Integrated Analysis of Cache Related Preemption Delays and Cache Persistence Reload Overheads

Syed Aftab Rashid*, Geoffrey Nelissen*, Sebastian Altmeyer†, Robert I. Davis‡, Eduardo Tovar*
*CISTER/INESC TEC, ISEP, Polytechnic Institute of Porto, Portugal, †University of Amsterdam, Netherlands
‡University of York, UK

***Abstract*—Schedulability analysis for tasks running on microprocessors with cache memory is incomplete without a treatment of Cache Related Preemption Delays (CRPD) and Cache Persistence Reload Overheads (CPRO). State-of-the-art analyses compute CRPD and CPRO independently, which might result in counting the same overhead more than once.**

In this paper, we analyze the pessimism associated with the independent calculation of CRPD and CPRO in comparison to an integrated approach. We answer two main questions: (1) Is it beneficial to integrate the calculation of CRPD and CPRO? (2) When and to what extent can we gain in terms of schedulability by integrating the calculation of CRPD and CPRO? To achieve this, we (i) identify situations where considering CRPD and CPRO separately might result in overestimating the total memory overhead suffered by tasks, (ii) derive new analyses that integrate the calculation of CRPD and CPRO; and (iii) perform a thorough experimental evaluation using benchmarks to compare the performance of the integrated analysis against the separate calculation of CRPD and CPRO.

I. Introduction

The increasing gap between processor and main memory speeds has motivated the introduction of caches in modern microprocessors. Program data and instructions that are loaded into cache are available to the processor in a few clock cycles compared to fetches from main memory which may take tens or even hundreds of clock cycles. Most Commercial-Off-The-Shelf (COTS) microprocessors use caches to decrease average-case memory access latency; however, as caches have a limited capacity in comparison to main memory, typically not all of the data and instructions of all tasks can simultaneously reside in the cache. With an unpartitioned cache, tasks compete for limited cache space, with the execution of one task potentially evicting memory blocks previously loaded into the cache by other tasks. This can cause large variations in the execution times of the tasks, depending on whether the instructions and data that they require are already present in the cache or not.

In systems where preemptions are allowed, preempted tasks may suffer additional delays if *useful cache blocks* (UCBs) (that are resident in the cache and will be re-used before being replaced) are evicted from the cache by preempting tasks. Such evictions cause Cache-Related Preemption Delays (CRPDs) to occur after task resumption when the useful cache blocks are reloaded from main memory.

Considering multiple jobs of a particular task; the next job of the task can benefit from the presence in cache of persistent memory blocks that were loaded by a previous job of the same task and that have remained in the cache until the next job executes and can make use of those blocks. These cache blocks are called *Persistent Cache Blocks* (PCBs) and this concept is referred to as *cache persistence*[1] [21]. Analysis of cache persistence can be used to reduce pessimism in the computation of interference from multiple jobs of a higher priority task in state-of-the-art worst-case response time (WCRT) analysis for systems using Fixed Priority Preemptive Scheduling (FPPS). The PCBs of a task are identified assuming that the task runs in isolation, i.e. assuming there are no other tasks in the system. In practice this is not the case, PCBs may be evicted due to interleaved or preemptive execution of other tasks, leading to *Cache Persistence Reload Overheads* (CPRO).

In [21], the authors derived two analyses for CPRO that were integrated into an improved response time analysis for FPPS that takes account of reductions in memory demand due to cache persistence, along with the CRPD. Their analysis considers both the CRPD and CPRO and dominates the state-of-the-art approaches that only consider CRPD. However, the analysis in [21] may sometimes result in over-estimation of the task response times. This is due to the fact that CRPD and CPRO are calculated separately, providing independent upper bounds on the two classes of overheads. However, as we later show, scenarios maximising CRPD and those maximising CPRO may be mutually exclusive, meaning that the total overheads can be substantially less than the sum of the two bounds.

In this paper we focus on two questions: (1) Is it beneficial to integrate the calculation of CRPD and CPRO to remove the over-estimation in the total overheads of tasks? (2) Under what conditions and by how much can we gain in terms of schedulability by integrating the calculation of CRPD and CPRO? We answer these questions by: (i) identifying situations where considering CRPD and CPRO separately might result in overestimating the total memory overhead suffered by tasks due to double counting of some memory blocks that need to be reloaded, (ii) demonstrating how to integrate the calculation of CRPD and CPRO to include only the additional CPRO that are not already included in the CRPD calculation, and (iii) through experimental evaluation using a set of benchmarks to derive important observations that lead to situations where the integrated CRPD-CPRO analysis may or may not outperform separate treatment of CRPD and CPRO.

A. Related Work

Early work on accounting for scheduling overheads in FPPS by Katcher et al. [15] and Burns et al. [9] focused on scheduler overheads and context switch costs. Subsequent work on the

[1]Note that this form of cache persistence between jobs is distinct from cache persistence within loops, as studied for example by Cullmann [11].

analysis of CRPD and their integration into schedulability analyses used the concepts of UCBs and *evicting cache blocks* (ECBs), i.e., all the cache blocks that are accessed by a task during its execution. A number of methods have been developed for computing CRPD under FPPS. Namely, the ECB-Only approach [10], which considers just the preempting task, as does the work by Tomiyama et al. [24]. The UCB-Only approach [16], which considers just the preempted task(s). The UCB-Union [23], ECB-Union [2] and an alternative approach developed by Staschulat et al. [22] that consider both the preempted and preempting tasks. These approaches were later superseded by multi-set based methods (ECB-Union Multi-set and UCB-Union Multi-set) which dominate them [3]. These methods have been adapted to EDF scheduling [17], [20] and to hierarchical scheduling with local fixed priority [18] and EDF [19] schedulers. They have also been integrated into analysis for multi-core systems [1].

Cache partitioning is one way of eliminating CRPD; however, this results in inflated WCETs due to the reduced cache partition size available to each task. Altmeyer et al. [4], [5] derived an optimal cache partitioning algorithm when each task has its own partition. They concluded that the trade off between longer WCETs and CRPD often favours sharing the cache rather than partitioning it.

The notion of cache persistence and CPRO was recently introduced in [21]. Methods to compute the CPRO cost and to integrate it in the WCRT analysis for FPPS were proposed, showing significant improvement in the accuracy of the response time analysis.

II. System Model

In this work, we focus on single-core platforms with a single level (L1) instruction cache. The cache is assumed to be direct-mapped[2], which means that each memory block in the main memory can be mapped to only one specific block in the cache[3].

We consider a sporadic task model where each task has a *unique* fixed priority. Any priority assignment scheme (e.g., Rate Monotonic or Deadline Monotonic) is acceptable. We also assume that the tasks are independent and do not suspend themselves during their execution. A task τ_i is defined by a triplet (C_i, T_i, D_i), where C_i is the worst-case execution time (WCET) of τ_i, T_i is its minimum inter-arrival time and D_i is the relative deadline of each instance (or job) of τ_i. We assume that the tasks have constrained deadlines, i.e., $D_i \leq T_i$. We further decompose each task's WCET into separate terms for processing and memory access demand, respectively. The worst-case processing demand PD_i denotes the worse-case execution time of τ_i considering that every memory access is a cache hit. Consequently, it only accounts for execution requirements of the task and does not include the time needed to fetch data and instructions from the main memory. MD_i is the worst-case memory access demand of any job of task τ_i; that is, the maximum time during which any job of τ_i is performing memory operations. The values for C_i, PD_i and MD_i are determined assuming τ_i executes *in isolation* (i.e., without preemption, starting from an empty cache). It is also important to note that the worst-case processing demand and the worst-case memory access demand may not necessarily be experienced on the same execution path of τ_i. Therefore, it holds that $C_i \leq PD_i + MD_i$. The worst-case response time (WCRT) of task τ_i, denoted by R_i, is defined as the longest time between the arrival and the completion of any of its jobs.

We consider that preemption costs only refer to additional cache reloads due to those preemptions. Other overheads that remain constant over the execution of a task, e.g., due to context switches and scheduler invocations, are assumed to be included in the task's WCET. The worst-case reload time of a cache block from main memory is denoted by d_{mem}.

We use $\mathrm{hp}(i)$ to denote the set of tasks with priorities higher than that of τ_i. Similarly, $\mathrm{lp}(i)$ to denote the set of tasks with priorities lower than that of τ_i. Further, $\mathrm{hep}(i)$ denotes the set of tasks with priorities higher than or equal to that of τ_i (i.e. $hep(i)$ includes τ_i). Finally, $\mathrm{aff}(i,j) = \mathrm{hep}(i) \cap \mathrm{lp}(j)$ denotes the set of intermediate tasks that can execute during the response time of τ_i but may also be preempted by some higher priority task τ_j.

Note in this paper, similar to earlier work on CRPD and CPRO, we assume a timing-compositional architecture [14], i.e. the timing contribution of memory overheads can be analyzed separately from other architectural features.

III. Existing CRPD and CPRO Analysis

In this section, we provide definitions for a number of key concepts and summarize existing analyses of CRPD and CPRO, which we later build upon.

A. Cache Related Preemption Delays

Definition 1 (Useful Cache Block (UCB) [16])**.** A memory block m is called a Useful Cache Block at program point P, if it is cached at P and will be reused at program point Q that may be reached from P without eviction of m.

In this work we use the basic UCB definition from [16]; however, our approach is also compatible with the refined definition given by Altmeyer et al. [6].

Definition 2 (Evicting Cache Block (ECB) [10])**.** Any cache block accessed during the execution of the task and which can then evict the memory block cached by another task is called an Evicting Cache Block.

We now summarize the UCB-union and the UCB-union multi-set approaches to CRPD analysis, which we later build upon. We denote the CRPD caused by a task τ_j executing during the response time of a task τ_i by $\gamma_{i,j}$.

To calculate the preemption cost $\gamma^{ucb}_{i,j}$, the UCB-union approach [23] uses the ECBs of the preempting task τ_j and the UCBs of all tasks in $aff(i,j)$ possibly affected by the preemption caused by τ_j:

$$\gamma^{ucb}_{i,j} = d_{mem} \times \left| \left(\bigcup_{\forall k \in aff(i,j)} UCB_k \right) \cap ECB_j \right| \quad (1)$$

where, UCB_k and ECB_j are the sets of UCBs and ECBs of task τ_k and τ_j, respectively. The preemption cost can then be accounted for in the WCRT analysis as follows:

[2]Examples of microprocessors with direct-mapped caches include the Renesas SH7750 and NEC VR4181 and VR4121.

[3]In common with most critical real-time systems, we assume that the platform does not provide memory address translation or virtual memory.

$$R_i^{ucb} = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i^{ucb}}{T_j} \right\rceil \times (C_j + \gamma_{i,j}^{ucb}) \quad (2)$$

where the WCRT of τ_i is the smallest positive solution to (2).

Note that the UCB-union approach does not take into account the actual number of job releases of a task. Therefore, it overestimates the number of preemptions tasks can cause or suffer and hence results in pessimistic CRPD bounds. To reduce this pessimism, a multi-set extension of this analysis was proposed in [3].

The UCB-union multi-set approach [3] takes into account the maximum number of jobs $E_j(R_i) \stackrel{\text{def}}{=} \left\lceil \frac{R_i}{T_j} \right\rceil$ that each higher priority task τ_j can release during the response time of τ_i. It upper bounds the number of preemptions each task $\tau_k \in \text{aff}(i,j)$ can suffer due to a higher priority task τ_j during the response time of τ_i by $E_j(R_k)E_k(R_i) \stackrel{\text{def}}{=} \left\lceil \frac{R_k}{T_j} \right\rceil \times \left\lceil \frac{R_i}{T_k} \right\rceil$. The resulting CRPD cost is denoted by $\gamma_{i,j}^{ucb-m}$ and it accounts for the total preemption cost that can be caused by all jobs of τ_j released during the response time of τ_i. $\gamma_{i,j}^{ucb-m}$ is given by

$$\gamma_{i,j}^{ucb-m} = d_{mem} \times \left| M_{i,j}^{ucb} \cap M_{i,j}^{ecb} \right| \quad (3)$$

where $M_{i,j}^{ucb}$ and $M_{i,j}^{ecb}$ are multi-sets defined as

$$M_{i,j}^{ucb} = \bigcup_{\forall k \in aff(i,j)} \left(\bigcup_{E_j(R_k)E_k(R_i)} UCB_k \right) \quad (4)$$

$$M_{i,j}^{ecb} = \bigcup_{E_j(R_i)} ECB_j \quad (5)$$

The UCB-union multi-set approach dominates the UCB-union approach and provides more precise bounds on the CRPD cost by using the following WCRT equation.

$$R_i^{ucb-m} = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i^{ucb-m}}{T_j} \right\rceil \times C_j + \sum_{\forall j \in hp(i)} \gamma_{i,j}^{ucb-m} \quad (6)$$

For a more detailed description of the formulation of (3) to (6), including worked examples, see [3].

B. Cache Persistence

The notion of cache persistence and the concept of *persistent and non-persistent cache blocks (PCBs and nPCBs)* was introduced by Rashid et al. [21].

Definition 3 (Persistent Cache Block (PCB))**.** A memory block of a task τ_i is called a persistent cache block, if once loaded by τ_i, it will **never** be invalidated or evicted from the cache (in the same or different job of τ_i) when τ_i executes **in isolation**.

Definition 4 (non-Persistent Cache Block (nPCB))**.** A non-persistent cache block of task τ_i is an ECB that is not a PCB. That is, it is a memory block that may need to be reloaded at some point during the execution of τ_i (in the same or different job), even when τ_i executes in isolation.

Based on the definition of non-persistent cache blocks (nPCBs), the notion of the *residual memory demand (MD_i^r)* of a task τ_i is defined as follows.

Definition 5 (Residual Memory Demand)**.** The residual memory demand MD_i^r of task τ_i is the worst-case memory demand of any job of τ_i when all its PCBs are already loaded in the cache memory.

The number of PCBs and the residual memory demand (MD_i^r) of a task can be used to bound its total memory demand $\hat{MD}_i(t)$ in *isolation* during a time interval of length t:

$$\hat{MD}_i(t) \stackrel{\text{def}}{=} \min \left\{ \left\lceil \frac{t}{T_i} \right\rceil MD_i \; ; \left\lceil \frac{t}{T_i} \right\rceil MD_i^r + \mid PCB_i \mid \times d_{mem} \right\} \quad (7)$$

The notion of CPRO is also defined in [21] as:

Definition 6 (Cache-Persistence Reload Overhead (CPRO))**.** The cache-persistence reload overhead denoted by $\rho_{j,i}$ is the maximum memory reload overhead suffered by a task τ_j due to evictions of its PCBs by tasks in $\text{hep}(i) \setminus \tau_j$ while τ_j is executing during the response time of τ_i.

CPRO can be calculated using the CPRO-union and the CPRO multi-set approaches [21]. The CPRO-union approach uses the PCBs of task τ_j and the union of the ECBs of all tasks in $\text{hep}(i) \setminus \tau_j$ to calculate the total CPRO $\rho_{j,i}^{union}$ of task τ_j during the response time of task τ_i as follows:

$$\rho_{j,i}^{union} \stackrel{\text{def}}{=} \left(\left\lceil \frac{R_i}{T_j} \right\rceil - 1 \right) \times \rho'_{j,i} \quad (8)$$

where $\rho'_{j,i}$ is the CPRO associated with a single job of τ_j.

$$\rho'_{j,i} = d_{mcm} \times \left| PCB_j \cap \left(\bigcup_{\forall \tau_k \in \text{hep}(i) \setminus \tau_j} ECB_k \right) \right| \quad (9)$$

The CPRO-union approach assumes that the ECBs of all tasks $\tau_k \in \text{hep}(i) \setminus \tau_j$ are interfering with every job of τ_j released within the response time of τ_i. This is pessimistic. Indeed, considering two different tasks τ_k and τ_l in $\text{hep}(i) \setminus \tau_j$, the number of times τ_l can execute between different jobs of τ_j is not necessarily equal to the number of times τ_k can interfere with those jobs. The CPRO multi-set approach removes this pessimism by first categorizing all the tasks that can execute during the response time of τ_i, i.e., $\tau_k \in \text{hep}(i) \setminus \tau_j$ into two different sets: $\text{hp}(j)$ and $\text{aff}(i,j)$. It then uses the actual number of executions of intermediate ($\in \text{aff}(i,j)$) and higher priority tasks ($\in \text{hp}(j)$) to bound the CPRO cost $\rho_{j,i}^{mul}$:

$$\rho_{j,i}^{mul} \stackrel{\text{def}}{=} d_{mem} \times \left| M_{j,i}^{ecb} \cap M_{j,i}^{pcb} \right| \quad (10)$$

where $M_{j,i}^{ecb}$ and $M_{j,i}^{pcb}$ are multi-sets defined as

$$M_{j,i}^{pcb} = \bigcup_{E_j(R_i)-1} PCB_j \quad \text{and} \quad M_{j,i}^{ecb} = M_{j,i}^{ecb-aff} \cup M_{j,i}^{ecb-hp}$$

$$\text{with } M_{j,i}^{ecb-aff} = \bigcup_{\forall k \in \text{aff}(i,j)} \left(\bigcup_{(E_j(R_k)+1)E_k(R_i)} ECB_k \right) \quad (11)$$

$$M_{j,i}^{ecb-hp} = \bigcup_{\forall l \in \text{hp}(j)} \left(\bigcup_{E_l(R_i)} ECB_l \right) \quad (12)$$

Note that the CPRO multi-set approach dominates the CPRO-union approach.

When considering cache persistence and CPRO, the WCRT equation of a task τ_i under FPPS can be re-written as follows:

$$R_i = C_i + \sum_{\forall j \in \text{hp}(i)} \left(\gamma_{i,j} + \min \left\{ \left\lceil \frac{R_i}{T_j} \right\rceil C_j \; ; \left\lceil \frac{R_i}{T_j} \right\rceil PD_j + \hat{MD}_j(R_i) + \rho_{j,i} \right\} \right) \quad (13)$$

where $\gamma_{i,j}$ is either equal to $\gamma^{ucb-m}_{i,j}$ (3), or $\left\lceil \frac{R^{ucb}_i}{T_j} \right\rceil \times \gamma^{ucb}_{i,j}$ (1), and $\rho_{j,i}$ is either $\rho^{union}_{j,i}$ (8), or $\rho^{mul}_{j,i}$ (10). In the remainder of this paper, unless stated otherwise, we assume that (13) is used to calculate the WCRT of a task τ_i.

For more information on the formulation of (7)-(13), readers are referred to [21].

IV. Problem Formalization

The CRPD of a task accounts for the evictions of its UCBs due to preemptions caused by higher priority tasks. Similarly, the CPRO accounts for the evictions of its PCBs between successive job executions. Therefore, the total time spent reloading cache blocks evicted during the response time of τ_i is bounded by the sum of the CRPD and the CPRO experienced by every task executing during τ_i's response time. This overhead is denoted by μ_i and is defined as follows.

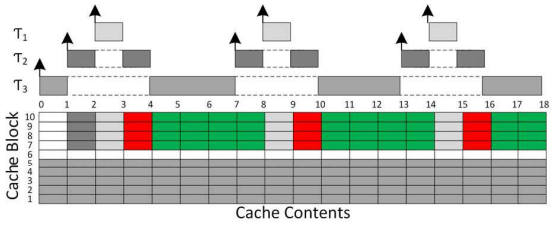
Definition 7 (Total Memory Reload Overhead (μ_i))**.** Let $CRPD_{i,j}(S)$ and $CPRO_{i,j}(S)$ be the total *actual* CRPD and CPRO suffered by τ_j during the response time of one job of τ_i in a given schedule S. The total memory reload overhead μ_i during the response time of τ_i is the maximum sum of the CRPD and CPRO of all tasks executing during τ_i's response time in any schedule S. Formally,

$$\mu_i \stackrel{\text{def}}{=} \max_{\forall S} \left\{ \sum_{\forall \tau_j \in \text{hep}(i)} \Big(CRPD_{i,j}(S) + CPRO_{i,j}(S) \Big) \right\} \quad (14)$$

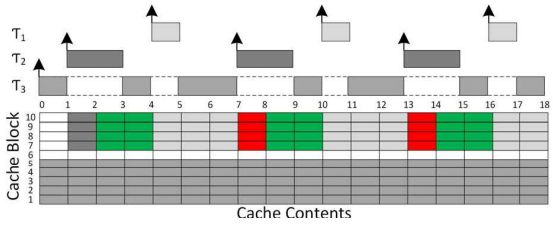
From the above definition, it follows that μ_i is upper-bounded by $\sum_{\tau_j \in \text{hep}(i)} (\gamma^{ucb-m}_{i,j} + \rho^{mul}_{j,i})$ where $\gamma^{ucb-m}_{i,j}$ and $\rho^{mul}_{j,i}$ are computed by (3) and (10), respectively. However, independently computing CRPD and CPRO may result in overestimating the actual total memory reload overhead μ_i as illustrated in the example below.

Example 1. *Let τ be composed of three tasks $\{\tau_1, \tau_2, \tau_3\}$ with τ_1 having the highest priority and τ_3 the lowest. Fig. 1 presents the task set parameters and the worst-case schedule for τ_3 together with the evolution of the cache contents over time. Cache blocks that have been evicted either due to CRPD or CPRO and must be reloaded from main memory are highlighted in red. The set of PCBs are highlighted in green.*

Initially, the cache is empty and with τ_3 being the first task to execute it loads all its ECBs into the cache. When τ_2 preempts τ_3 for the first time, it also loads its ECBs. Similarly, τ_2 is preempted by the highest priority task τ_1 at time 2. Note that ECBs of task τ_1 and UCBs/PCBs of task τ_2 are mapped to the same cache blocks, i.e., $\{7, 8, 9, 10\}$. Therefore, when τ_2 resumes its execution after the completion of the first job of τ_1 it needs to reload all its UCBs, (highlighted in red) as they



(a) Schedule maximizing CRPD during the response time of τ_3



(b) Schedule maximizing CPRO during the response time of τ_3

Fig. 1: Schedules maximizing τ_3's response time when $C_1 = 1$, $C_2 = 2$, $C_3 = 9$, $T_1 = 6$, $T_2 = 6$, $T_3 = 25$, $ECB_1 = \{7, 8, 9, 10\}$, $ECB_2 = \{7, 8, 9, 10\}$, $ECB_3 = \{1, 2, 3, 4, 5\}$, $UCB_2 = \{7, 8, 9, 10\}$, $PCB_2 = \{7, 8, 9, 10\}$ and $UCB_1 = UCB_3 = PCB_1 = PCB_3 = \emptyset$

were evicted by τ_1. These additional memory accesses will be accounted for as CRPD.

Since, the first job of τ_2 loads all τ_2's ECBs (PCBs and nPCBs) into the cache, subsequent jobs of τ_2 may have a lower memory demand due to the existence of PCBs in the cache, i.e., blocks $\{7, 8, 9, 10\}$. However, some of these PCBs may be evicted due to other task executions. The additional memory accesses required to reload evicted cache blocks are accounted for as CPRO. Such a situation where the CPRO is maximized is depicted in Fig. 1b.

Based on Fig. 1a, the total memory reload overhead μ_3 during τ_3's response time is equal to the time needed to reload 12 cache blocks (i.e., the number of red blocks).

Now, if we use the UCB-union multi-set and the CPRO multi-set approaches to calculate μ_3, we have the following.

$$\mu_3 \leq \gamma^{ucb-m}_{3,1} + \gamma^{ucb-m}_{3,2} + \rho^{mul}_{1,3} + \rho^{mul}_{2,3}$$

Since τ_2 is the only task with useful cache blocks ($UCB_2 = \{7, 8, 9, 10\}$), it is also the only task suffering from CRPD. Therefore, $\gamma^{ucb-m}_{3,2} = 0$. Using (3), we have (note that $E_1(R_3) = 3$, $E_1(R_2) = 1$, $E_2(R_3) = 3$, and $E_3(R_3) = 1$):

$$\begin{aligned} \gamma^{ucb-m}_{3,1} &= d_{mem} \times |\,(3 \times UCB_3 \cup 3 \times UCB_2) \cap (3 \times ECB_1)| \\ &= d_{mem} \times 12 \end{aligned}$$

Similarly, when calculating the CPRO we can see that the set of PCBs for all tasks except τ_2 is empty. Hence, the total CPRO during the response time of task τ_3 comes only from the evictions of PCBs of task τ_2. Assuming that the CPRO is calculated using (10) we have $\rho^{mul}_{1,3} = 0$ and

$$\rho^{mul}_{2,3} = d_{mem} \times |(2 \times PCB_2) \cap (4 \times ECB_3 \cup 3 \times ECB_1)\,| = d_{mem} \times 8$$

Adding CRPD and CPRO, it follows that the total memory reload overhead during the response time of τ_3 is upper-bounded by $d_{mem} \times 20$. Thus it appears that 20 cache blocks

need to be reloaded during the response time of τ_3. The reason for the overestimation is that the total CRPD is indeed upper-bounded by 12 cache blocks reloads (as shown in Fig. 1a) and the total CPRO is indeed upper-bounded by 8 cache blocks reloads (as shown on Fig. 1b), but both scheduling scenarios cannot happen at the same time. It is not possible for the three jobs of τ_1 to result in the group of 4 cache block reloads three times over due to preemptions (accounted for in $\gamma_{3,1}^{ucb-m}$) and two times over due to cache persistence overheads (accounted for in $\rho_{2,3}^{mul}$). This observation leads to the following lemma.

Lemma 1. Let us assume that the total CRPD during the response time of task τ_i is computed using (1) or (3) and that the total CPRO during τ_i's response time is computed with (8) or (10). Let $b_{k,\ell}$ be the ℓ^{th} cache block of a task $\tau_k \in \mathrm{hp}(i)$, i.e., $b_{k,\ell} \in ECB_k$. The eviction of $b_{k,\ell}$ will be accounted for in both the CRPD and CPRO, only if $b_{k,\ell}$ is a UCB and a PCB of τ_k, i.e., $b_{k,\ell} \in UCB_k \cap PCB_k$.

Proof. This claim follows directly from the fact that (1) and (3) account for the evictions of UCBs of tasks in $\mathrm{hep}(i)$. Therefore, the eviction of cache block $b_{k,\ell}$ will be considered in the CRPD calculation only if it is a UCB. Similarly, (8) and (10) account for the evictions of PCBs of tasks in $\mathrm{hp}(i)$. Hence, the eviction of cache block $b_{k,\ell}$ will be considered in the CPRO calculation only if it is a PCB. Therefore, the eviction of $b_{k,\ell}$ may be accounted for in both the CRPD and CPRO, only if $b_{k,\ell} \in UCB_k \cap PCB_k$. □

It can also be seen in Example 1 that for any task $\tau_k \in \mathrm{hp}(i)$ (e.g., τ_2) executing during the response time of a lower priority task τ_i (e.g., τ_3), only higher priority tasks than τ_k (e.g., τ_1) can participate in both the CRPD and CPRO of τ_k. This observation leads to the following lemma.

Lemma 2. For any task $\tau_k \in \mathrm{hp}(i)$ executing during the response time of a lower priority task τ_i, only the tasks in $\mathrm{hp}(k)$ can contribute to both the CRPD and CPRO of τ_k.

Proof. By Definition 6, all tasks in $\mathrm{hep}(i) \setminus \tau_k$ can contribute to the CPRO of τ_k during the response time of τ_i.

Let τ_ℓ be any task in $\mathrm{hep}(i) \setminus \tau_k$. Two cases must be considered:

1) If $\tau_\ell \in \mathrm{aff}(i,k)$ then τ_ℓ has a lower priority than τ_k. Therefore, τ_ℓ can never preempt τ_k and hence cannot contribute to τ_k's CRPD.
2) If $\tau_\ell \in \mathrm{hp}(k)$ then τ_ℓ has a higher priority than that of τ_k. Task τ_ℓ can therefore preempt τ_k and cause CRPD.

Hence, only tasks in $\mathrm{hp}(k)$ can contribute to both τ_k's CRPD and CPRO. □

V. Integrated CRPD-CPRO Analysis

In the existing literature, CRPD and CPRO are calculated independently of each other. As discussed in Section IV, this can lead to an overestimation of the total memory reload overhead. In this section, we present a novel approach to bound the total memory reload overhead during the response time of a task τ_i. This section builds upon the UCB-union and CPRO-union approaches for the calculation of CRPD and CPRO, respectively. In Section VI, we extend this analysis to consider the more precise, but also more complex, multi-set variants of the CPRD and CPRO calculation.

It follows from Lemma 1 that only the cache blocks in $\bigcup_{\forall \tau_j \in \mathrm{hp}(i)} (UCB_j \cap PCB_j)$ can have their evictions counted twice during the CRPD and CPRO calculations. This double counting can be removed either (i) during the CRPD calculation by not considering the evictions of PCBs in $\bigcup_{\forall \tau_j \in \mathrm{hp}(i)} (UCB_j \cap PCB_j)$, since their eviction will be accounted for in the CPRO; or, (ii) during the CPRO calculation by not considering the eviction of UCBs in $\bigcup_{\forall \tau_j \in \mathrm{hp}(i)} (UCB_j \cap PCB_j)$, since their eviction will be considered in the CRPD. In this section, we focus on the latter solution assuming that the CRPD is computed using the UCB-union approach (i.e., using (1)).

Lemma 3. Let Γ_i be an upper-bound on the total CRPD during the response time R_i of τ_i. Further assume that Γ_i is computed using the UCB-union approach, i.e., $\Gamma_i \stackrel{\text{def}}{=} \sum_{\forall \tau_j \in \mathrm{hp}(i)} \left\lceil \frac{R_i}{T_j} \right\rceil \gamma_{i,j}^{ucb}$. Let Δ_i be an upper-bound on the portion of the total memory reload overhead during τ_i's response time that is not accounted for in Γ_i, that is, $\Delta_i = \mu_i - \Gamma_i$, then we have $\Delta_i \leq \sum_{\forall \tau_j \in \mathrm{hp}(i)} \left(\left\lceil \frac{R_i}{T_j} \right\rceil - 1 \right) \times \delta_{j,i}$ where

$$\delta_{j,i} \stackrel{\text{def}}{=} d_{mem} \times \left| PCB_j \bigcap \left(\left(\bigcup_{\forall \tau_k \in \mathrm{aff}(i,j)} ECB_k \right) \bigcup \left(\bigcup_{\forall \tau_k \in \mathrm{hp}(j)} ECB_k \setminus (UCB_j \cap PCB_j) \right) \right) \right| \quad (15)$$

Proof. It was proven in [23] that Γ_i upper-bounds the total CRPD during τ_i's response time. Therefore, the portion of the total memory reload overhead μ_i that is not accounted for in Γ_i is a subset of the total CPRO during τ_i's response time. Similar to the calculation of the total CPRO, at most $\left(\left\lceil \frac{R_i}{T_j} \right\rceil - 1 \right)$ jobs of each higher priority task τ_j can suffer memory reload overhead $\delta_{j,i}$ not yet accounted for in Γ_i. Since the total CPRO is an upper-bound on Δ_i, using (8) and (9) we have $\Delta_i \leq \sum_{\forall \tau_j \in \mathrm{hp}(i)} \left(\left\lceil \frac{R_i}{T_j} \right\rceil - 1 \right) \times \delta_{j,i}$ with $\delta_{j,i} \leq \rho'_{j,i}$. We now prove the validity of $\delta_{j,i}$.

Since a fixed-priority scheduling algorithm is used, only tasks with priorities higher than or equal to the priority of τ_i (i.e., tasks in $\mathrm{hep}(i)$) can execute during the response time of τ_i. Therefore, any task $\tau_k \in \mathrm{hep}(i) \setminus \tau_j$ can execute between two subsequent jobs of another task τ_j and hence participate in τ_j's CPRO by evicting some or all its PCBs. Let τ_k be any task in $\mathrm{hep}(i) \setminus \tau_j$. Two cases need to be considered (note that $\mathrm{hep}(i) \setminus \tau_j = \mathrm{aff}(i,j) \cup \mathrm{hp}(j)$).

1) $\tau_k \in \mathrm{aff}(i,j)$. Since τ_k has a lower priority than τ_j it cannot preempt τ_j, and hence τ_k does not contribute to the CRPD of τ_j. Therefore, the memory reload overhead generated by τ_k on τ_j is not part of Γ_i and must be entirely accounted for in $\delta_{i,j}$. This worst-case interference of τ_k on τ_j is maximized when τ_k loads all its cache blocks (i.e., ECB_k).
2) If $\tau_k \in \mathrm{hp}(j)$ then, by Lemma 2, τ_k may contribute to both the CRPD and CPRO of τ_j. As stated in Lemma 1, the evictions of cache blocks of τ_j in $UCB_j \cap PCB_j$

were already considered in Γ_i. Therefore, the number of cache block evictions caused by τ_k on τ_j that were not accounted for in Γ_i is maximized when τ_k loads all the cache blocks in $ECB_k \setminus (UCB_j \cap PCB_j)$.

From 1. and 2., the biggest set $\mathcal{S}_{j,i}$ of cache blocks that can be loaded by tasks in $\mathrm{hep}(i) \setminus \tau_j$ and were not yet considered in Γ_i is given by:

$$\mathcal{S}_{j,i} = \left(\bigcup_{\forall \tau_k \in \mathrm{aff}(i,j)} ECB_k \right) \bigcup \left(\bigcup_{\forall \tau_l \in \mathrm{hp}(j)} ECB_l \setminus (UCB_j \cap PCB_j) \right)$$

The set of PCBs that must be reloaded by τ_j at each job execution is thus upper-bounded by the intersection between τ_j's PCBs (i.e., PCB_j) and the set $\mathcal{S}_{j,i}$ derived above. Since each cache block reload takes at most d_{mem} time units, the time $\delta_{j,i}$ spent by τ_j at each job execution to reload evicted PCBs that were not yet considered in Γ_i is bounded by (15). □

As a corollary of Lemma 3, we can upper-bound the total memory reload overhead μ_i as stated in the following theorem:

Theorem 1. The total memory reload overhead μ_i during τ_i's response time is upper-bounded by

$$\sum_{\forall \tau_j \in \mathrm{hp}(i)} \left(\left(\left\lceil \frac{R_i}{T_j} \right\rceil \times \gamma_{i,j}^{ucb} \right) + \left(\left\lceil \frac{R_i}{T_j} \right\rceil - 1 \right) \times \delta_{j,i} \right) \tag{16}$$

Proof. Follows from Lemma 3 since $\mu_i = \Delta_i + \Gamma_i$. □

This directly leads to the following theorem:

Theorem 2. The WCRT of τ_i is upper-bounded by the smallest positive solution to

$$R_i = C_i + \sum_{\forall j \in \mathrm{hp}(i)} \left(\gamma_{i,j} + \min \left\{ \left\lceil \frac{R_i}{T_j} \right\rceil C_j \; ; \left\lceil \frac{R_i}{T_j} \right\rceil PD_j + \hat{MD}_j(R_i) + \hat{\delta}_{j,i} \right\} \right) \tag{17}$$

where

$$\hat{\delta}_{j,i} \stackrel{\mathrm{def}}{=} \left(\left\lceil \frac{R_i}{T_j} \right\rceil - 1 \right) \delta_{j,i} \tag{18}$$

and $\gamma_{i,j}$ is given by $\left\lceil \frac{R_i}{T_j} \right\rceil \gamma_{i,j}^{ucb}$ for UCB-Union.

Proof. By Theorem 1 and substituting $\hat{\delta}_{j,i}$ for $\rho_{j,i}$ in (13) □

Since, $\delta_{j,i}$ calculated using (15) is always less than or equal to $\rho'_{j,i}$ calculated using (9), the resulting WCRT obtained using (17) is always less than or equal to the WCRT obtained using (13) when $\gamma_{i,j}$ is computed using the UCB-union approach. In other words, the integrated approach to CRPD and CPRO analysis given by Theorem 2 *dominates* the simple combination of the UCB-union and CPRO-union approaches.

Example 2. *We now compute the total memory reload overhead of task τ_3 in Example 1 using the results derived in Theorem 1.*

Note that the UCB-union (1) *and the UCB-union multi-set* (3) *approaches would give exactly the same values for the total CRPD. Therefore, the total CRPD is upper-bounded by* $d_{mem} \times 12$.

The set of PCBs for all tasks except τ_2 is empty. Therefore, based on (15)*, we have $\delta_{1,3} = 0$ and*

$$\begin{aligned} \delta_{2,3} &= d_{mem} \times |PCB_2 \cap (ECB_3 \cup (ECB_1 \setminus (UCB_2 \cap PCB_2)))| \\ &= d_{mem} \times |\{7,8,9,10\} \cap (\{7,8,9,10\} \setminus \{7,8,9,10\})| = 0 \end{aligned}$$

According to Theorem 1, μ_3 is thus upper-bounded by $(12 \times d_{mem})$*, which is in this case the exact overhead experienced during the response time of τ_3 as illustrated in Fig. 1a.*

VI. Multi-set Approach to Integrated CRPD-CPRO analysis

In this section, we improve the analysis presented in Section V by building upon the UCB-union multi-set (3) and CPRO-union multi-set (10) analyses that were shown to dominate the UCB-union and CPRO-union approaches.

While the UCB-union approach assumes that every job of a task $\tau_k \in \mathrm{hp}(i)$ executing during the response time of τ_i can contribute to the total CRPD, the UCB-union multi-set approach (3) considers that only a subset of τ_k's jobs actually contribute to the preemption overhead. Hence, we must also differentiate between jobs that are considered in the CRPD and those that are not, when computing the portion of the total memory reload overhead μ_i that is not yet accounted for in the total CRPD.

Example 3. *The example task set in Fig. 2 has three tasks τ_1, τ_2 and τ_3 with priorities assigned in numerical order such that τ_1 has the highest priority. We want to analyze the total memory reload overhead μ_3 during the response time of τ_3. Task τ_2 is the only task with $UCB_2 \cap PCB_2 \neq \emptyset$. The sets of UCBs and PCBs of τ_1 and τ_3 are empty. Therefore, τ_2 is the only task that may suffer CRPD and CPRO. The total memory reload overhead μ_3 is thus bounded by the sum of the CRPD and CPRO suffered by τ_2 during the response time of τ_3.*

By Lemma 2, τ_1 is the only task that can contribute to both τ_2's CRPD and CPRO. Since τ_1 can preempt each job of τ_2 at most once (i.e., $E_1(R_2) = 1$), and because τ_2 releases three jobs during τ_3's response time (i.e., $E_2(R_3) = 3$), at most three jobs of τ_1 are preempting jobs of τ_2 during the response time of τ_3, i.e., $E_1(R_2)E_2(R_3) = 3$. Therefore, at most three jobs of τ_1 may be contributing to both τ_2's CRPD and CPRO during τ_3's response time. The two remaining jobs of τ_1 can only execute between jobs of τ_2, and hence contribute only to τ_2's CPRO.

To calculate the CPRO that any task $\tau_j \in \mathrm{hp}(i)$ can suffer during the response time of τ_i, taking into consideration what has already been accounted for in the CRPD cost, we first analyze the impact of each task in $\mathrm{hep}(i) \setminus \tau_j$ on the CPRO of τ_j. We characterize the maximum number of times a task $\tau_k \in \mathrm{hep}(i) \setminus \tau_j$ can execute between successive jobs of τ_j. To do so, we separately analyze the tasks in $\mathrm{aff}(i,j)$ (Lemma 4) and the tasks in $\mathrm{hp}(j)$ (Lemma 5). We then identify how many jobs of each task contribute only to the CPRO of τ_j and how many jobs contribute to both the CRPD and the CPRO of τ_j (Lemma 6). We then make use of this information to derive a multi-set formulation (Lemma 7) that calculates the additional CPRO of a task $\tau_j \in \mathrm{hp}(i)$ that is not already accounted for in the CRPD cost computed with (3).

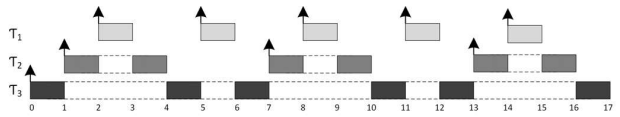


Fig. 2: Illustrating the pessimism associated with the separate UCB-union multi-set and CPRO multi-set analysis using the task set $\{\tau_1, \tau_2, \tau_3\}$ with $C_1 = 1$, $C_2 = 2$, $C_3 = 6$, $T_1 = 3$, $T_2 = 6$ and $T_3 = 20$.

Lemma 4 (from [21])**.** The maximum number of times a task $\tau_k \in \text{aff}(i,j)$ can execute between jobs of τ_j released during τ_i's response time is upper-bounded by $(E_j(R_k)+1) \times E_k(R_i)$.

Proof. Lemma 4 in [21]. □

Lemma 5 (from [21])**.** The maximum number of times a task $\tau_k \in \text{hp}(j)$ can execute between successive jobs of τ_j released during τ_i's response time is upper bounded by $E_k(R_i)$.

Proof. Lemma 3 in [21]. □

Example 3 shows that not all of the jobs released by a higher priority task $\tau_l \in \text{hp}(j)$ (e.g., τ_1 in Fig. 2) during the response time of a lower priority task τ_i (e.g., τ_3 in Fig. 2) can preempt τ_j (e.g., τ_2 in Fig. 2). The jobs that do not preempt cannot contribute to both the CRPD and the CPRO of τ_j. This observation leads to the following Lemma:

Lemma 6. For a task $\tau_j \in \text{hp}(i)$ executing during the response time of τ_i, the number of jobs of any higher priority task $\tau_l \in \text{hp}(j)$ that are already accounted for in the CRPD $\gamma_{i,j}^{ucb-m}$ is given by $N_{l,j}^{double}(R_i) = \min\{E_l(R_i)\ ;\ E_l(R_j)E_j(R_i)\}$.

Proof. The CRPD $\gamma_{i,j}^{ucb-m}$ in (3) is composed of the intersection of the two multi-sets $M_{i,j}^{ucb}$ and $M_{i,j}^{ecb}$.

1) The calculation of $M_{i,j}^{ecb}$ (5) assumes that a task $\tau_l \in hp(j)$ can release at most $E_l(R_i)$ jobs during the response time R_i of τ_i. Therefore, at most $E_l(R_i)$ jobs of τ_l preempting τ_j are accounted for in the calculation of $\gamma_{i,j}^{ucb-m}$ in (3).
2) The calculation of $M_{i,j}^{ucb}$ (4) assumes that for any task $\tau_j \in aff(i,j)$, $E_l(R_j)E_j(R_i)$ is an upper bound on the number of times τ_j can be preempted by τ_l during τ_i's response time. Therefore, at most $E_l(R_j)E_j(R_i)$ jobs of τ_l are accounted for in $\gamma_{i,j}^{ucb-m}$ (3).

It follows that the number of jobs of τ_l accounted for in $\gamma_{i,j}^{ucb-m}$ is given by $N_{l,j}^{double}(R_i)$. □

Using Lemmas 2, 4–6 we derive an upper bound on the CPRO any task $\tau_j \in hp(i)$ can suffer during τ_i's response time, discounting what has already been taken into account in the CRPD cost $\gamma_{i,j}^{ucb-m}$. This upper bound is denoted by $\delta_{i,j}^{mul}$.

Lemma 7. Let Γ_i^m be an upper-bound on the total CRPD during the response time R_i of τ_i. Further assume that Γ_i^m is computed using the UCB-union multi-set approach, i.e., $\Gamma_i^m = \sum_{\forall \tau_j \in hp(i)} \gamma_{i,j}^{ucb-m}$. Let Δ_i^m be an upper-bound on the portion of the total memory reload overhead that was not accounted for in Γ_i^m, that is, $\Delta_i^m = \mu_i - \Gamma_i^m$, then:

$$\Delta_i^m \leq \sum_{\forall \tau_j \in hp(i)} \delta_{i,j}^{mul} \tag{19}$$

where

$$\delta_{i,j}^{mul} \stackrel{\text{def}}{=} d_{mem} \times \left| M_{j,i}^{ecb} \cap M_{j,i}^{pcb} \right| \tag{20}$$

where $M_{j,i}^{ecb}$ and $M_{j,i}^{pcb}$ are multi-sets defined as

$$M_{j,i}^{pcb} = \bigcup_{E_j(R_i)-1} PCB_j \tag{21}$$

$$M_{j,i}^{ecb} = M_{j,i}^{ecb-aff} \cup M_{j,i}^{hp-int} \tag{22}$$

$$\text{with } M_{j,i}^{ecb-aff} = \bigcup_{\forall k \in \text{aff}(i,j)} \left(\bigcup_{(E_j(R_k)+1)E_k(R_i)} ECB_k \right) \tag{23}$$

$$M_{j,i}^{hp-int} = \bigcup_{\forall l \in \text{hp}(j)} \left(\left(\bigcup_{E_l(R_i) - N_{l,j}^{double}(R_i)} ECB_l \right) \bigcup \left(\bigcup_{N_{l,j}^{double}(R_i)} ECB_l \setminus \left(UCB_j \cap PCB_j \right) \right) \right) \tag{24}$$

Proof. Since Γ_i^m upper-bounds the total CRPD during τ_i's response time calculated using (3), the portion of μ_i that is not accounted for in Γ_i^m is a subset of the total CPRO during τ_i's response time that is,

$$\Delta_i^m \leq \sum_{\forall \tau_j \in \text{hp}(i)} \delta_{j,i}^{mul}$$

where $\delta_{j,i}^{mul} \leq \rho_{j,i}^{mul}$.

We prove the validity of $\delta_{j,i}^{mul}$ below.

1. Since τ_j can release at most $\left\lceil \frac{t}{T_j} \right\rceil$ jobs in a time window of length t, the PCBs of τ_j can be evicted at most $\left(\left\lceil \frac{t}{T_j} \right\rceil - 1 \right)$ times within the time window of length t, contributing to CPRO[4]. Therefore, the largest set of PCBs of τ_j that can be evicted during the response time of τ_i is upper bounded by the multi-set $M_{j,i}^{pcb} = \bigcup_{E_j(R_i)-1} PCB_j$ given in (21).

2. By Lemma 4, the maximum number of times a task $\tau_k \in \text{aff}(i,j)$ can execute between two successive jobs of τ_j during the response time of τ_i is upper bounded by $(E_j(R_k)+1) \times E_k(R_i)$. Hence, the largest set of ECBs that can be loaded by τ_k between successive jobs of τ_j during the response time of τ_i is given by $\bigcup_{(E_j(R_k)+1)E_k(R_i)} ECB_k$. Therefore the largest set of ECBs loaded by the tasks in $\text{aff}(i,j)$ between successive executions of τ_j is upper bounded by $M_{j,i}^{ecb-aff} = \bigcup_{\forall k \in \text{aff}(i,j)} \left(\bigcup_{(E_j(R_k)+1)E_k(R_i)} ECB_k \right)$ given in (23).

3. By Lemma 5, the maximum number of times a task $\tau_l \in \text{hp}(j)$ can execute between two successive jobs of τ_j during the response time of τ_i is upper bounded by $E_l(R_i)$. Hence, the largest set of ECBs that can be loaded by τ_l and interfere with the PCBs of τ_j is given by $\bigcup_{E_l(R_i)} ECB_l$. However, by Lemma 2, as $\tau_l \in \text{hp}(j)$ it can contribute to both the CRPD and CPRO of τ_j during the response time of τ_i. Further, by Lemma 6, the number of jobs of τ_l that were already considered

[4]Recall from (7) that all PCBs are assumed to be loaded once anyway.

in the CRPD of τ_j is equal to $N_{l,j}^{double}(R_i)$. Therefore, instead of assuming that all jobs released by $\tau_l \in \text{hp}(j)$ during the response time of τ_i contribute to $\delta_{j,i}^{mul}$, the multi-set $M_{j,i}^{hp-int}$ separately categorizes the impact of jobs of τ_l that can/cannot be contributing to both the CRPD and CPRO of τ_j during the response time of τ_i.

3.1 Since $N_{l,j}^{double}(R_i)$ is the number of jobs of τ_l that were already considered in the CRPD of τ_j, then $E_l(R_i) - N_{l,j}^{double}(R_i)$ jobs of τ_l only contribute to the CPRO of τ_j. The memory reload overhead generated by these $E_l(R_i) - N_{l,j}^{double}(R_i)$ jobs of τ_l on τ_j is not part of Γ_i^m and must therefore be entirely accounted for in $\delta_{j,i}^{mul}$. The worst-case interference of all these jobs is maximized when every job of τ_l loads all its cache blocks (i.e., ECB_l). Hence, the worse-case impact that these jobs of τ_l can have on the τ_j's CPRO is bounded by the multi-set $\bigcup_{E_l(R_i)-N_{l,j}^{double}(R_i)} ECB_l$ given in the first term of (24).

3.2 For all jobs of τ_l that can contribute to both the CRPD and CPRO of τ_j, i.e., $N_{l,j}^{double}(R_i)$, then as stated in Lemma 1, the evictions of caches blocks of τ_j in $UCB_j \cap PCB_j$ were already considered in Γ_i^m. Therefore, the number of cache block evictions caused by these $N_{l,j}^{double}(R_i)$ jobs of τ_l on τ_j that were not accounted for in Γ_i^m is maximized when each job loads all the cache blocks in $ECB_l \setminus (UCB_j \cap PCB_j)$. Hence, the worse-case additional impact of all jobs of τ_l that contribute to both the CRPD and CPRO of τ_j is bounded by the multi-set, $\bigcup_{N_{l,j}^{double}(R_i)} ECB_l \setminus (UCB_j \cap PCB_j)$ given by the second term of (24).

Therefore, by 2. and 3. above, the largest set of ECBs that can interfere with the PCBs of τ_j during the response time of τ_i is upper bounded by $M_{j,i}^{ecb} = M_{j,i}^{ecb-aff} \cup M_{j,i}^{hp-int}$ given by (22). Hence, the largest set of PCBs of τ_j that can be evicted by the tasks in $\text{hep}(i) \setminus \tau_j$ within the response time of τ_i with evictions not already considered in Γ_i^m, is upper bounded by the intersection of $M_{j,i}^{pcb}$ with $M_{j,i}^{ecb}$. Since reloading a cache block takes at most d_{mem} time units, an upper bound on the total CPRO $\delta_{j,i}^{mul}$, not already included in the CRPD, is given by $d_{mem} \times \left| M_{j,i}^{ecb} \cap M_{j,i}^{pcb} \right|$ in (20). □

As a corollary of Lemma 7, we can upper-bound the total memory reload overhead μ_i as stated in the following theorem:

Theorem 3. The total memory reload overhead μ_i during τ_i's response time is upper-bounded by

$$\sum_{\forall \tau_j \in \text{hp}(i)} \left(\gamma_{i,j}^{ucb-m} + \delta_{j,i}^{mul} \right) \tag{25}$$

Proof. Follows from Lemma 7 since $\mu_i = \Delta_i^m + \Gamma_i^m$. □

This leads directly to the following theorem.

Theorem 4. The WCRT of τ_i is upper-bounded by the smallest positive solution to

$$R_i = C_i + \sum_{\forall j \in \text{hp}(i)} \left(\gamma_{i,j} + \min \left\{ \left\lceil \frac{R_i}{T_j} \right\rceil C_j \; ; \left\lceil \frac{R_i}{T_j} \right\rceil PD_j + \hat{MD}_j(R_i) + \delta_{j,i}^{mul} \right\} \right) \tag{26}$$

where $\gamma_{i,j}$ is given by $\gamma_{i,j}^{ucb-m}$ for UCB-Union Multi-set.

Proof. By Theorem 3 and substituting $\delta_{j,i}^{mul}$ for $\rho_{j,i}$ in (13) □

Since, $\delta_{j,i}^{mul}$ calculated using (20) is always less than or equal to $\rho_{j,i}^{mul}$ calculated using (10), the resulting WCRT obtained using (26) is always less than or equal to the WCRT obtained using (13) when $\gamma_{i,j}$ is computed using the UCB-Union Multiset approach. In other words, the integrated multiset approach to CRPD and CPRO analysis given by Theorem 4 *dominates* the seperate combination of the UCB-Union Multiset and CPRO-Multiset approaches.

VII. Experimental Evaluation

In this section, we evaluate how the integrated CRPD-CPRO analyses perform in terms of schedulability and if it is beneficial to use the integrated approaches in comparison to the state-of-the art approaches that separately account for CRPD and CPRO. We performed experiments using the Mälardalen benchmark suite [13] and a set of sequential benchmarks from TACLEBench [12] with various parameter settings.

The tasks parameters C_i, PD_i, MD_i, MD_i^r along with the sets of UCB, ECB, PCB and $nPCB$ were extracted using Heptane, a static WCET analysis tool[5], as presented in [21]. The target architecture was MIPS R2000/R3000 assuming a cache line size of 32 Bytes, a cache size of 8kB and a block reload time $d_{mem} = 8\mu$s. The memory footprint of each task was upper bounded by 256 cache sets (i.e., 100% of the cache size). Table I (See Appendix A) shows the resulting task parameters for the benchmarks used during the experiments.

The other task set parameters were randomly generated as follows. The default number of tasks was 10 with task utilizations generated using UUnifast [8]. Each task was randomly assigned the values C_i, PD_i, MD_i, MD_i^r, UCB, ECB, PCB and $nPCB$ of one of the analyzed benchmarks. Task periods were set such that $T_i = C_i/U_i$. Task deadlines were implicit and priorities were assigned in deadline monotonic order.

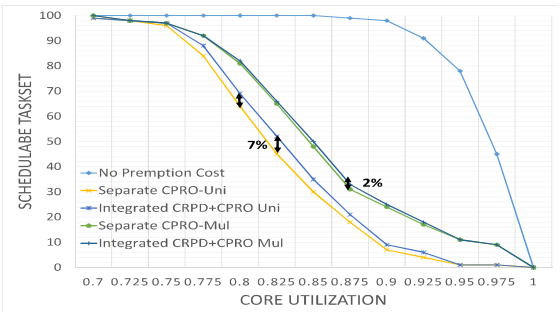
We conducted experiments varying the total task utilization, cache size, block reload time and task memory footprint (for the experiment on task memory footprint See Appendix B). A WCRT based schedulability analysis is performed using the same task sets for all approaches.

1) Core Utilization.: In this experiment, we randomly generated 100 task sets (with 10 tasks each) with a total utilizations varied from 0.025 to 1 in steps of 0.025. The experiment was first performed using the Mälardalen benchmarks and then using TACLEBench's sequential benchmarks.

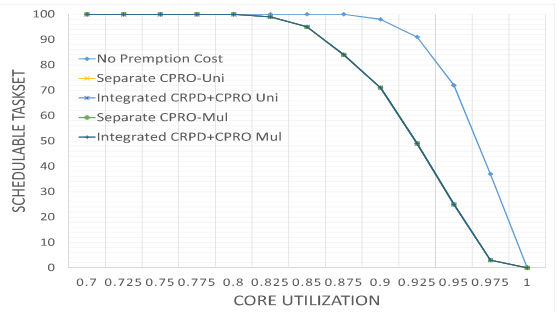
Fig. 3(a) and (b) show the number of task sets that were deemed schedulable by the different analyses. Both plots also show the number of task sets that were deemed schedulable without considering any CRPD or CPRO. We only show cropped versions of the plots starting from a utilization of 0.7. All approaches produce identical results below this point.

Observation 1. *Integrated CRPD-CPRO analyses out-perform the state-of-the-art CPRO-union and multi-set approaches that separately account for CRPD and CPRO.*

[5]See https://team.inria.fr/alf/software/heptane/



(a) Schedulability ratio using Mälardalen benchmarks



(b) Schedulability ratio using TACLEBench

Fig. 3: Schedulability ratio with respect to total core utilization

Fig. 3a shows that when using Mälardalen benchmarks the integrated schedulability tests accepted more task sets in comparison to tests using separate CRPD and CPRO analyses. The difference between the integrated CRPD-CPRO union approach and the separate CPRO-union approach is more significant in comparison to their multi-set counterparts. The schedulability ratio is increased by up to 7%. However, as the separate CPRO multi-set approach is already much more precise the difference between the integrated CRPD-CPRO multi-set and the separate CPRO multi-set approach is only around 2%. Nevertheless, we can observe that there are task sets that were schedulable using the integrated CRPD-CPRO approaches but not with the separate CPRO-union and multi-set approaches, therefore in this case the integrated CRPD-CPRO approaches outperforms the separate CPRO-union and multi-set approaches. Note also that the schedulability gain slightly increases when the cache size increases. For instance, when there are 512 cache sets the gain is 8% for the integrated CRPD-CPRO union analysis, and 4% for the multi-set analysis.

Observation 2. *For benchmarks (i.e., tasks) with large memory footprints, there is no gain when integrating the CRPD-CPRO calculation.*

As shown in Fig. 3b, the integrated CRPD-CPRO analyses do not improve over the state-of-the-art for the TACLEBench benchmarks. In fact, the same number of task sets were schedulable using all the approaches. The difference with Fig. 3a can be understood as follows. Mälardalen benchmarks consist of both light and heavy tasks (see Table I in Appendix A) whereas the majority of tasks in TACLEBench have large memory footprints using the entire cache. Therefore, almost all tasks overlap in the cache, in which case the tasks with lower priority than a task τ_j (i.e., the tasks in $\mathrm{aff}(i,j)$) evict the same cache blocks of τ_j as the tasks with higher priority (i.e, in $\mathrm{hp}(j)$). Hence, according to (15) and (20), integrating the CRPD and CPRO analyses does not provide any gain.

From here on, we only show experimental results obtained using the Mälardalen benchmarks.

2) Cache size: In FPPS, the cache size can have a significant impact on the overall schedulability of the system. In this experiment, we vary the total number of cache sets from 32 to 512. Fig. 4a shows the resulting weighted schedulability [7] of each approach plotted against the total cache size[6].

Observation 3. *The integrated CRPD-CPRO analyses tend to outperform the separate analyses when the cache size increases.*

We can see from the plot in Fig. 4a, that initially increasing the cache size decreases the schedulability of all the approaches (i.e., from 32 to 128). This is mainly because most tasks use between 32 to 128 cache sets. Hence, increasing the cache size in this interval increases the number of ECBs and UCBs of tasks resulting in higher values of CRPD. Most of the cache blocks are evicted (and reloaded) for every task execution and hence we observe that all the approaches produce similar results. However, a further increase in cache size (i.e., from 128 to 512) means more tasks fit in the cache with less conflicts between tasks. Therefore, we see an increase in schedulability of all approaches. Also increasing the cache size results in increasing the number of PCBs of tasks, so the overlap between UCBs and PCBs of tasks also increase. Hence, we observe that with an increase in cache size from 128 to 512, the integrated CRPD-CPRO union and multi-set approach tend to perform better than the state-of-the-art approaches.

3) Block Reload Time (d_{mem}): In this experiment, we analyze the impact of block reload time d_{mem} on the performance of all the approaches by varying it between 2μs to 20μs, with all other parameters set to default values. Fig. 4b shows the resulting weighted schedulability.

Observation 4. *For very low or very high values of block reload time d_{mem}, the integrated and separate CRPD-CPRO analyses produce similar results.*

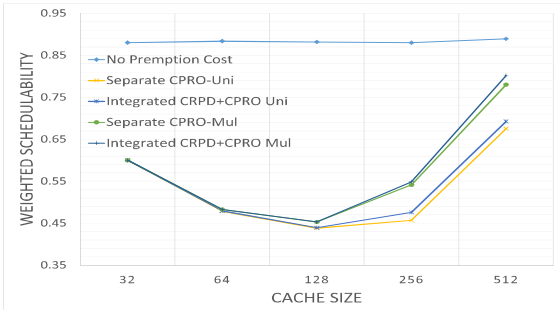
For smaller values of d_{mem} (i.e., between 2μs and 4μs) the impact of CRPD and CPRO on the schedulability of tasks is minimal. This means that similar results are achieved for integrated and separate union and multi-set approaches. Similarly, for higher values of d_{mem} (i.e., $d_{mem} > 15\mu$s), the CRPD becomes very high and thus negates any gain in schedulability resulting from the reduction of the CPRO cost in the integrated analysis. In contrast, for values of d_{mem} between 8μs to 12μs the impact of the overlap between CRPD and CPRO is visible.

4) Task Priority and Memory footprint: An additional experiment showing the impact of the highest priority task's memory footprint on the gain that can be achieved with the integrated analysis is provided in Appendix B.

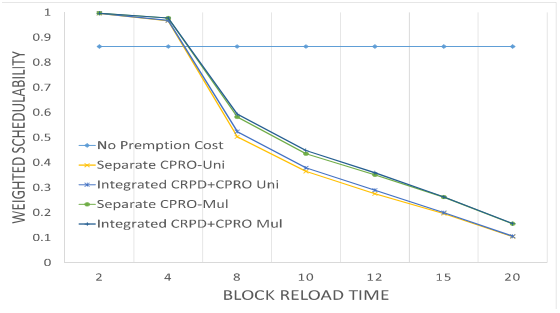
VIII. Conclusion and Future Work

In this paper we answer two questions: (1) Is it beneficial to integrate the calculation of CRPD and CPRO? and (2) when

[6]When calculating weighted schedulability we only consider task set utilizations between 0.6 to 1 since for lower utilizations, all task sets are schedulable.



(a) Varying cache size



(b) Varying block reload time d_{mem}

Fig. 4: Weighted schedulability measure by varying cache utilization, block reload time d_{mem} and cache size

and to what extent can we gain in terms of schedulability by integrating the calculation of CRPD and CPRO? Our experimental evaluation, as well as theoretical dominance results, showed that integrated CRPD-CPRO analysis can, in some cases, increase the schedulability ratio by 2% to 7% by providing a tighter calculation of total memory reload overheads compared to state-of-the-art approaches. However, as pointed out using a set of observations in the experimental evaluation the gains obtained using the integrated CRPD-CPRO analysis are dependent on certain system configurations and parameter values. The average gains in terms of schedulability resulting from the integration of CRPD-CPRO calculations may not be large; however, it is important to note that nevertheless, the integrated approaches dominate the state-of-the-art approaches and this dominance is obtained with no increase in complexity, or need for extra information. Therefore, it is indeed beneficial to integrate the calculation of CRPD and CPRO.

As future work, we aim to extend the integrated CRPD-CPRO analysis to set-associative LRU caches by adapting the calculation of CPRO using a similar approach to that presented in [3] for CRPD. Further research directions include adapting ECB-union [3] CRPD analysis and exploring the effect that the memory layout has on the integrated CRPD-CPRO analysis.

Acknowledgments. This paper is supported by NWO Veni Project, "The time is now: Timing Verification for Safety-Critical Multi-Cores"and by the ESPRC grant, MCCps (EP/K011626/1), and also by the Inria International Chair program. EPSRC Research Data Management: No new primary data was created during this study. This work was partially supported by National Funds through FCT (Portuguese Foundation for Science and Technology) and co-financed by ERDF (European Regional Development Fund) under the Portugal2020 Program, within the CISTER Research Unit (CEC/04234); by FCT and the ESF (European Social Fund) through POPH (Portuguese Human Potential Operational Program), under PhD grant SFRH/BD/119150/2016.

References

[1] S. Altmeyer, R. I. Davis, L. Indrusiak, C. Maiza, V. Nelis, and J. Reineke. A generic and compositional framework for multicore response time analysis. In *RTNS '15*, pages 129–138, 2015.

[2] S. Altmeyer, R. I. Davis, and C. Maiza. Cache related pre-emption delay aware response time analysis for fixed priority pre-emptive systems. In *RTSS'11*, pages 261–271, 2011.

[3] S. Altmeyer, R. I. Davis, and C. Maiza. Improved cache related pre-emption delay aware response time analysis for fixed priority pre-emptive systems. *Real-Time Systems*, 48(5):499–526, 2012.

[4] S. Altmeyer, R. Douma, W. Lunniss, and R.I. Davis. Evaluation of cache partitioning for hard real-time systems. In *ECRTS'14*, pages 15–26, 2014.

[5] S. Altmeyer, R. Douma, W. Lunniss, and R.I. Davis. On the effectiveness of cache partitioning in hard real-time systems. *Real-Time Systems*, pages 1–46, Jan 2016.

[6] S. Altmeyer and C. Maiza. Cache-related preemption delay via useful cache blocks: Survey and redefinition. volume 57, pages 707–719. Elsevier, 2011.

[7] A. Bastoni, B. Brandenburg, and J. Anderson. Cache-related preemption and migration delays: Empirical approximation and impact on schedulability. *OSPERT'10*, pages 33–44, 2010.

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

[9] A. Burns. Preemptive priority based scheduling: An appropriate engineering approach. In S.H. Son, editor, *Advances in Real-Time Systems*, pages 225–248. Prentice-Hall, 1994.

[10] J. 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 *RTAS'96.*, pages 204–212, 1996.

[11] C. Cullmann. Cache persistence analysis: Theory and practice. *ACM Trans. Embed. Comput. Syst.*, 12(1s):40:1–40:25, March 2013.

[12] Heiko F. et al. TACLeBench: A benchmark collection to support worst-case execution time research. In Martin Schoeberl, editor, *WCET 2016*, volume 55 of *OpenAccess Series in Informatics*, pages 2:1–2:10. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2016.

[13] J. Gustafsson, A. Betts, A. Ermedahl, and B. Lisper. The Mälardalen WCET benchmarks: Past, present and future. In *OASIcs-OpenAccess Series in Informatics*, volume 15. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2010.

[14] S. Hahn, J. Reineke, and R. Wilhelm. Towards compositionality in execution time analysis–definition and challenges. In *CRTS'13*, 2013.

[15] D.I. Katcher, H. Arakawa, and J.K. Strosnider. Engineering and analysis of fixed priority schedulers. *IEEE Trans. Softw. Eng.*, 19, 1993.

[16] C. G. Lee, J. Hahn, 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. *Computers, IEEE Transactions on*, 47(6):700–713, 1998.

[17] W. Lunniss, S. Altmeyer, and R. I. Davis. A comparison between fixed priority and edf scheduling accounting for cache related pre-emption delays. *Leibniz Transactions on Embedded Systems*, 1(1), 2014.

[18] W. Lunniss, S. Altmeyer, G. Lipari, and R. I. Davis. Accounting for cache related pre-emption delays in hierarchical scheduling. In *RTNS'14*, pages 183–192, 2014.

[19] W. Lunniss, S. Altmeyer, G. Lipari, and R. I. Davis. Cache related pre-emption delays in hierarchical scheduling. *Real-Time Systems*, 52(2):201–238, 2016.

[20] W. Lunniss, R.I. Davis, C. Maiza, and S. Altmeyer. Integrating cache related pre-emption delay analysis into edf scheduling. In *RTAS'13*, 2013.

[21] S. A. Rashid, G. Nelissen, D. Hardy, B. Akesson, I. Puaut, and E. Tovar. Cache-persistence-aware response-time analysis for fixed-priority preemptive systems. In *ECRTS'16*, pages 262–272, 2016.

[22] J. Staschulat, S. Schliecker, and R. Ernst. Scheduling analysis of real-time systems with precise modeling of cache related preemption delay. In *ECRTS'05*, pages 41–48, 2005.

[23] Y. Tan and V. Mooney. Timing analysis for preemptive multitasking real-time systems with caches. *ACM TECS*, 6(1):7, 2007.

[24] H. Tomiyama and N. D. Dutt. Program path analysis to bound cache-related preemption delay in preemptive real-time systems. In *CODES'00*, pages 67–71, 2000.

APPENDIX A
BENCHMARK PARAMETERS

TABLE I: Benchmark parameters used in the experiments

Name	C_i	PD_i	MD_i	$MD^r{}_i$	ECB_i	PCB_i	UCB_i	$nPCB_i$	Benchmark Type
lcdnum	3440	984	2740	192	20	20	20	0	Mälardalen
bs	1399	203	1223	34	11	11	10	0	Mälardalen
fibcall	1585	785	886	89	8	8	7	0	Mälardalen
bsort100	712289	710289	90893	88907	20	20	18	0	Mälardalen
select	17138	11158	7858	1394	60	60	60	0	Mälardalen
fir	8407	6112	3076	792	22	22	20	0	Mälardalen
sqrt	5667	2770	3242	362	26	26	25	0	Mälardalen
ns	30149	28149	6172	4186	20	20	19	0	Mälardalen
jfdctint	17347	7747	10473	965	96	96	96	0	Mälardalen
matmult	429286	426086	48560	45384	28	28	27	0	Mälardalen
expint	59446	57046	13586	11102	31	31	29	0	Mälardalen
insertsort	7574	5974	2343	752	16	16	10	0	Mälardalen
ludcmp	37335	27036	13757	3545	98	98	98	0	Mälardalen
cnt	10090	7191	3818	933	27	27	26	0	Mälardalen
prime	25891	23791	4246	2152	17	17	16	0	Mälardalen
minmax	2522	122	2400	0	22	22	19	0	Mälardalen
ndes	137968	120823	31871	14834	121	75	100	46	Mälardalen
compress	176564	164273	38187	25594	86	86	85	0	Mälardalen
crc	143172	135796	25288	17932	44	44	43	0	Mälardalen
fdct	17350	6550	11525	9327	106	22	58	84	Mälardalen
minver	21668	4868	17265	518	167	167	159	0	Mälardalen
fft1	157880	123681	45816	11888	141	141	140	0	Mälardalen
ud	28427	20627	10415	10415	75	53	31	22	Mälardalen
adpcm	230123	196131	55609	21501	240	240	237	0	Mälardalen
nsichneu	316409	22009	294400	294400	256	0	256	256	Mälardalen
statemate	190496	10586	180110	180110	256	36	256	220	Mälardalen
fmref	12117800	2143590	10148500	10063200	256	161	256	95	TACLEBench
adpcm-dec	479761	460616	84090	64892	173	173	172	0	TACLEBench
adpcm-enc	482994	462750	70921	50646	178	178	177	0	TACLEBench
h264-dec	2609630	1661910	1143780	1130800	256	133	256	123	TACLEBench
huff-dec	821956	808273	112838	97680	84	84	84	0	TACLEBench
lift	1945120	1929300	282201	265799	140	140	140	0	TACLEBench
petrinet	38532	4632	34191	9633	256	229	256	27	TACLEBench
audiobeam	1883880	1824060	310955	302240	253	75	253	178	TACLEBench

APPENDIX B
TASK PRIORITY AND MEMORY FOOTPRINT

The integrated CRPD-CPRO approaches avoid double counting in the total memory reload overhead caused by the higher priority tasks. Therefore, the memory footprints of higher priority tasks can greatly affect the performance of the integrated CRPD-CPRO analysis.

To evaluate the impact of task memory footprints on the performance of the integrated CRPD-CPRO approaches, we performed a simple experiment using a single task set comprising 6 tasks (τ_1 to τ_6, where τ_1 has the highest priority). We increased the memory footprint (i.e., number of ECBs) of the highest priority task τ_1 and analyzed its impact on the total memory reload overhead μ_4 of the medium priority task τ_4. Task set parameters used in this experiment were set as follows. Core utilization was fixed at 0.7, with task utilizations generated using UUnifast algorithm. Each task was assigned parameters using the *ludcmp* benchmark[7]. Task periods were set such that $T_i = C_i/U_i$ (i.e., $T_1 = 161586$, $T_2 = 171642$, $T_3 = 220971$, $T_4 = 710848$, $T_5 = 1363503$ and $T_6 = 14533791$). Cache size was fixed to 256 cache sets with $d_{mem} = 8\mu$s.

[7]Here, we deliberately chose a benchmark with significant memory footprint to impact the memory reload overhead of other tasks.

TABLE II: Relative gain μ_4^{gain} for the CRPD-CPRO union and multi-set approaches by increasing the number of ECBs of τ_1

Increase of τ_1's ECBs (%)	μ_4^{gain} with integrated CRPD-CPRO union	μ_4^{gain} with integrated CRPD-CPRO multi-set
No Increase	9%	12%
20%	11%	16%
40%	13%	18%
60%	14%	20%
80%	15%	20%
100%	16%	20%

In this experiment, we evaluate the relative performance of the integrated CRPD-CPRO approaches in terms of memory reload overhead μ. Therefore, we report the *gain* on the total memory reload overhead μ^{gain} for task τ_4, i.e., μ_4^{gain}, by increasing the number of ECBs of the highest priority task τ_1.

The relative gain μ_i^{gain} is defined as $\mu_i^{gain} \stackrel{\text{def}}{=} \frac{\mu_i^{sep} - \mu_i^{int}}{\mu_i^{sep}}$ where μ_i^{sep} is the total memory reload overhead for task τ_i under the separate CRPD and CPRO analysis and μ_i^{int} is similarly the total obtained with the integrated analysis. For the integrated CRPD-CPRO Union approach, μ_i^{int} is given by (16), whereas for the CRPD-CPRO multi-set approach μ_i^{int} is given by (25). For the separate approaches, in each case the value of $\rho_{j,i}^{union}$ or $\rho_{j,i}^{mul}$ is used instead of $\delta_{j,i}$ or $\delta_{j,i}^{mul}$.

Observation 5. *If the memory footprint of higher priority tasks increase, then the relative gain of the integrated analyses over the state-of-the-art analyses increases.*

Table II shows that the gain in total memory reload overhead of τ_4 increases with the τ_1's memory footprint.

This behavior can be explained as follows. If one of the higher priority tasks (e.g., τ_1) has a big memory footprint (i.e., more ECBs) it can contribute more to both CRPD and CPRO of lower priority tasks. This results in increasing the overlap between the CRPD and CPRO of those tasks. In contrast, if the higher priority tasks have small memory footprints, they will have less impact on the CRPD and CPRO of medium and lower priority tasks and hence the overlap between the CRPD and CPRO will also be small.

This observation explains the rather small average schedulability gain in the experiments presented until now. Since tasks with smaller memory footprints mostly have lower execution times, their periods are most of the time shorter. Therefore, higher priority tasks usually have smaller memory footprints in the randomly generated task sets, hence resulting in a reduced gain. Yet, we note that this relationship between memory footprint, WCET, and period does not always hold in practice. Tasks with short periods and a relatively small WCET may still have a substantial memory footprint if they implemented via straight-line code. Similarly tasks with long WCETs may have a small memory footprint in the case where they implement a small loop that is repeated many times