions, identified and proved by Bril et al. [23]. It states that the worst-case response time of a task does not necessarily occur in the first job after the critical instant, but it may occur in one of the succeeding jobs, during the Level-i active period. As proposed by Bril et al. [30]:

Definition 1: A Level-i pending workload at time point t is the amount of unfinished execution of jobs with a priority higher than or equal to P_i, released strictly before t.

Definition 2: A Level-i active period is an interval $[a, b]$ such that Level-i pending workload is positive for all $t \in (a, b)$ and zero in a and b.

Theorem 1: A maximum response time of task τ_i under sporadic LP-FPPS is assumed in a level-i active period that is started when τ_i has a simultaneous release with all higher priority tasks, and a subjob that produces maximum lower priority blocking starts an infinitesimal time before the simultaneous release. (proved and proposed by Bril et al. [30])

2) Feasibility analysis: For a single job $\tau_{i,j}$ of τ_i, feasibility analysis is divided into two parts. First, it is necessary to compute the latest start time $S_{i,j}$ of the last non-preemptive region. Then, the latest finish time $F_{i,j}$ of a job is computed. This is performed in two steps because a task can experience the interference from the higher priority tasks prior to the last NPR. But, when the last NPR starts to execute, it cannot be further interfered. The latest start time $S_{i,j}$ of the last NPR of $\tau_{i,j}$ is equal to the least fixed point of the following equation:

$$\begin{cases} S_{i,j}^{(0)} = b_i + C_i - q_{i,l_i} + \displaystyle\sum_{\forall \tau_h \in hp(i)} C_h \\ S_{i,j}^{(r)} = b_i + j \times C_i - q_{i,l_i} + \displaystyle\sum_{\forall \tau_h \in hp(i)} \left(\left\lfloor \frac{S_{i,j}^{(r-1)}}{T_h} \right\rfloor + 1 \right) \times C_h \end{cases}$$

where b_i denotes the maximum lower priority blocking that can be imposed on τ_i. The latest finish time $F_{i,j}$ of the j-th job is computed with the following equation $F_{i,j} = S_{i,j} + q_{i,l_i}$, i.e. by adding the latest start time of the j-th job's last NPR and the worst-case execution time q_{i,l_i} of the NPR δ_{i,l_i}.

The worst-case response time R_i is finally defined as the maximum latest finish time of all jobs released during the Level-i active period.

Furthermore, Yao et al. [11] proved that the analysis can be simplified to the first job of each task, if for each task $D_i \leq T_i$, and if the taskset is feasible under fully-preemptive scheduling.

B. Limitations of the existing approach

Considering the preemption related overhead, Yao et al. [11] proposed that all C_i values should be updated with the following equation: $C_i = C_i^{np} + (l_i - 1) \times \epsilon_i$, where C_i^{np} is the worst-case execution time without considering preemptions, and ϵ_i is the largest possible preemption cost that τ_i can experience at one of its preemption points. This can be a significant over-approximation because:

- ◇ the analysis omits the fact that CRPD of a task τ_i can be caused only upon the execution start of the first NPR $\delta_{i,1}$, until the latest start time of the last NPR δ_{i,l_i}.
- ◇ the analysis does not account for the case where a preempting task τ_h can preempt only a limited number of preemption points of a preempted task τ_i. Instead, it implicitly assumes that each preempting task can preempt τ_i at all of its preemption points.
- ◇ the analysis does not account for the case that between the two accesses of a memory block, it can be reloaded at most once due to preemption.

Therefore, in the following section, we provide motivating examples for the novel method and then we define a response time analysis with a more precise CRPD estimation.

IV. Motivating Examples

In this section, we present two motivating examples. In Fig. 2, we show a high-level scheduling perspective of a job $\tau_{i,j}$ with three preemption points ($PP_{i,1}$, $PP_{i,2}$, and $PP_{i,3}$). Also, we show three preempting jobs of τ_h such that $P_h > P_i$.

In Fig. 2, it is shown that immediately after the NPR, with the maximum blocking b_i on $\tau_{i,j}$, starts to execute, τ_i and τ_h are released. Further, we distinguish among three important time intervals. With $S_{i,j}$ we denote the latest start time of the last NPR of $\tau_{i,j}$, and with $F_{i,j}$ we denote its latest finish time. However, notice that we also denote I_i, which is the maximum duration from the start time of the first NPR of $\tau_{i,j}$ until the start time of the last NPR of $\tau_{i,j}$. Therefore, this interval also

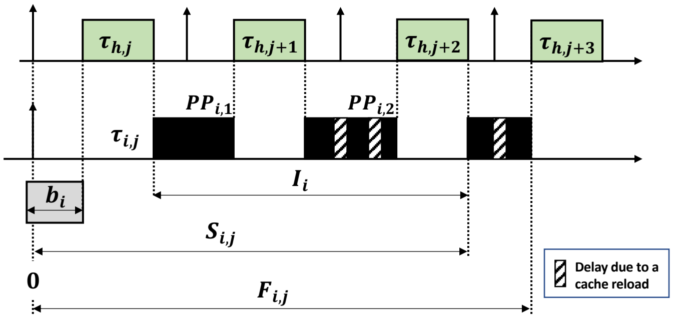

Fig. 2: Example of different time intervals used in the analysis.

represents the maximum time interval during which higher priority jobs can affect CRPD of $\tau_{i,j}$ due to preemptions. Note that the latest start time $S_{i,j}$ of the last NPR includes lower priority blocking (b_i time units long), and possibly some higher priority interference prior to the start time of $\tau_{i,j}$. During those two events, $\tau_{i,j}$ cannot be preempted nor can its CRPD be affected. Therefore, I_i can be less than $S_{i,j}$ and $F_{i,j}$ and its precise estimation is important since by an over-approximation of I_i it is possible to account for the CRPD impact from multiple preemptions which in reality cannot occur.

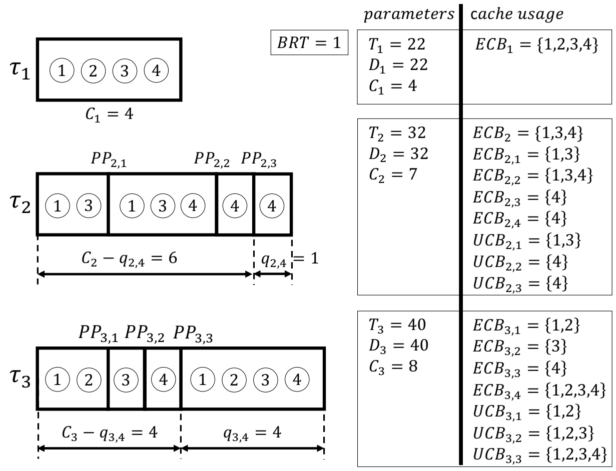

Fig. 3: Running taskset example.

In Fig. 3, we show a low-level perspective of the tasks, with associated task and cache usage parameters. The accessed cache-sets are depicted as circled integers in the NPRs where they are accessed. For example, task τ_1 accesses cache blocks in cache-sets 1,2,3, and 4. Considering task τ_3, in its first NPR it accesses cache blocks in cache sets 1 and 2, and so on. Therefore, a set of useful cache blocks at $PP_{3,1}$ is $UCB_{3,1} = \{1,2\}$, while at $PP_{3,2}$ is $UCB_{3,2} = \{1,2,3\}$, and at $PP_{3,3}$ it is $UCB_{3,3} = \{1,2,3,4\}$.

To motivate the necessity for the analysis on this level, assume that τ_1 preempts τ_3 at all of its three preemption points. Let us now focus on cache-set 1. Although it is considered that there might be a UCB residing in cache-set 1 at each of the three preempting points, this cache block can be reloaded at most once due to preemption. This reload can only occur within the last NPR of τ_3 since only in the last NPR it is re-accessed. To consider the worst-case eviction at each of those three points in isolation, as in method proposed by Yao et al.[11], can introduce over-approximation, accounting for 3 block reloads in cache set 1, opposed to at most one possible reload. Next to this, we also consider several more properties of task interactions on the cache level which are defined and formalised in the following analysis.

V. Cache-aware response time analysis

Considering the computation of the response time R_i of τ_i, the proposed analysis accounts for the following main computations (see Fig. 2):

1) b_i – Computation of the maximum lower-priority blocking on τ_i, accounting for the CRPD in the blocking.
2) I_i – Computation of the maximum time duration from the start time of $\delta_{i,1}$ until the start time of δ_{i,l_i}. I_i represents the upper bound on the interval during which higher priority jobs can affect CRPD of $\tau_{i,j}$ due to preemptions (see Fig. 2).
3) $S_{i,j}$ – Computation of the latest start time of the last NPR of $\tau_{i,j}$ accounting for the CRPD on all jobs with higher priority than τ_i which can be released within $S_{i,j}$, and accounting for the CRPD on $j-1$ jobs of τ_i. The CRPD of $\tau_{i,j}$ should only include the CRPD in first $l_i - 1$ NPRs, since in the last NPR cache block reloads occur after $S_{i,j}$, during the execution of δ_{i,l_i}. (see Fig. 2)
4) $F_{i,j}$ – Computation of the latest finish time of the j-th job of τ_i, where the worst-case execution time of the last NPR δ_{i,l_i} of τ_i is included, with the CRPD which can be caused during the execution of δ_{i,l_i}.

In the remainder of this section, we formally define the terms for computing the response time of a task, divided into three subsections: 1) *Maximum lower priority blocking*, 2) *CRPD computations*, and 3) *Latest start time, finish time, and response time*. In order to ease the understanding of the following equations, the motivating example from Fig. 3 will also serve as the running example.

A. Maximum lower priority blocking

We start by defining the longest worst-case execution time of a NPR in τ_i including the time the NPR may be prolonged due to the CRPD.

Definition 3: The longest worst-case execution time q_i^{max}, accounting for CRPD, of a non-preemptive region in τ_i is defined as follows:

$$q_i^{max} = \max_{k\in[1,l_i]} \Big(q_{i,k} + \Big| UCB_{i,k-1} \cap ECB_{i,k} \cap \cap \bigcup_{\forall \tau_h \in hp(i)} ECB_h \Big| \times BRT \Big), \text{ where } UCB_{i,0} = \emptyset \qquad (1)$$

Proposition 1: q_i^{max} is an upper bound on the longest worst-case execution time, accounting for CRPD, of a non-preemptive region in τ_i.

Proof: The CRPD of a non-preemptive region $\delta_{i,k}$ cannot be higher than BRT multiplied by the number of cache blocks such that: 1) the block can be evicted by some task with higher priority than τ_i at $PP_{i,k-1}$, and 2) due to preemption, the block can be reloaded in $\delta_{i,k}$, i.e. can be accessed in $\delta_{i,k}$, and is useful at $PP_{i,k-1}$. This is accounted in Equation 1 by the set intersections between $UCB_{i,k-1}$, $ECB_{i,k}$, and the union of all evicting cache blocks from the tasks with higher priority than τ_i. Since the equation computes the maximum sum of the WCET and the CRPD upper bound, from all NPRs in τ_i, then the proposition holds. ■

Example: Given the task τ_3 from Fig. 3, $q_3^{max} = 8$, since the NPR $\delta_{3,4}$ has the non-preemptive WCET equal to 4, and all of its 4 useful cache blocks might be reloaded during its execution due to preemption at $PP_{3,3}$.

Given the q_i^{max} term, we now define the maximum lower priority blocking that may block the start time of an instance of τ_i. Informally, it is the longest worst-case execution time of a NPR belonging to the one of the tasks with a lower priority than τ_i.

Definition 4: Maximum lower priority blocking b_i which may delay the start time of an instance of τ_i is equal to:

$$b_i = \max_{\tau_l \in lp(i)} (q_l^{max}) \tag{2}$$

Example: Given the task τ_1, its maximum lower priority blocking is equal to 8, since the NPR $\delta_{3,4}$ has the largest WCET value accounting for possible CRPD, among all the other NPRs of τ_2 and τ_3.

B. CRPD related computations

This subsection is divided into four main parts:

1) *Reloadable cache blocks* – where a modification on the term of useful cache blocks in tasks with NPRs is defined.
2) *CRPD of a single job of τ_i during the time interval t* – Where the computations for the upper bound on CRPD of a single job during the time interval t are defined.
3) *CRPD of all jobs of τ_i released during the time interval t* – Where the computations for the upper bound on CRPD for all jobs of a task that can be released during the time interval t are defined.
4) *Maximum time interval I_i* – Where the computations for the maximum time duration from the start time of $\delta_{i,1}$ until the start time of δ_{i,l_i} is defined.

1) Reloadable cache blocks:

In this part we first define an upper bound on the number of times a cache memory block m can be reloaded during the execution of a task. We exploit the fact that although a single cache block m may be useful and evicted at several consecutive preemption points, it does not necessarily mean that it can be reloaded as many times as it is useful. We prove that between the two consecutive accesses of m in τ_i, there is at most one cache block reload caused by preemptions between the accesses.

Lemma 1: If a memory block m is not accessed in a NPR $\delta_{i,k}$, then it cannot be reloaded in $\delta_{i,k}$.

Proof: A cache block reload of m in $\delta_{i,k}$ implies that it must be accessed during the execution of $\delta_{i,k}$, without being present in the cache memory, which concludes the proof. ■

Lemma 2: A memory block m can be reloaded at most once in $\delta_{i,k}$ due to preemption.

Proof: (By contradiction) Let us assume that due to preemption m can be reloaded more than once in $\delta_{i,k}$. During the execution of $\delta_{i,k}$, the first access request for the memory block m can result in a reload due to preemption and eviction of m prior the execution of $\delta_{i,k}$. However, the next reload due to preemption within the same region implies that $\delta_{i,k}$ was preempted. This is a contradiction since $\delta_{i,k}$ is non-preemptive. ■

Note that m can be self-evicted during the execution of $\delta_{i,l}$ and afterwards again reloaded in the same NPR which is accounted for by the worst-case execution time of the region. However, the source of such reload is not a preemption.

Proposition 2: If a memory block m is accessed in a NPR $\delta_{i,k}$, then due to preemptions it can be reloaded at most once from $PP_{i,k}$ until the end of the first following NPR $\delta_{i,l}$ where it is accessed.

Proof: (By contradiction) Let us assume that m is accessed in $\delta_{i,k}$, and due to preemption m can be reloaded more than once from $PP_{i,k}$ until the end of $\delta_{i,l}$. This implies two possible cases: 1) m is reloaded in a NPR between $\delta_{i,k}$ and $\delta_{i,l}$ where it is not accessed, which contradicts Lemma 1, or 2) due to preemptions, m is reloaded more than once in $\delta_{i,l}$ which contradicts Lemma 2. ■

Given the Proposition 2, in order to compute the upper bound on reloads of m during the execution of first x NPRs of τ_i, we can compute the number of first x NPRs for which m is in the ECB set of a NPR and in the UCB set of the first succeeding preemption point.

Definition 5: The upper bound $ub_{i,x}^m$ on reloads of a memory block m during execution of the first x NPRs of τ_i is equal to:

$$ub_{i,x}^m = \left| \{k \mid (1 \leq k < x \leq l_i) \land m \in ECB_{i,k} \cap UCB_{i,k}^x\} \right|, \tag{3}$$

where $UCB_{i,k}^x$ is a set of cache blocks that must be cached at $PP_{i,k}$, and may be reused until the end of the *x*-th NPR

Proposition 3: $ub_{i,x}^m$ is an upper bound on the number of cache block reloads of m during the execution of the first x NPRs of a single job of τ_i.

Proof: By Proposition 2, it holds that m can be reloaded at most once between the two succeeding accesses of m in two different NPRs, $\delta_{i,k}$ and $\delta_{i,l}$. Also, it holds that between $\delta_{i,k}$ and $\delta_{i,l}$, there is exactly one preemption point for which $m \in UCB_{i,k}^x \land m \in ECB_{i,k}$, and it is the first preemption point succeeding $\delta_{i,k}$. Then, by counting the number of non-preemptive regions $\delta_{i,k}$ $(1 \leq k < x)$ for which $m \in ECB_{i,k} \cup UCB_{i,k}^x$ is a safe upper bound on the number of reloads of m during the first x NPRS of τ_i, which is accounted by $ub_{i,x}^m$. ■

Example: Given the task τ_3 from Fig. 3, the upper bound on the number of reloads $ub^1_{3,4}$ of a memory block 1 during the execution of the entire task is $ub^1_{3,4} = 1$. Notice that memory block 1 is useful at all three preemption points and can be evicted upon preemption on any of them, however it can be reloaded at most once. Next, we define:

Definition 6: A multiset $RCB_{i,x}$, where each memory block is present the maximum number of times it can be reloaded during the execution of the first x NPRs of a single job of τ_i:

$$RCB_{i,x} = \biguplus_{\forall m \in ECB_i} \Big(\biguplus_{ub^m_{i,x}} \{m\} \Big) \tag{4}$$

Proposition 4: $RCB_{i,x}$ is a multiset of all possible reloadable cache blocks during the execution of the first x NPRs of a single job of τ_i.

Proof: Since each accessed block m (given by $m \in ECB_i$) is included $ub^m_{i,x}$ times in the multiset, then by Proposition 3 all the possible reloads for each memory block during the execution of the first x NPRs of a job of τ_i are accounted. ■

Example: Given the task τ_3 from Fig. 3, the $RCB_{3,4}$ multiset is equal to $\{1, 2, 3, 4\}$, and $RCB_{3,3} = \emptyset$. Notice that memory blocks $1, 2, 3$ and 4 are not present in $RCB_{3,3}$ because their reloads cannot occur during the execution of the first 3 regions of τ_3. For τ_2, $RCB_{2,4} = \{1, 3, 4, 4\}$.

2) CRPD of a single job of τ_i within the time interval t:

For the upper bound on CRPD caused on a job of τ_i during the time interval t, we define two complementary types of CRPD computations. In the analysis, we compute CRPD with both approaches, and later use the lower value as the final upper bound. The proposed concepts are:

◇ **RCB-based upper bound** ($\gamma^{\cup}$) – which considers that all higher priority jobs, releasable during t, can preempt a job $\tau_{i,j}$ at all preemption points. In this case, we consider that each reloadable cache block is reloaded during the execution of a job if it can be evicted by some higher priority job released during t.

◇ **Preemption-based upper bound** (γ^{+}) – which considers that a preemption, from a single higher priority job $\tau_{h,j}$ can evict and cause a reload only for the useful cache blocks of the point at $\tau_{i,j}$ where the preemption occurred. In this case, for each preempting task τ_h, whose jobs can be released j times during t interval, we account for j highest CRPD values resulting from individual preemptions of those jobs on preemption points of $\tau_{i,j}$.

To give an informal example of the two CRPD computations, let us assume that given the taskset from Fig. 3 we compute the upper bound on CRPD caused on a single job of τ_3 during a 24 units long time interval. During the interval of 24 units, at most two jobs of τ_1 can be released, and at most one job of τ_2 can be released.

For the RCB-based method it does not matter how we allocate preemptions of these jobs on preemption points, we care only about what are possibly evicting cache blocks in two jobs from τ_1 and one job from τ_2, and those are $\{1, 1, 1, 2, 2, 3, 3, 3, 4, 4, 4\}$. Also, we consider what cache blocks can be reloaded upon those evictions in single job of τ_3, and those are $RCB_{3,4} = \{1, 2, 3, 4\}$. For each cache block, present in both – the RCB- and the evicting cache blocks multisets, we account that it is evicted and reloaded, meaning that CRPD at the end considers four cache block reloads.

For the preemption-based method the allocation of preemptions on preemption points of a task is important. First, we consider two preempting jobs of τ_1 and their two possible preemptions on a job of τ_3. In this case, the maximum CRPD from two preemptions is caused on PP_3 and PP_2, causing a number of reloads equal to 4 (due to evictions and reloads of blocks in $UCB_{3,3}$), and 3 (due to evictions and reloads of blocks in $UCB_{3,2}$). For the preemption from a job of τ_2, the maximum CRPD occurs when PP_3 is preempted, since useful cache blocks 1,2, and 3 can be evicted, and CRPD accounts for 3 cache block reloads. In total, $7 + 3 = 10$ cache block reloads are accounted for, which is the less precise upper bound than the one derived with the RCB-based method.

However, in some cases, e.g., computing CRPD for a single job of τ_2, the preemption-based method provides a better result. In this case, the RCB-based method would account for 3 cache block reloads from a single job of τ_1. However, the preemption-based method computes a tighter upper bound on CRPD accounting for one preemption from τ_1 on $PP_{2,1}$. This is the case that produces the maximum CRPD, equal to two cache block reloads of memory blocks 1, and 3. For a single preemption at any other point, the CRPD is equal to one.

It is important to mention that in this paper, for a single job of any task, we are interested in the CRPD that can be caused:

◇ during the execution of the first $l_i - 1$ NPRs of a job.

◇ during the execution of all NPRs of a job.

This is important because when considering the time intervals (see Fig.2) for the job $\tau_{i,j}$ itself, we should only consider CRPD caused during the first $l_i - 1$ of its NPRs. This is because in I_i and $S_{i,j}$ intervals, we do not consider the CRPD of the last NPR of the job. But, when considering how this job affects other jobs, we must consider CRPD in all of its NPRs.

For these reasons, many equations in this paper consider the CRPD caused during the execution of the first x NPRs of a job, where x is the provided parameter. We distinguish among:

◇ $\gamma^{\cup}_x$: for RCB-based CRPD computations, and

◇ γ^{+}_x: for preemption-based CRPD computations.

Based on whether or not the last NPR should be included, as discussed above, x can be assigned the value $l_i - 1$ or l_i, respectively.

We first define the upper bound on CRPD for a single job within a time interval t using RCB-based computation:

Definition 7: The upper bound $\gamma^{\cup}_{i,x}(t)$ on the CRPD caused on the first x NPRs of a single job of τ_i by all preempting jobs which can be released within time a interval t is defined by the following equation:

$$\gamma^{\cup}_{i,x}(t) = \Big| RCB_{i,x} \cap \biguplus_{\forall \tau_h \in hp(i)} \Big(\biguplus_{\lceil t/T_h \rceil} ECB_h \Big) \Big| \times BRT \tag{5}$$

The equation accounts for the maximum number $\lceil t/T_h \rceil$ of releases of jobs of each higher priority task τ_h during t, and includes all of their evicting cache blocks as many times as they can be released.

Proposition 5: $\gamma^{\cup}_{i,x}(t)$ is an upper bound on the CRPD caused on the first x NPRs of a single job of τ_i during the interval t.

Proof: By Proposition 4, all possibly reloadable cache blocks in a single job of τ_i during the execution of the first x NPRs are given by $RCB_{i,x}$. By the multiset union, Equation 5 accounts for the maximum number of possibly evicting cache blocks of all jobs with higher priority than τ_i that can be released during t. Since by the multiset intersection Equation 5 accounts that every reloadable cache block from $RCB_{i,x}$ indeed results in a reload if there is at least one evicting block from all possibly preempting jobs released during t, then $\gamma^{\cup}_{i,x}(t)$ is an upper bound on CRPD caused on the first x NPRs of a single job of τ_i during t. ■

Example: Given all the NPRs of a task τ_3 (see Fig. 3) the upper bound on their CRPD during the 24 units long time interval is equal to:

$$\begin{aligned}\gamma^{\cup}_{3,4}(24) &= \Big|\{1,2,3,4\} \cap \big(\biguplus_{\lceil \frac{24}{22} \rceil} \{1,2,3,4\} \uplus \biguplus_{\lceil \frac{24}{50} \rceil} \{1,2,3\}\big)\Big| \\ &= \Big|\{1,2,3,4\} \cap \big(\{1,1,1,2,2,2,3,3,3,4,4\}\big)\Big| = 4\end{aligned}$$

Before defining the preemption-based CRPD upper bound, we first define a multiset which consists of all possible CRPD values that jobs of τ_h can cause on the first x NPRs of τ_i.

Definition 8: A multiset $CRPD^h_{i,x}$ of all upper bounded CRPDs that can be caused by a job of τ_h on each of the first x NPRs of a single job of τ_i is equal to:

$$CRPD^h_{i,x} = \biguplus_{k=1}^{x} \big\{ \big| ECB_h \cap UCB_{i,k-1} \big| \times BRT \big\} \qquad \text{where } UCB_{i,0} = \emptyset \tag{6}$$

Proposition 6: $CRPD^h_{i,x}$ is a multiset of upper bounds of the CRPD that can be caused by a job of τ_h on the first x NPRs of a single job of τ_i.

Proof: A CRPD upper bound on a single NPR $\delta_{i,k}$ is computed by accounting that all useful cache blocks at the preemption point directly preceding $\delta_{i,k}$ can be evicted by the evicting cache blocks of τ_h. Thus, CRPD on $\delta_{i,k}$ is computed as the intersection between the ECB_h and $UCB_{i,k-1}$ sets. Since Equation 6 combines the upper bounds from the x first NPRs in a multiset, the proposition holds. ■

Example: Given the taskset from Fig. 3, a multiset $CRPD^1_{3,4}$ of CRPDs from a job of τ_1 caused on first 4 NPRs of τ_3 is:

$$CRPD^1_{3,4} = \{|\emptyset|, |\{1,2\}|, |\{1,2,3\}|, |\{1,2,3,4\}|\} = \{0,2,3,4\}$$

Considering the cumulative CRPD caused by preemptions from jobs of τ_h during a time interval t we define the term:

Definition 9: The upper bound $\gamma^{h}_{i,x}(t)$ on CRPD caused by preemptions from jobs of τ_h on the first x NPRs of a single job of τ_i, within the time interval t, is defined in the following equation:

$$\gamma^h_{i,x}(t) = \begin{cases} \sum CRPD^h_{i,x} & , x \leq \lceil t/T_h \rceil \\ \max\big(\{ \sum c \,|\, c \subseteq CRPD^h_{i,x} \wedge \\ \qquad |c| = \lceil t/T_h \rceil \}\big) & \text{, otherwise} \end{cases} \tag{7}$$

Equation 7 computes the maximum cumulative CRPD from a maximum number of preemptions of jobs of τ_h on the first x NPRs of an instance of τ_i. Equation 7 considers two cases:

Case 1: ($x \leq \lceil \frac{t}{T_h} \rceil$), the number of preempting jobs of τ_h which can cause CRPD on first x regions of a single job of τ_i is greater than or equal to the number of preemption points that can be preempted.

Lemma 3: $\gamma^h_{i,x}(t)$ is an upper bound on CRPD caused by preemptions from jobs of τ_h on the first x NPRs of a single job of τ_i, within the time interval t, in case $x \leq \lceil \frac{t}{T_h} \rceil$.

Proof: When $x \leq \lceil \frac{t}{T_h} \rceil$, Equation 7 sums all the CRPD values in the multiset $CRPD^h_{i,x}$, and since by Proposition 6, $CRPD^h_{i,x}$ is a multiset of upper bounds of the CRPD that can be caused by a job of τ_h on the first x NPRs of a single job of τ_i, then the lemma holds. ■

Example for $x \leq \lceil \frac{t}{T_h} \rceil$: Given the taskset from Fig. 3, the upper bound on CRPD caused by preemptions from jobs of τ_1 on all NPRs of a single job of τ_3 during 100 time units is equal to $\gamma^1_{3,4}(100) = 9$ although jobs of τ_1 can be released 5 times during 100 time units, given by $\lceil 100/T_1 \rceil = \lceil 100/22 \rceil$. This is the case because only three preemption points can be preempted, and each of those preemptions individually cause maximum CRPD of 2,3, and 4 cache block reloads respectively.

Case 2: ($\lceil \frac{t}{T_h} \rceil < x$), the number of jobs which can cause CRPD on the first x regions of a single instance of τ_i is less than the number of preemption points of τ_i.

Lemma 4: $\gamma^h_{i,x}(t)$ is an upper bound on CRPD caused by preemptions from jobs of τ_h on the first x NPRs of a single job of τ_i, within the time interval t, in case $x > \lceil \frac{t}{T_h} \rceil$.

Proof: By Proposition 6, $CRPD^h_{i,x}$ is a multiset of upper bounds of the CRPD that can be caused by a job of τ_h on the first x NPRs of a single job of τ_i. Since $\lceil \frac{t}{T_h} \rceil$ is the maximum number of preemptions from the jobs of τ_h during t, then only $\lceil \frac{t}{T_h} \rceil$ number of NPRs can experience CRPD caused by jobs of τ_h. Since Equation 7 sums the $\lceil \frac{t}{T_h} \rceil$ highest values from $CRPD^h_{i,x}$ multiset, then the lemma holds. ■

Example for $\lceil \frac{t}{T_h} \rceil < x$: Given the taskset from Fig. 3, the upper bound on CRPD caused by preemptions from jobs of τ_1 on all NPRs of a single job of τ_3 during 20 time units is equal to: $\gamma^1_{3,4}(20) = 4$. This is the case because only one job of τ_1 can be released during 20 time units, given by $\lceil \frac{20}{22} \rceil$, meaning that only one preemption point can be preempted. Therefore, Equation 7 picks the single highest value in $CRPD^1_{3,4}$ which is 4 (representing the case when $PP_{3,3}$ is preempted).

Proposition 7: $\gamma^h_{i,x}(t)$ is an upper bound on CRPD caused by preemptions from jobs of τ_h on the first x NPRs of a single job of τ_i, within the time interval t.

Proof: Follows directly from Lemma 3, and 4. ■

Now, we consider that all higher priority jobs may contribute to the CRPD of a single job of τ_i during the time interval t.

Definition 10: The upper bound $\gamma^{+}_{i,x}(t)$ on CRPD caused by preemptions of all higher priority jobs during the time interval t on the first x NPRs of a single job of τ_i is defined in the following equation:

$$\gamma^{+}_{i,x}(t) = \sum_{\forall \tau_h \in hp(\tau_i)} \gamma^{h}_{i,x}(t) \tag{8}$$

Proposition 8: $\gamma^{+}_{i,x}(t)$ is an upper bound on CRPD caused by all higher priority jobs which can be released within time interval t on the first x NPRs of a single job of τ_i.

Proof: By Proposition 7, for a single higher priority task τ_h, $\gamma^{h}_{i,x}(t)$ is an upper bound on CRPD caused by all jobs of τ_h on the first x NPRs of a single job of τ_i within t. Since Equation 8 sums $\gamma^{h}_{i,x}(t)$ for each task with higher priority than τ_i, the proposition holds. ■

Example: Given the taskset from Fig. 3: $\gamma^{+}_{3,4}(24) = \gamma^{1}_{3,4}(24) + \gamma^{2}_{3,4}(24) = 7 + 3 = 10$. This is the case because Equation 8 accounts for two preemptions from τ_1 resulting in adding the two highest CRPD values (occurring at points $PP_{3,3}$: four reloads, and $PP_{3,2}$: three reloads), and one preemption from τ_2 resulting in a single highest CRPD value (occurring at $PP_{3,3}$: three reloads).

Finally, we define the CRPD upper-bound using both – the RCB-based and the preemption-based computations.

Definition 11: Combining the two proposed methods, a safe upper bound $\gamma_{i,x}(t)$ on CRPD caused on the first x NPRs of a job of τ_i during the time interval t is derived with the following equation:

$$\gamma_{i,x}(t) = \min\left(\gamma^{+}_{i,x}(t)\ ,\ \gamma^{\cup}_{i,x}(t)\right) \tag{9}$$

Proposition 9: $\gamma_{i,x}(t)$ is an upper bound on CRPD caused on the first x NPRs of a job of τ_i during the time interval t.

Proof: $\gamma_{i,x}(t)$ is an upper bound because it takes the least of two upper bounds, given by Propositions 5 and 8. ■

Example: Given the taskset from Fig. 3, the final safe upper bound is $\gamma_{3,4}(24) = \min(\gamma^{\cup}_{3,4}(24), \gamma^{+}_{3,4}(24)) = \min(4, 10) = 4$, meaning that the tighter upper bound is given by the RCB-based method. However, in case of τ_2, the CRPD on a single job within 100 time units is computed as follows: $\gamma_{2,4}(10) = \min(\gamma^{\cup}_{2,4}(10), \gamma^{+}_{2,4}(10)) = \min(3, 2) = 2$, meaning that the tighter upper bound is given by the preemption-based method.

3) CRPD on all jobs of τ_i released within time interval t:

Considering that I_i is the maximum time interval during which jobs with higher priority than τ_i may affect the CRPD of a single job of τ_i (see Fig. 2), then the CRPD upper bound of all jobs of τ_i which can be released during the time interval t, is defined as follows:

Definition 12: The upper bound $\gamma_i(t)$ on the CRPD on all the jobs of τ_i that can be released during the time interval t is defined with the following equation:

$$\gamma_i(t) = \left\lceil \frac{t}{T_i} \right\rceil \times \gamma_{i,l_i}(I_i) \tag{10}$$

Proposition 10: Assuming that I_i is an upper bound on the time duration from the start of the first NPR of τ_i until the start of the last NPR of τ_i, then $\gamma_i(t)$ is a CRPD upper bound on all the jobs of τ_i which can be released during t.

Proof: By Proposition 9 and the assumption in Proposition 10, Equation 10 derives the CRPD upper bound $\gamma_{i,l_i}(I_i)$ caused during the I_i interval on all NPRs of a single job of τ_i. Since for each job of τ_i, that can be released within t, its CRPD upper bound $\gamma_i(t)$ is multiplied by the maximum number of releases, then $\gamma_i(t)$ is a CRPD upper bound on all the jobs of τ_i which can be released during t. ■

Computation of I_i is defined in Equation 11.

4) Maximum time interval I_i:

Next, we define the computation of the I_i interval. In Equation 11, we consider the critical instant, i.e. that just after the start of execution of $\delta_{i,1}$ all the higher priority jobs may be released. It also considers the execution of a job of τ_i until the start of its last NPR, accounting for the possible CRPD during the execution of the first $l_i - 1$ NPRs. In each new iteration, the Equation accounts that the computed interval can be enlarged with the additional CRPD by the newly released higher priority jobs with their respective CRPDs:

Definition 13: The upper bound I_i of the time duration from the start time of $\delta_{i,1}$ until the start time of δ_{i,l_i} is defined as the least fixed point of the following recursive equation:

$$\begin{cases} I_i^{(0)} = & E_i \\ I_i^{(r)} = & E_i + \gamma_{i,l_i-1}(I_i^{(r-1)}) + \\ & \displaystyle\sum_{\forall \tau_h \in hp(i)} \left(\left(\left\lfloor \frac{I_i^{(r-1)}}{T_h} \right\rfloor + 1 \right) \times C_h +\ \gamma_h(I_i^{(r-1)}) \right) \end{cases} \tag{11}$$

Proposition 11: I_i is the upper bound of the time duration from the start time of the first NPR of τ_i until the start time of the last NPR of τ_i for any job of τ_i.

Proof: By induction over the tasks in Γ, in a decreasing priority order.

Base case: $I_1 = E_1$, because $hp(\tau_1) = \emptyset$ and $\gamma_{i,l_i-1}(t) = 0$ for any t since τ_1 cannot experience CRPD. E_1 is the upper bound on the time duration from the start time of the first NPR until the start time of the last NPR of τ_1 since it is the worst-case execution time between those two points.

Inductive hypothesis: Assume that for all τ_h in $hp(i)$, I_h is an upper bound on the time duration from the start of the first NPR, until the start of the last NPR of τ_h.

Inductive step: We show that Equation 11 computes the safe upper bound I_i. Consider the least fixed point of Equation 11, for which $I_i = I_i^r = I_i^{r-1}$. At this point, the equation accounts for the following upper bounds and worst-case execution times:

◇ E_i, which is the worst-case execution time, assumed by the system model, of the first $l_i - 1$ NPRs.

◇ $\gamma_{i,l_i-1}(I_i)$, which is proved by Proposition 9 to be the CRPD upper bound of a single job of τ_i in the first $l_i - 1$ NPRs during the time interval I_i.

◇ $\sum_{\forall \tau_h \in hp(i)} (\lfloor \frac{I_i}{T_h} \rfloor + 1) \times C_h$ – Worst-case interference caused by the execution of jobs of higher priority tasks. The interference is accounted as the sum of individual WCET values of each job with higher priority than τ_i, that can be released during I_i. The maximum number of jobs, of a single higher priority task τ_h, which can be released during I_i is accounted by term $\lfloor \frac{I_i}{T_h} \rfloor + 1$.

◇ $\sum_{\forall \tau_h \in hp(i)} \gamma_h(I_i)$ – CRPD upper bound on all jobs of the higher priority tasks that can be released during t. By Proposition 10 and the inductive hypothesis, $\gamma_h(I_i)$ is an upper bound on the CRPD of all jobs of τ_h which can be released within I_i. Then, the sum of CRPD bounds $\gamma_h(I_i)$ for each higher priority task than τ_i is a safe upper bound on the cumulative CRPD in all jobs with higher priority than τ_i, released during I_i.

Since we proved for all the factors, which can prolong the time duration from the start time of the first NPR until the finish time of the last NPR of τ_i, that they are accounted as the respective upper bounds in Equation 11, then their sum results in an upper bound, which concludes the proof. ■

Note that Equation 11 uses Equation 10, and vice versa. However, Equation 11 is always computable, because in order to compute I_i, Equation 11 computes $\gamma_h(t)$ term only for higher priority tasks than τ_i, meaning that it computes $\gamma_h(t)$ using only the values I_1 to I_{i-1}. Since $\gamma_1(t)$ always evaluates to zero for any value of t, because τ_1 cannot experience CRPD, this implies that I_2 can be computed using $\gamma_1(t) = 0$, which implies that I_3 can be computed using the previously computed $I_1 = 0$, and I_2 values, and so on until I_i.

C. Latest start time, finish time, and response time

In this subsection, we define the equations for the latest start time $S_{i,j}$ of the last NPR a job $\tau_{i,j}$, latest finish time $F_{i,j}$ of $\tau_{i,j}$, and the response time R_i of a task τ_i. All of those equations include the computation of the CRPD values for each job.

We first define the latest start time $S_{i,j}$ computation, for the last NPR of the j-th job of τ_i after the critical instant.

Definition 14: The latest start time $S_{i,j}$ of the last NPR of the j-th instance of τ_i is equal to the least fixed point of the following recursion:

$$\left\{\begin{array}{rl} S_{i,j}^{(0)} = & b_i + E_i \\ S_{i,j}^{(r)} = & b_i + (j-1) \times C_i + \gamma_i((j-1) \times T_i) \\ & + \ E_i \ + \gamma_{i,l_i-1}(I_i) + \\ & + \displaystyle\sum_{\forall \tau_h \in hp(i)} \left(\left(\left\lfloor \frac{S_{i,j}^{(r-1)}}{T_h} \right\rfloor + 1 \right) \times C_h + \gamma_h(S_{i,j}^{(r-1)}) \right) \end{array}\right. \tag{12}$$

In Equation 12, we consider the critical instant from Theorem 1, proved by Bril et al. [30].

Proposition 12: Assuming that the Level-i pending workload is greater than zero at each time point from the critical instant until $j \times T_i$, then starting from the critical instant, $S_{i,j}$ is the upper bound on the latest start time of the last NPR of $\tau_{i,j}$.

Proof: Based on the proposition assumption on the Level-i pending workload, the start time of $\tau_{i,j}$ is affected by all preceding $j-1$ jobs. Consider the least fixed point of Equation 12, for which $S_{i,j} = S^{r}_{i,j} = S^{r-1}_{i,j}$. At this point, the equation accounts for the following upper bounds and worst-case execution times:

◇ b_i – By Proposition 1 and Definition 4, b_i is an upper bound on lower priority blocking on τ_i.

◇ $(j-1) \times C_i$ – WCETs of the first $j-1$ jobs of τ_i, which execute prior to $\tau_{i,j}$.

◇ $\gamma_i((j-1) \times T_i)$ – By Proposition 10, it is an upper bound on CRPD of the first $j-1$ jobs of τ_i.

◇ E_i, which is the worst-case execution time, assumed by the system model, of the first $l_i - 1$ NPRs.

◇ $\gamma_{i,l_i-1}(I_i)$ – By Propostion 9, it is an upper bound on CRPD of a single job of τ_i (in this case $\tau_{i,j}$) in the first $l_i - 1$ NPRs during the time interval I_i.

◇ $\sum_{\forall \tau_h \in hp(i)} (\lfloor \frac{S_{i,j}}{T_h} \rfloor + 1) \times C_h$ – Worst-case interference caused by the execution of jobs of higher priority tasks. The interference is accounted as the sum of individual WCET values of each job with higher priority than τ_i, that can be released during $S_{i,j}$. The maximum number of jobs, of a single higher priority task τ_h, which can be released during $S_{i,j}$ is accounted by term $\lfloor \frac{S_{i,j}}{T_h} \rfloor + 1$.

◇ $\sum_{\forall \tau_h \in hp(i)} \gamma_h(S_{i,j})$ – CRPD upper bound on all jobs of the higher priority tasks that can be released during $S_{i,j}$. By Proposition 10, $\gamma_h(S_{i,j})$ is an upper bound on the CRPD of all jobs of τ_h which can be released within $S_{i,j}$. Then, the sum of CRPD bounds $\gamma_h(S_{i,j})$ for each higher priority task than τ_i is a safe upper bound on the cumulative CRPD in all jobs with higher priority than τ_i, released during $S_{i,j}$.

Since Equation 12 sums and considers upper bounds on all factors that can delay the start time of the last NPR of $\tau_{i,j}$, the proposition holds. ■

For the computation of $F_{i,j}$ (see Fig. 2) we need to consider the worst-case execution time of the last NPR of τ_i including the upper bound on CRPD:

Definition 15: The worst-case execution time q_i^{last} of the last NPR of τ_i, including the upper bound on CRPD during its execution, is defined in the following equation:

$$q_i^{last} = q_{i,l_i} + \left| UCB_{i,l_i-1} \cap \bigcup_{\forall h \in hp(i)} ECB_h \right| \times BRT \tag{13}$$

Proposition 13: q_i^{last} is an upper bound on the longest execution time of the last NPR of τ_i, including the upper bound on CRPD during its execution.

Proof: The CRPD of a non-preemptive region δ_{i,l_i} can not be higher than BRT multiplied by the number of cache blocks such that: 1) the block is useful at the point preceding δ_{i,l_i}, and 2) the block can be evicted by some task with higher priority than τ_i at PP_{i,l_i-1}. This is accounted in Equation 13 by the set intersection between UCB_{i,l_i-1} and union of ECB_h sets. Since the equation sums the WCET of δ_{i,l_i} and its CRPD upper bound, then the proposition holds. ■

Definition 16: The latest finish time $F_{i,j}$ of the j-th instance of τ_i is equal to:

$$F_{i,j} = S_{i,j} + q_i^{last} \tag{14}$$

Proposition 14: Assuming that the Level-i pending workload is greater than zero at each time point from the critical instant until $j \times T_i$, then starting from the critical instant, $F_{i,j}$ is the upper bound on the latest finish time of the last NPR of $\tau_{i,j}$.

Proof: Follows from Propositions 12 and 13. ■

Now, we define the upper bound on the Level-i active period accounting for the CRPD.

Definition 17: The level-i active period L_i, accounting for the CRPD on all jobs within the period, is defined as follows:

$$\begin{cases} L_i^{(0)} = b_i + C_i \\ L_i^{(r)} = b_i + \sum\limits_{\forall \tau_k \in hpe(i)} ((\lfloor L_i^{(r-1)}/T_k \rfloor + 1) \times C_k + \gamma_k(L_i^{(r-1)})) \end{cases} \tag{15}$$

Proposition 15: L_i is an upper bound on Level-i period.

Proof: Similar to the proof for Proposition 12. Consider the least fixed point of Equation 15, for which $L_i = L_i^r = L_i^{r-1}$. At this point, the equation accounts for the following upper bounds and worst-case execution times:

⋄ b_i – By Proposition 1 and Definition 4, b_i is an upper bound on lower priority blocking on τ_i.

⋄ $\sum_{\forall \tau_k \in hpe(i)} (\lfloor \frac{L_i}{T_k} \rfloor + 1) \times C_h$ – The worst-case execution time of all jobs of τ_i and higher priority jobs than τ_i which can be released within L_i. The maximum number of jobs, of a single task $\tau_k \in hpe(i)$, which can be released during L_i is accounted by the following term $\lfloor \frac{L_i}{T_k} \rfloor + 1$.

⋄ $\sum_{\forall \tau_k \in hpe(i)} \gamma_k(L_j)$ – CRPD upper bound on all jobs of τ_i and higher priority jobs than τ_i which can be released within L_i. Follows from Proposition 10.

Since Equation 15 sums and considers upper bounds on all factors that can extend the duration of the Level-i active period, the proposition holds. ■

Finally, we can define the worst-case response time:

Definition 18: The worst case response time R_i of τ_i is equal to the maximum relative latest finish time of a job $\tau_{i,j}$ which is released within L_i.

$$R_i = \max_{j \in [1, \lceil L_i/T_i \rceil]} \{F_{i,j} - (j-1) \times T_i\} \tag{16}$$

Theorem 2: R_i is the safe upper bound on the worst-case response time of τ_i.

Proof: Since each respective $F_{i,j}$ value is computed within L_i, then the proposition assumption from Proposition 14 holds. Then, by Theorem 1 R_i is the safe upper bound on the worst-case response time of τ_i, which concludes the proof. ■

1) Schedulability analysis: Given the above-proposed response time analysis, the taskset schedulability is determined by computing whether for each task the derived response time is less than or equal to the task deadline. I.e. if for any task τ_i in Γ the inequation $R_i > D_i$ holds, the taskset is deemed unschedulable by the proposed analysis. As described before, in Section III-A, the schedulability test can be simplified to the first job of each task, as proposed by Yao et al. [11]. Also, the proposed approach can be adopted in another simplified schedulability test (see Theorem 3 from [11]), which is based on discontinuous points of cumulative execution-request functions, proposed by Bini and Buttazzo [31]. In this case, the cumulative execution request for τ_i and all higher priority tasks can be computed using the second line of Equation 11 by changing the term $I_i^{(r-1)}$ with t, where $t \in (0, D_i - q_i^{last}]$, while the maximum blocking on τ_i is computed using Equation 2.

D. Set-associative LRU caches

As shown by Altmeyer et al. [8], [9], in case of set-associative LRU caches, a cache-set may contain several useful cache blocks, e.g., $UCB_{1,2} = \{1, 2, 2, 2\}$, meaning that at $PP_{1,2}$, task τ_1 contains three UCBs in cache-set 2, and one UCB in cache set 1. When a preemption point is preempted, one ECB of a preempting task is enough to evict all UCBs of the same cache-set. Therefore, multiple accesses to the same set by the preempting task do not need to appear in the ECB set, which means that the ECB set can remain the same as in direct-mapped caches. A bound on CRPD due to preemption from τ_h on $PP_{i,k}$ can be defined with: $UCB_{i,k} \cap' ECB_h$ where the result is a multiset that contains each element from UCB_h if it is also in ECB_i. Lastly, the RCB definition needs to be modified to account for each reloadable block as many times as it is present in the UCB set at each preemption point where the reload-ability condition from Equation 3 is satisfied. This is achieved by updating Equation 3 with $m \in UCB_{i,k}^x \cap' ECB_i$.

VI. Evaluation

In this section, we present and discuss the evaluation results on the effectiveness of different approaches to identify schedulable tasksets. We primarily compare the existing feasibility analysis (*Feasibility Analysis*), proposed by Yao et al. [10], [11], and the proposed RTA analysis (denoted as *CRPD-Fixed-PP*). Additionally, we compare those two approaches with state of the art CRPD analysis (*Combined-Multiset*) for fully-preemptive scheduling, proposed by Altmeyer et al. [8], and unsafe (*NO-CRPD*) which does not account for any preemption cost on preemption points, thus resulting in an unsafe approximation.

For the evaluation, we use basic cache-set configurations obtained with a low-level analysis tool, named LLVMTA [32]. LLVMTA uses LLVM machine intermediate representation after

Program	ECB	UCB	Max	Program	ECB	UCB	Max
adpcm	256	230	103	lcdnum	51	11	9
bs	43	23	20	lms	242	134	38
bsort100	57	40	30	ludcmp	210	168	44
cnt	123	58	44	matmult	85	51	31
compress	247	150	63	minver	256	178	47
cover	256	38	15	ndes	253	176	38
crc	121	62	30	ns	55	37	34
edn	256	222	123	nsichneu	256	183	2
expint	117	47	29	prime	75	47	33
fdct	126	113	62	qsort-ex.	142	83	39
fft1	222	154	63	qurt	130	40	26
fibcall	28	16	16	select	159	73	55
fir	94	42	21	sqrt	53	21	12
insertsort	29	16	15	st	192	95	52
janne_co.	39	28	27	statemate	256	105	1
jfdctint	132	122	54	ud	194	151	39

TABLE I: Cache configurations obtained with LLVMTA [32] analysis tool used on Mälardalen benchmark programs [33].

all backend compiler passes have run. The resulting machine program corresponds to a CFG reconstruction of a binary file. Cache-set configurations are obtained on real-time programs from the Mälardalen benchmark [33], given in Table I. The obtained parameters per each program are set of evicting (ECB) and definitely useful cache blocks (UCB) (in the table we show the size of each set). Also, the maximum number (Max) of definitely useful cache blocks per any preemption point of each program is obtained. The assumed direct-mapped cache memory consists of 256 sets with a line size of 8 bytes. For more details about the low-level analysis refer to [34].

In all of the experiments, we generate 2000 tasksets for each investigated parameter value. In order to include realistic cache-set usage, each task in a taskset is randomly assigned one of the distinct cache-set configurations presented in Table I. Since the task binaries were analysed individually, they all start at the same address (mapping to cache set 0). In a multi-task scheduling situation, this can hardly be the case because the ECB and UCB placement is determined by their respective locations in memory. We took this into account by randomly shifting the cache set indices, e.g. the ECB in cache set i is shifted to the cache line equal to $(i + random(256))$ $modulo$ 256. Further task parameters were generated using the evaluation setup proposed by Altmeyer et al. [18], [8] which is used in many CRPD-related papers in order to exhaustively evaluate CRPD-based analyses. Utilisation of each task was generated using the standardised algorithm in real-time research – UUnifast, proposed by Bini and Butazzo [35]. The task periods were generated according to a uniform distribution from 5 to 500 milliseconds, representing the general spread of periods in automotive and aerospace hard real-time applications [18]. The worst-case execution time for each task was computed with the following equation $C_i = U_i \times T_i$, where U_i represents the utilisation of τ_i. Task deadlines were implicit, meaning that $D_i = T_i$. Task priorities were assigned according to the deadline-monotonic order. Furthermore, in order to allow for an exhaustive evaluation, we generated other parameters for LP-FPP task model and in different experiments we varied different parameters. We finish this paragraph by enlisting the default evaluation setup. The number of subjobs for each task was randomly generated according to a uniform distribution from the range $[3, 100]$ as this is a representative range of non-preemptive regions used in automotive real-time applications (where NPRs are also called runnables). The worst-case execution time of each subjob was set to C_i/l_i. The default taskset size was set to 6. For each preemption point $PP_{i,k}$ of a task τ_i, its useful cache blocks $UCB_{i,k}$ were generated from the UCB set, assigned to τ_k from Table I. Also, we pessimistically assumed that each point of τ_k exhibits a maximum number of UCBs, given in the table. Accordingly, each evicting cache block was included at the NPRs prior and after the preemption point where it is useful. The assumed block reload time was 8 microseconds, as given in [18].

In order to increase the exhaustiveness of the performed evaluation, we used the weighted schedulability measure which also enables emulation of a 3-dimensional plot to a 2-dimensional one, as proposed by Bastoni et al. [7]. In the shown figures, we show the weighted schedulability measure $W_y(\rho)$, for schedulability test y as a function of parameter ρ. For each value of ρ, this measure combines data for all of the tasksets generated for each utilisation level from 0.7 to 1, with a step of 0.2. For each value ρ of the selected parameter, the schedulability measure is $W_y(\rho) = \sum_{\forall\Gamma}(U_\Gamma \times B_y(\Gamma, \rho)) / \sum_{\forall\Gamma} U_\Gamma$, where $B_y(\Gamma, U_i)$ is a result (1 if schedulable, 0 otherwise) of a schedulability test y for a taskset Γ and parameter value ρ (U_Γ is a taskset utilisation). A method producing the highest weighted measure is the most prone to identify schedulable tasksets.

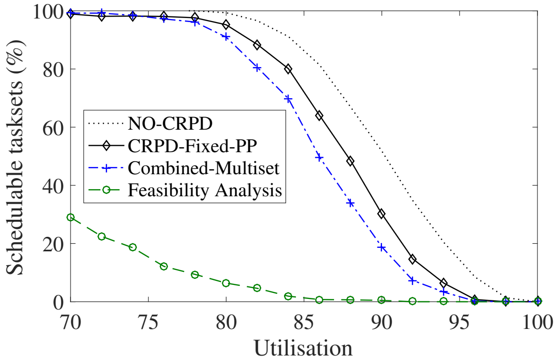

Fig. 4: Schedulability ratio at different taskset utilisation.

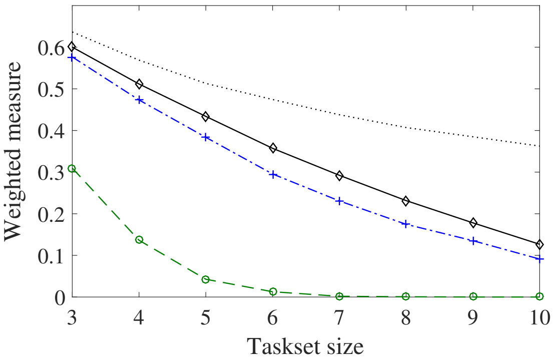

Fig. 5: Weighted measure at different taskset size.

In the first experiment (Fig.4), we evaluated the impact of the taskset utilisation (x-axis) on the effectiveness of the analyses to determine the taskset schedulability (y-axis). We notice that *CRPD-Fixed-PP* succeeds to identify significantly more schedulable tasksets compared to (*Feasibility Analysis*) and (*Combined-Multiset*), e.g. for $U_\Gamma = 0.88$, *CRPD-Fixed-PP* identifies 48%, *Combined-Multiset* 34%, and *Feasibility-Analysis* only 0.6% of all generated tasksets as schedulable.

In the second experiment (Fig.5), we evaluated the impact of the taskset size (x-axis) on the effectiveness of the analyses to determine the taskset schedulability (y-axis), with the weighted measure. The evaluated taskset size ranges from 3 to 10. We notice that with the increase in taskset size, all methods deteriorate in finding schedulable tasksets. However, *CRPD-Fixed-PP* deteriorates with the lowest rate, still being able to find the largest number of schedulable tasksets compared to *Feasibility-Analysis* and *Combined-Multiset*.

In the third experiment (Fig.6), we evaluated the impact of the numbers of NPRs per task (x-axis) on the effectiveness of the analyses to determine the taskset schedulability (y-axis), with the weighted measure. In this experiment, we randomly generated the number of NPRs from the uniform distribution in the range [3,x] where x is a value on the x-axis from the figure. We notice that the greater the number of NPRs, the higher the weighted schedulability of *CRPD-Fixed-PP*, meaning it is more prone to identify schedulable tasksets. This is the case because the greater the number of NPRs, the lower the blocking from the lower priority tasks. In contrast, with *Feasibility-Analysis* it is the opposite because the greater number of NPRs, the greater estimation of preemption cost in this case.

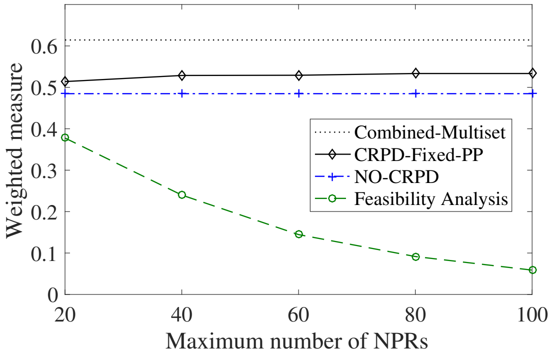

Fig. 6: Weighted measure at different upper bound on number of non-preemptive regions.

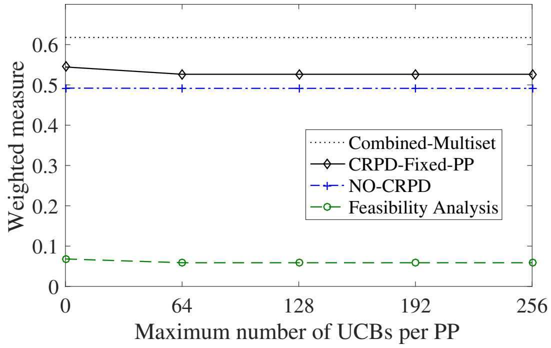

Fig. 7: Weighted measure at different upper bound on number of UCBs per preemption point.

In the fourth experiment (Fig.7), we evaluated the impact of the number of UCBs per preemption point (x-axis) on the effectiveness of the analyses to determine the taskset schedulability (y-axis), with the weighted measure. In this experiment, we removed the pessimistic consideration from the default evaluation setup that all preemption points exhibit the maximum number (Max from Table I) of UCBs at each preemption point. Therefore, for each preemption point $PP_{i,k}$, we randomly generated the number of UCBs from the uniform distribution in the range $[x, Max_i]$ where x is a value on the x-axis from the figure, and Max_i the maximum number of UCBs of τ_i, derived from the configuration assigned from Table I. In cases when x value exceeded Max_i, Max_i was assigned. We notice that only *CRPD-Fixed-PP* is affected in this experiment. In case when the number of UCBs per point is derived from the range $[0, Max_i]$ the proposed method is more prone to identify schedulable taskset, as it accounts for the variability of CRPD throughout the task's execution, unlike the other presented methods.

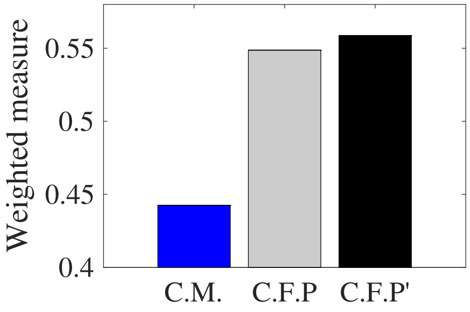

Fig. 8: Weighted measure at different reload-ability conditions.

In the fifth experiment (Fig.8), we evaluated the impact of changing the reload-ability condition on CRPD-aware methods. We considered two cases. Bars (C.M. – *Combined-Multiset*) and (C.F.P. – *CRPD-Fixed-PP*) represent the weighted measures obtained with default evaluation setup, where each UCB within the task's execution is definitely reloadable (since ECBs are assigned in NPRs directly preceding and succeeding the point where the block is useful). Bar (C.F.P'. – *CRPD-Fixed-PP*) considers the case where UCB is reloadable only once throughout the entire sequence of preemption points where it is useful, thus resulting in more identified schedulable tasksets. This is even more important for tasks with variable initialization in the beginning, and dispatching at the task's end.

VII. Conclusions

In this paper, we propose a cache-aware response time analysis for tasks with non-preemptive regions (NPR) under fixed-priority preemptive real-time scheduling. The analysis accounts for a tighter estimation of cache-related preemption delays thus dominating the existing feasibility analysis for the fixed-point–based tasks. It is based on the observations that: 1) the number of cache blocks which can be reloaded during the execution of a task can be smaller than the number of cache blocks which can be useful in a task, 2) a single job with NPRs can be affected by cache-related preemption delay from the preemptions occurring only during the maximum time duration from the start of its first NPR, until the start of its last NPR.

We evaluated the proposed analysis by comparing its effectiveness to identify schedulable tasksets with state of the art feasibility analysis for tasks with non-preemptive regions and fully-preemptive tasks. The evaluation showed that the proposed analysis significantly improves the state of the art of such schedulability analyses, since it accounts for the estimation of a tighter bound on cache-related preemption delay.

Acknowledgement

Foremost, we are thankful to our colleagues Sebastin Hahn, Jan Reineke, and Darshit Shah who provided us with evaluation data, derived from the code-level analysis of benchmark programs. Especially, we are thankful to Sebastian Hahn who provided additional insights and computed the cache and task parameters using the low-level analysis tool from Saarland University, which significantly improved the quality of this paper. Also, we are very grateful to Davor Čirkinagić who provided us with a powerful computing system for performing the schedulability evaluation. Lastly, we are very grateful to all reviewers, whose comments were insightful and important for correcting this paper.

References

[1] G. Yao, G. Buttazzo, and M. Bertogna, "Comparative evaluation of limited preemptive methods," in *2010 IEEE 15th Conference on Emerging Technologies & Factory Automation (ETFA 2010)*. IEEE, 2010, pp. 1–8.

[2] G. C. Buttazzo, M. Bertogna, and G. Yao, "Limited preemptive scheduling for real-time systems. A survey," *industrial Informatics, IEEE Transactions on*, vol. 9, no. 1, pp. 3–15, 2013.

[3] S. Baruah, "The limited-preemption uniprocessor scheduling of sporadic task systems," in *17th Euromicro Conference on Real-Time Systems (ECRTS'05)*. IEEE, 2005, pp. 137–144.

[4] Y. Wang and M. Saksena, "Scheduling fixed-priority tasks with preemption threshold," in *Real-Time Computing Systems and Applications, 1999. RTCSA'99. Sixth International Conference on*. IEEE, 1999, pp. 328–335.

[5] A. Burns and E. S. Son, "Preemptive priority based scheduling: An appropriate engineering approach," *Advances in Real-Time Systems*, pp. 225–248, 1994.

[6] R. Pellizzoni, B. D. Bui, M. Caccamo, and L. Sha, "Coscheduling of CPU and I/O transactions in COTS-based embedded systems," in *Real-Time Systems Symposium, 2008*. IEEE, 2008, pp. 221–231.

[7] A. Bastoni, B. Brandenburg, and J. Anderson, "Cache-related preemption and migration delays: Empirical approximation and impact on schedulability," *Proceedings of OSPERT*, pp. 33–44, 2010.

[8] S. Altmeyer, R. I. Davis, and C. Maiza, "Improved cache related preemption delay aware response time analysis for fixed priority pre-emptive systems," *Real-Time Systems*, vol. 48, no. 5, pp. 499–526, 2012.

[9] S. Altmeyer, C. Maiza, and J. Reineke, "Resilience analysis: tightening the CRPD bound for set-associative caches," in *ACM Sigplan Notices*, vol. 45, no. 4. ACM, 2010, pp. 153–162.

[10] G. Yao, G. Buttazzo, and M. Bertogna, "Feasibility analysis under fixed priority scheduling with fixed preemption points," in *2010 IEEE 16th International Conference on Embedded and Real-Time Computing Systems and Applications*. IEEE, 2010, pp. 71–80.

[11] ——, "Feasibility analysis under fixed priority scheduling with limited preemptions," *Real-Time Systems*, vol. 47, no. 3, pp. 198–223, 2011.

[12] A. Hamann, D. Dasari, S. Kramer, M. Pressler, and F. Wurst, "Communication centric design in complex automotive embedded systems," in *29th Euromicro Conference on Real-Time Systems (ECRTS 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[13] M. Becker, N. Khalilzad, R. J. Bril, and T. Nolte, "Extended support for limited preemption fixed priority scheduling for osek/autosar-compliant operating systems," in *10th IEEE International Symposium on Industrial Embedded Systems (SIES)*. IEEE, 2015, pp. 1–11.

[14] L. Hatvani, R. J. Bril, and S. Altmeyer, "Schedulability using native non-preemptive groups on an autosar/osek platform with caches," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, 2017, pp. 244–249.

[15] C.-G. Lee, J. Han, Y.-M. Seo, S. L. Min, R. Ha, S. Hong, C. Y. Park, M. Lee, and C. S. Kim, "Analysis of cache-related preemption delay in fixed-priority preemptive scheduling," *IEEE Transactions on Computers*, vol. 47, no. 6, pp. 700–713, 1998.

[16] S. Altmeyer and C. Burguiere, "A new notion of useful cache block to improve the bounds of cache-related preemption delay," in *Real-Time Systems, 2009. ECRTS'09. 21st Euromicro Conference on*. IEEE, 2009, pp. 109–118.

[17] Y. Tan and V. Mooney, "Timing analysis for preemptive multitasking real-time systems with caches," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 6, no. 1, p. 7, 2007.

[18] S. Altmeyer, R. I. Davis, and C. Maiza, "Cache related pre-emption delay aware response time analysis for fixed priority pre-emptive systems," in *2011 IEEE 32nd Real-Time Systems Symposium*. IEEE, 2011, pp. 261–271.

[19] H. Tomiyama and N. D. Dutt, "Program path analysis to bound cache-related preemption delay in preemptive real-time systems," in *Proceedings of the eighth international workshop on Hardware/software codesign*. ACM, 2000, pp. 67–71.

[20] J. V. Busquets-Mataix, J. J. Serrano, R. Ors, P. Gil, and A. Wellings, "Adding instruction cache effect to schedulability analysis of preemptive real-time systems," in *Real-Time Technology and Applications Symposium, 1996. Proceedings., 1996 IEEE*. IEEE, 1996, pp. 204–212.

[21] H. Ramaprasad and F. Mueller, "Tightening the bounds on feasible preemption points," in *2006 27th IEEE International Real-Time Systems Symposium (RTSS'06)*. IEEE, 2006, pp. 212–224.

[22] ——, "Bounding worst-case response time for tasks with non-preemptive regions," in *2008 IEEE Real-Time and Embedded Technology and Applications Symposium*. IEEE, 2008, pp. 58–67.

[23] R. J. Bril, J. J. Lukkien, and W. F. Verhaegh, "Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption," *Real-Time Systems*, vol. 42, no. 1-3, pp. 63–119, 2009.

[24] M. Bertogna, O. Xhani, M. Marinoni, F. Esposito, and G. Buttazzo, "Optimal selection of preemption points to minimize preemption overhead," in *Real-Time Systems (ECRTS), 2011 23rd Euromicro Conference on*. IEEE, 2011, pp. 217–227.

[25] B. Peng, N. Fisher, and M. Bertogna, "Explicit preemption placement for real-time conditional code," in *Real-Time Systems (ECRTS), 2014 26th Euromicro Conference on*. IEEE, 2014, pp. 177–188.

[26] J. Cavicchio, C. Tessler, and N. Fisher, "Minimizing cache overhead via loaded cache blocks and preemption placement," in *Real-Time Systems (ECRTS), 2015 27th Euromicro Conference on*. IEEE, 2015, pp. 163–173.

[27] F. Markovic, J. Carlson, and R. Dobrin, "Tightening the bounds on cache-related preemption delay in fixed preemption point scheduling," in *17th International Workshop on Worst-Case Execution Time Analysis (WCET 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[28] F. Marković, J. Carlson, and R. Dobrin, "Improved cache-related preemption delay estimation for fixed preemption point scheduling," in *Ada-Europe International Conference on Reliable Software Technologies*. Springer, 2018, pp. 87–101.

[29] G. Buttazzo, *Hard real-time computing systems: predictable scheduling algorithms and applications*. Springer Science & Business Media, 2011, vol. 24.

[30] R. J. Bril, W. F. Verhaegh, and J. J. Lukkien, "Exact worst-case response times of real-time tasks under fixed-priority scheduling with deferred preemption," in *Proc. Work-in-Progress (WiP) session of the 16th Euromicro Conference on Real-Time Systems (ECRTS), Technical Report from the University of Nebraska-Lincoln, Department of Computer Science and Engineering (TR-UNL-CSE-2004-0010)*, 2004, pp. 57–60.

[31] E. Bini and G. C. Buttazzo, "Schedulability analysis of periodic fixed priority systems," *IEEE Transactions on Computers*, vol. 53, no. 11, pp. 1462–1473, 2004.

[32] S. Hahn, M. Jacobs, and J. Reineke, "Enabling compositionality for multicore timing analysis," in *Proceedings of the 24th international conference on real-time networks and systems*. ACM, 2016, pp. 299–308.

[33] J. Gustafsson, A. Betts, A. Ermedahl, and B. Lisper, "The mälardalen wcet benchmarks: Past, present and future," in *10th International Workshop on Worst-Case Execution Time Analysis (WCET 2010)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2010.

[34] D. Shah, S. Hahn, and J. Reineke, "Experimental evaluation of cache-related preemption delay aware timing analysis," in *18th International Workshop on Worst-Case Execution Time Analysis (WCET 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

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