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

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† [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 (FPSS). 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 FPSS 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 re-loaded, (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 FPSS by Katcher et al. [15] and Burns et al. [9] focused on scheduler overheads and context switch costs. Subsequent work on the

†Note that this form of cache persistence between jobs is distinct from cache persistence within loops, as studied for example by Cullmann [11].
We further decompose each task’s WCET into separate terms (WCET) of ECBs, i.e., all the cache blocks that are accessed by a platform does not provide memory address translation or virtual memory. Renesas SH7750 and NEC VR4181 and VR4121. Performing memory operations. The values for \( \tau \) is the worst-case memory access demand of any job of task to fetch data and instructions from the main memory. Consequently, it only accounts for execution \( \tau \) of the response time analysis.

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 \( \tau_i \) is defined by a triplet \((C_i, T_i, D_i)\), where \( C_i \) is the worst-case execution time (WCET) of \( \tau_i \), \( T_i \) is its minimum inter-arrival time and \( D_i \) is the relative deadline of each instance (or job) of \( \tau_i \). 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 worst-case execution time of \( \tau_i \) 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 \( \tau_i \); that is, the maximum time during which any job of \( \tau_i \) is performing memory operations. The values for \( C_i, PD_i \) and \( MD_i \) are determined assuming \( \tau_i \) 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 \( \tau_i \). Therefore, it holds that \( C_i \leq PD_i + MD_i \). The worst-case response time (WCRT) of task \( \tau_i \), 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 \( lp(i) \) to denote the set of tasks with priorities higher than that of \( \tau_i \). Similarly, \( lp(i) \) denotes the set of tasks with priorities lower than that of \( \tau_i \). Further, \( hep(i) \) denotes the set of tasks with priorities higher than or equal to that of \( \tau_i \) (i.e. \( hep(i) \) includes \( \tau_i \)). Finally, \( aff(i,j) = hep(i) \cap lp(j) \) denotes the set of intermediate tasks that can execute during the response time of \( \tau_i \) but may also be preempted by some higher priority task \( \tau_j \).

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 \( \tau_j \) executing during the response time of a task \( \tau_i \) by \( \gamma_{i,j} \).

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

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

where, \( UCB_k \) and \( ECB_j \) are the sets of UCBs and ECBs of task \( \tau_k \) and \( \tau_j \), 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.
\[ R_{i}^{ucb} = C_i + \sum_{j \in \text{hp}(i)} \left[ \frac{R_{i,j}^{ucb}}{T_j} \right] \times (C_j + \gamma_{i,j}^{ucb}) \] (2)

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

Note that the UCB-union multi-set 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 \( E_j(R_k) \) of each higher priority task \( \tau_j \) that can release during the response time of \( \tau_i \). It upper bounds the number of preemptions each task \( \tau_k \in \text{hp}(i,j) \) can suffer due to a higher priority task \( \tau_j \) during the response time of \( \tau_i \) by \( E_j(R_k) \) \( E_k(R_i) \) defined as \( \left[ \frac{R_{i,j}^{ucb}}{T_j} \right] \times \left[ \frac{R_{k}^{ucb}}{T_k} \right] \).

The resulting CRPD cost is denoted by \( \gamma_{i,j}^{ucb} \) and it accounts for the total preemption cost that can be caused by all jobs of \( \tau_j \) released during the response time of \( \tau_i \). \( \gamma_{i,j}^{ucb} \) is given by

\[ \gamma_{i,j}^{ucb} = d_{\text{mem}} \times \left| M_{i,j}^{ucb} \cap M_{i,j}^{ecb} \right| \] (3)

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

\[ M_{i,j}^{ucb} = \bigcup_{k \in \text{hp}(i,j)} \left( E_j(R_k) \bigcup E_k(R_j) \right) \] (4)

\[ M_{i,j}^{ecb} = \bigcup_{E_j(R_k)} \bigcup ECB_j \] (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.

\[ R_{i}^{ecb-m} = C_i + \sum_{j \in \text{hp}(i)} \frac{R_{i,j}^{ecb-m}}{T_j} \times C_j + \sum_{j \in \text{hp}(i)} \gamma_{i,j}^{ecb-m} \] (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 \( \tau_i \) is called a persistent cache block, if once loaded by \( \tau_i \), it will never be invalidated or evicted from the cache (in the same or different job) when \( \tau_i \) executes in isolation.

Definition 4 (non-Persistent Cache Block (nPCB)). A non-persistent cache block of task \( \tau_i \) 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 \( \tau_i \) (in the same or different job), even when \( \tau_i \) executes in isolation.

Based on the definition of non-persistent cache blocks (nPCBs), the notion of the residual memory demand \( MD'_i \) of a task \( \tau_i \) is defined as follows.

Definition 5 (Residual Memory Demand). The residual memory demand \( MD'_i \) of a task \( \tau_i \) is the worst-case memory demand of any job of \( \tau_i \) when all its PCBs are already loaded in the cache memory.

The number of PCBs and the residual memory demand \( MD'_i \) of a task can be used to bound its total memory demand \( MD_i(t) \) in isolation during a time interval of length \( t \):

\[ MD_i(t) \overset{\text{def}}{=} \min \left\{ \left( \frac{f}{T_i} \right) MD'_i + \left| \text{PCB}_i \right| \times d_{\text{mem}} \right\} \] (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 \( \tau_j \) due to evictions of its PCBs by tasks in \( \text{hep}(i) \setminus \tau_j \) while \( \tau_j \) is executing during the response time of \( \tau_i \).

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

\[ \rho_{j,i}^{\text{union}} \overset{\text{def}}{=} \left( \left( \frac{R_{i,j}}{T_j} \right) - 1 \right) \times \rho_{j,i}^{'} \] (8)

where \( \rho_{j,i}^{'} \) is the CPRO associated with a single job of \( \tau_j \).

\[ \rho_{j,i}^{'} = d_{\text{mem}} \times \left( \bigcup_{E_j(R_k)} \left( \bigcup_{\forall \tau_k \in \text{hep}(i) \setminus \tau_j} ECB_k \right) \right) \] (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 \( \tau_j \) released within the response time of \( \tau_i \). This is pessimistic. Indeed, considering two different tasks \( \tau_k \) and \( \tau_i \) in \( \text{hep}(i) \setminus \tau_j \), the number of times \( \tau_i \) can execute between different jobs of \( \tau_j \) is not necessarily equal to the number of times \( \tau_k \) 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 \( \tau_i \), 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 \( \rho_{j,i}^{\text{mul}} \) as:

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

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

\[ M_{j,i}^{ecb} = \bigcup_{E_j(R_k)+1} \bigcup_{E_j(R_k)} \bigcup ECB_k \] (11)

\[ M_{j,i}^{pcb} = \bigcup_{\forall \tau_k \in \text{hp}(j)} \bigcup_{E_j(R_k)} \bigcup ECB_k \] (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 \( \tau_i \) under FPPS can be re-written as follows:
where $\gamma_{i,j}$ is either equal to $\gamma_{i,j}^{\text{ucb-m}}$, (3), or $\rho_{j,i}^{\text{mul}} \times \gamma_{i,j}^{\text{ucb}}$, (1), and $\rho_{j,i}$ is either $\rho_{j,i}^{\text{nom}}$, (8), or $\rho_{j,i}^{\text{mul}}$, (10). In the remainder of this paper, unless stated otherwise, we assume that (13) is used to calculate the WCRT of a task $\tau_i$.

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 $\tau_i$ is bounded by the sum of the CRPD and the CPRO experienced by every task executing during $\tau_i$’s response time. This overhead is denoted by $\mu_i$ and is defined as follows.

**Definition 7** (Total Memory Reload Overhead ($\mu_i$)). Let $\text{CRPD}_{i,j}(S)$ and $\text{CPRO}_{i,j}(S)$ be the total actual CRPD and CPRO suffered by $\tau_j$ during the response time of one job of $\tau_i$ in a given schedule $S$. The total memory reload overhead $\mu_i$ during the response time of $\tau_i$ is the maximum sum of the CRPD and CPRO of all tasks executing during $\tau_i$’s response time in any schedule $S$. Formally,

$$
\mu_i \overset{\text{def}}{=} \max_{S} \left\{ \sum_{\tau_j \in \text{hep}(i)} \left( \text{CRPD}_{i,j}(S) + \text{CPRO}_{i,j}(S) \right) \right\}
$$

Example 1. Let $\tau$ be composed of three tasks $\{\tau_1, \tau_2, \tau_3\}$ with $\tau_1$ having the highest priority and $\tau_2$ the lowest. Fig. 1 presents the task set parameters and the worst-case schedule for $\tau_3$ 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 is highlighted in green.

Initially, the cache is empty and with $\tau_3$ being the first task to execute it loads all its ECBs into the cache. When $\tau_2$ preempts $\tau_3$ for the first time, it also loads its ECBs. Similarly, $\tau_2$ is preempted by the highest priority task $\tau_1$ at time 2. Note that ECBs of task $\tau_1$ and UCBs/PCBs of task $\tau_2$ are mapped to the same cache blocks, i.e., $\{7,8,9,10\}$. Therefore, when $\tau_2$ resumes its execution after the completion of the first job of $\tau_1$ it needs to reload all its UCBs, (highlighted in red) as they were evicted by $\tau_1$. These additional memory accesses will be accounted for as CRPD.

Since, the first job of $\tau_2$ loads all $\tau_2$’s ECBs (PCBs and nPCBs) into the cache, subsequent jobs of $\tau_2$ 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 $\mu_3$ during $\tau_3$’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 $\mu_3$, we have the following.

$$
\rho_3 \leq \gamma_{3,3}^{\text{ucb-m}} + \gamma_{3,2}^{\text{ucb-m}} + \rho_{1,3}^{\text{mul}} + \rho_{2,3}^{\text{mul}}
$$

Since $\tau_3$ is the only task with useful cache blocks ($\text{UCB}_2 = \{7,8,9,10\}$), it is also the only task suffering from CRPD.

Therefore, $\gamma_{3,2}^{\text{ucb-m}} = 0$. Using (3), we have (note that $E_1(R_3) = 3$, $E_1(R_2) = 2$, $E_2(R_3) = 3$, and $E_2(R_3) = 1$):

$$
\gamma_{3,3}^{\text{ucb-m}} = d_{\text{mem}} \times |(3 \times \text{UCB}_3 \cup 3 \times \text{UCB}_2) \cap (3 \times \text{ECB}_1)| = d_{\text{mem}} \times 12
$$

Similarly, when calculating the CPRO we can see that the set of PCBs for all tasks except $\tau_2$ is empty. Hence, the total CPRO during the response time of task $\tau_3$ comes only from the evictions of PCBs of task $\tau_2$. Assuming that the CPRO is calculated using (10) we have $\rho_{1,3}^{\text{mul}} = 0$ and $\rho_{2,3}^{\text{mul}} = d_{\text{mem}} \times |(2 \times \text{PCB}_2) \cap (4 \times \text{ECB}_3 \cup 3 \times \text{ECB}_1)| = d_{\text{mem}} \times 8$

Adding CRPD and CPRO, it follows that the total memory reload overhead during the response time of $\tau_3$ is upper-bounded by $d_{\text{mem}} \times 20$. Thus it appears that 20 cache blocks...
need to be reloaded during the response time of $\tau_3$. 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 in Fig. 1b), but both scheduling scenarios cannot happen at the same time. It is not possible for the three jobs of $\tau_i$ to result in the group of 4 cache block reloads three times over due to preemptions (accounted for in $\tau_3^{0.3b-m}$) and two times over due to cache persistence overheads (accounted for in $p_{\text{mul}}^{\text{pc}}$). This observation leads to the following lemma.

**Lemma 1.** Let us assume that the total CRPD during the response time of task $\tau_i$ is computed using (1) or (3) and that the total CPRO during $\tau_i$’s response time is computed with (8) or (10). Let $b_{k,\ell}$ be the $\ell$th cache block of a task $\tau_k \in \text{hp}(i)$, i.e., $b_{k,\ell} \in \text{ECB}_k$. 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 $\tau_k$, i.e., $b_{k,\ell} \in \bigcup \text{UCB}_k \cap \text{PCB}_k$.

The claim follows directly from the fact that (1) and (3) account for the evictions of UCBs of tasks in $\text{hp}(i)$. 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 $\text{hp}(i)$. 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 $b_{k,\ell} \in \text{UCB}_k \cap \text{PCB}_k$.

This observation leads to the following lemma.

**Lemma 2.** For any task $\tau_k \in \text{hp}(i)$ executing during the response time of a lower priority task $\tau_i$ (e.g., $\tau_j$), only higher priority tasks than $\tau_k$ (e.g., $\tau_1$) can participate in both the CRPD and CPRO of $\tau_k$. This observation leads to the following lemma.

**Lemma 3.** Let $\gamma_i$ be an upper-bound on the total CRPD during the response time $R_i$ of $\tau_i$. Further assume that $\gamma_i$ is computed using the UCB-union approach, i.e., $\gamma_i \overset{\text{def}}{=} \sum \binom{B_{\text{ucb}}}{\text{PCB}_j} \Gamma_{\text{UCB}_j \cap \text{PCB}_j}$. Let $\Delta_i$ be an upper-bound on the portion of the total memory reload overhead during $\tau_i$’s response time that is not accounted for in $\gamma_i$, that is, $\Delta_i = \mu_i - \gamma_i$, then we have

$$\Delta_i \overset{\text{def}}{=} \sum_{\tau_j \in \text{hp}(i)} \left( \binom{B_{\text{ucb}}}{\text{PCB}_j} - 1 \right) \times \delta_j, \quad \text{where}$$

$$\delta_j \overset{\text{def}}{=} \frac{d_{\text{mem}}}{\text{PCB}_j} \bigg| \bigg( \bigcup_{\tau_k \in \text{aff}(i,j)} \bigg( \bigcup_{\tau_{l,j} \in \text{hp}(j)} \text{ECB}_k \bigg) \bigg) \bigg|$$

**Proof.** It was proven in [23] that $\gamma_i$ upper-bounds the total CRPD during the response time $R_i$. Therefore, the portion of the total memory reload overhead $\mu_i$ that is not accounted for in $\gamma_i$ is a subset of the total CRPD during $\tau_i$’s response time. Similar to the calculation of the total CPRO, at most $\left( \binom{B_{\text{ucb}}}{\text{PCB}_j} - 1 \right)$ jobs of each higher priority task $\tau_j$ can suffer memory reload overhead $\delta_j$, not yet accounted for in $\gamma_i$. Since the total CPRO is an upper-bound on $\Delta_i$, using (8) and (9) we have

$$\Delta_i \leq \sum_{\tau_j \in \text{hp}(i)} \left( \binom{B_{\text{ucb}}}{\text{PCB}_j} - 1 \right) \times \delta_j, \quad \text{with} \quad \delta_j, 1 \leq \rho_j, 1.$$

We now prove the validity of $\delta_j, 1$. Since a fixed-priority scheduling algorithm is used, only tasks with priorities higher than or equal to the priority of $\tau_i$ (i.e., tasks in $\text{hp}(i)$) can execute during the response time of $\tau_i$. Therefore, any task $\tau_k \in \text{hp}(i) \setminus \text{trc}$ can execute between two subsequent jobs of another task $\tau_j$ and hence participate in $\tau_j$’s CRPD by eviction some or all its PCBs. Let $\gamma_k$ be any task in $\text{hp}(i) \setminus \text{trc}$. Two cases need to be considered (note that $\text{hp}(i) \setminus \text{trc} = \text{aff}(i,j) \cup \text{hp}(j)$).

1. If $\tau_k \in \text{aff}(i,j)$. Since $\tau_k$ has a lower priority than $\tau_j$ it cannot preempt $\tau_j$, and hence $\tau_k$ does not contribute to the CRPD of $\tau_j$. Therefore, the memory reload overhead generated by $\tau_k$ on $\tau_j$ is not part of $\gamma_i$, and must be entirely accounted for in $\delta_j, 1$. This worst-case interference of $\tau_k$ on $\tau_j$ is maximized when $\tau_k$ loads all its cache blocks (i.e., $\text{ECB}_k$).

2. If $\tau_k \in \text{hp}(j)$ then, by Lemma 2, $\tau_k$ may contribute to both the CRPD and CPRO of $\tau_j$. As stated in Lemma 1, the evictions of cache blocks of $\tau_j$ in $\text{UCB}_j \cap \text{PCB}_j$ consider the more precise, but also more complex, multi-set variants of the CRPD and CPRO calculation.

It follows from Lemma 1 that only the cache blocks in $\bigcup_{\tau_j \in \text{hp}(i)} \left( \text{UCB}_j \cap \text{PCB}_j \right)$ 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 $\bigcup_{\tau_k \in \text{hp}(i)} \left( \text{UCB}_j \cap \text{PCB}_j \right)$, since their evictions will be accounted for in the CPRO; or, (ii) during the CPRO calculation by not considering the eviction of UCBs in $\bigcup_{\tau_j \in \text{hp}(i)} \left( \text{UCB}_j \cap \text{PCB}_j \right)$, since their evictions 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)).
were already considered in $\Gamma_i$. Therefore, the number of cache block evictions caused by $\tau_k$ on $\tau_j$ that were not accounted for in $\Gamma_i$ is maximized when $\tau_k$ loads all the cache blocks in $ECB_k \setminus (UCB_j \cap PCB_j)$.

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

$$S_{\tau_j} = \bigcup_{\tau_k \in aff(i,j)} ECB_k \bigcup_{\tau_l \in hep(j)} ECB_l \setminus (UCB_j \cap PCB_j)$$

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

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

**Theorem 1.** The total memory reload overhead $\mu_i$ during $\tau_i$’s response time is upper-bounded by

$$\sum_{\forall j \in hep(i)} \left( \left\lceil \frac{R_i}{T_j} \right\rceil \times \gamma_{i,j}^{uch} + \left\lceil \frac{R_i}{T_j} \right\rceil - 1 \right) \delta_{\tau_j,i}$$

(16)

**Proof.** Follows from Lemma 3 since $\mu_i = \Delta_i + \Gamma_i$. \(\Box\)

This directly leads to the following theorem:

**Theorem 2.** The WCRT of $\tau_i$ is upper-bounded by the smallest positive solution to

$$R_i = C_i + \sum_{\forall j \in hep(i)} \left( \gamma_{i,j} + \min \left\{ \left\lceil \frac{R_i}{T_j} \right\rceil \frac{C_j}{T_j}, \left\lceil \frac{R_i}{T_j} \right\rceil PD_j \right\} \right) + MD_j(R_i) + \delta_{\tau_j,i}$$

(17)

where

$$\delta_{\tau_j,i} \overset{\text{def}}{=} \left( \frac{R_i}{T_j} - 1 \right) \delta_{\tau_j,i}$$

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

**Proof.** By Theorem 1 and substituting $\delta_{\tau_j,i}$ for $\rho_{\tau_j,i}$ in (13) \(\Box\)

Since, $\delta_{\tau_j,i}$ calculated using (15) is always less than or equal to $\rho_{\tau_j,i}$ 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 $\tau_3$ 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 $\tau_2$ is empty. Therefore, based on (15), we have $\delta_{1,3} = 0$ and

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

According to Theorem 1, $\mu_3$ is thus upper-bounded by $(12 \times d_{mem})$, which is in this case the exact overhead experienced during the response time of $\tau_3$ 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 $\tau_k \in hep(i)$ executing during the response time of $\tau_i$ can contribute to the total CRPD, the UCB-union multi-set approach (3) considers that only a subset of $\tau_k$’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 $\mu_i$ that is not yet accounted for in the total CRPD.

**Example 3.** The example task set in Fig. 2 has three tasks $\tau_1$, $\tau_2$ and $\tau_3$ with priorities assigned in numerical order such that $\tau_1$ has the highest priority. We want to analyze the total memory reload overhead $\mu_i$ during the response time of $\tau_3$. Task $\tau_2$ is the only task with $UCB_2 \cap PCB_2 \neq \emptyset$. The sets of PCBs and PCBs of $\tau_1$ and $\tau_3$ are empty. Therefore, $\tau_2$ is the only task that may suffer CRPD and CPRO. The total memory reload overhead $\mu_3$ is thus bounded by the sum of the CRPD and CPRO suffered by $\tau_2$ during the response time of $\tau_3$.

By Lemma 2, $\tau_1$ is the only task that can contribute to both $\tau_3$’s CRPD and CPRO. Since $\tau_1$ can preempt each job of $\tau_2$ at most once (i.e., $E_1(R_2) = 1$), and because $\tau_2$ releases three jobs during $\tau_1$’s response time (i.e., $E_2(R_3) = 3$), at most three jobs of $\tau_1$ are preempting jobs of $\tau_2$ during the response time of $\tau_2$, i.e., $E_1(R_2)E_2(R_3) = 3$. Therefore, at most three jobs of $\tau_1$ may be contributing to both $\tau_2$’s CRPD and CPRO during $\tau_1$’s response time. The two remaining jobs of $\tau_1$ can only execute between jobs of $\tau_2$, and hence contribute only to $\tau_2$’s CPRO.

To calculate the CPRO that any task $\tau_j \in hep(i)$ can suffer during the response time of $\tau_i$ taking into consideration what has already been accounted for in the CRPD cost, we first analyze the impact of each task in $hep(i) \setminus \tau_j$ on the CPRO of $\tau_j$. We characterize the maximum number of times a task $\tau_k \in hep(i) \setminus \tau_j$ can execute between successive jobs of $\tau_j$. To do so, we separately analyze the tasks in $aff(i,j)$ (Lemma 4) and the tasks in $hep(j)$ (Lemma 5). We then identify how many jobs of each task contribute only to the CPRO of $\tau_j$ and how many jobs contribute to both the CRPD and the CPRO of $\tau_j$ (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 $\tau_j \in hep(i)$ 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 \( \tau_j \) released during \( \tau_i \)'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 \( \tau_j \) released during \( \tau_i \)'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 \( \tau_i \in \text{hp}(j) \) (e.g., \( \tau_1 \) in Fig. 2) during the response time of a lower priority task \( \tau_j \) (e.g., \( \tau_2 \) in Fig. 2) can preempt \( \tau_j \) (e.g., \( \tau_2 \) in Fig. 2). The jobs that do not preempt cannot contribute to both the CRPD and the CPRO of \( \tau_j \). This observation leads to the following Lemma:

**Lemma 6.** For a task \( \tau_j \in \text{hp}(i) \) executing during the response time of \( \tau_i \), the number of jobs of any priority task \( \tau_k \in \text{hp}(j) \) that are already accounted for in the CRPD \( \gamma_{i,j}^{\text{ucb}} \) is given by \( N_{i,j}^{\text{double}}(R_i) = \min\{E_i(R_i) : E_i(R_j)E_j(R_i)\} \).

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

1) The calculation of \( M_{i,j}^{\text{ecb}} \) (5) assumes that a task \( \tau_i \in \text{hp}(j) \) can release at most \( E_i(R_i) \) jobs during the response time \( R_i \) of \( \tau_i \). Therefore, at most \( E_i(R_i) \) jobs of \( \tau_i \) preempting \( \tau_j \) are accounted for in the calculation of \( \gamma_{i,j}^{\text{ucb}} \) in (3).

2) The calculation of \( M_{i,j}^{\text{ecb}} \) (4) assumes that for any task \( \tau_k \in \text{aff}(i,j) \), \( E_i(R_k)E_k(R_i) \) is an upper bound on the number of times \( \tau_k \) can be preempted by \( \tau_j \) during \( \tau_i \)'s response time. Therefore, at most \( E_i(R_j)E_j(R_i) \) jobs of \( \tau_j \) are accounted for in \( \gamma_{i,j}^{\text{ucb}} \) in (3).

It follows that the number of jobs of \( \tau_k \) accounted for in \( \gamma_{i,j}^{\text{ucb}} \) is given by \( N_{i,j}^{\text{double}}(R_i) \).

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

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

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

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

\[
\Delta_i^m \leq \sum_{\tau_j \in \text{hp}(i)} \delta_{i,j}^{\text{mul}}.
\]

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

1. Since \( \tau_j \) can release at most \( \left\lceil \frac{T_j}{T_i} \right\rceil \) jobs in a time window of length \( t \), the PCBs of \( \tau_j \) can be evicted at most \( \left\lfloor \frac{T_i}{T_j} - 1 \right\rfloor \) times within the time window of length \( t \), contributing to \( \text{CPRD}^4 \). Therefore, the largest set of PCBs of \( \tau_j \) that can be evicted during the response time of \( \tau_i \) is upper bounded by the multi-set \( M_{i,j}^{\text{ecb}} = \bigcup_{E_j(R_i)} PCB_j \) 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 \( \tau_j \) during the response time of \( \tau_i \) is upper bounded by \( \left( E_j(R_k) + 1 \right) \times E_k(R_i) \). Hence, the largest set of ECBs that can be loaded by \( \tau_k \) between successive jobs of \( \tau_j \) during the response time of \( \tau_i \) is given by \( \bigcup_{E_j(R_i)} \text{ECB}_k \).

Therefore the largest set of ECBs loaded by the tasks in \( \text{aff}(i,j) \) between successive executions of \( \tau_j \) is upper bounded by \( M_{i,j}^{\text{ecb}} = \bigcup_{E_j(R_i)} \bigcup_{E_k(R_i)} \text{ECB}_k \) given in (23).

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

\[\text{Recall from (7) that all PCBs are assumed to be loaded once anyway.} \]
in the CRPD of $\tau_j$ is equal to $N_{double}^{double}(R_i)$. Therefore, instead of assuming that all jobs released by $\tau_j \in \text{hp}(j)$ during the response time of $\tau_i$ contribute to $\delta_{j,i}^{\text{mul}}$, the multi-set $M_{hp}^{\text{hp}} \cup M_{hp}^{\text{mt}}$ separately categorizes the impact of jobs of $\tau_j$ that can/cannot be contributing to both the CRPD and CPRO of $\tau_j$ during the response time of $\tau_i$.

Since $N_{double}^{double}(R_i)$ is the number of jobs of $\tau_i$ that were already considered in the CRPD of $\tau_j$, then $E_i(R_i) - N_{double}^{double}(R_i)$ jobs of $\tau_i$ only contribute to the CPRO of $\tau_j$. The memory reload overhead generated by these $E_i(R_i) - N_{double}^{double}(R_i)$ jobs of $\tau_i$ on $\tau_j$ is not part of $\Gamma_i^m$ and must therefore be entirely accounted for in $\delta_{j,i}^{\text{mul}}$. The worst-case interference of all these jobs is maximized when every job of $\tau_i$ loads all its cache blocks (i.e., $ECB_i$). Hence, the worse-case impact that these jobs of $\tau_i$ can have on the $\tau_j$’s CPRO is bounded by the multi-set $E_i(R_i) - N_{double}^{double}(R_i)$

For all jobs of $\tau_i$ that can contribute to both the CRPD and CPRO of $\tau_j$, i.e., $N_{double}^{double}(R_i)$, then as stated in Lemma 1, the evictions of cache blocks of $\tau_j$ in UCB$_j \cap PCB_j$ were already considered in $\Gamma_i^m$. Therefore, the number of cache block evictions caused by these $N_{double}^{double}(R_i)$ jobs of $\tau_i$ on $\tau_j$ that were not accounted for in $\Gamma_i^m$ is maximized when each job loads all the cache blocks in $ECB_i \setminus (UCB_j \cap PCB_j)$. Hence, the worse-case additional impact of all jobs of $\tau_i$ that contribute to both the CRPD and CPRO of $\tau_j$ is bounded by the multi-set $\bigcup ECB_i \setminus (UCB_j \cap PCB_j)$ 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 $\tau_j$ during the response time of $\tau_i$ is upper bounded by $M_{ecb}^{\tau_j} = M_{ecb}^{\tau_j - aff} \cup M_{hp}^{\text{hp} - mt}$ given by (22). Hence, the largest set of PCBs of $\tau_j$ that can be evicted by the tasks in $\text{hp}(i) \setminus \tau_j$ within the response time of $\tau_i$ by evictions not already considered in $\Gamma_i^m$, is upper bounded by the intersection of $M_{ecb}^{\tau_j}$ with $M_{ecb}^{\tau_j}$. Since reloading a cache block takes at most $d_{mem}$ time units, an upper bound on the total CRPO $\delta_{j,i}^{\text{mul}}$, not already included in the CRPD, is given by $d_{mem} \times M_{ecb}^{\tau_j} \cap M_{ecb}^{\tau_j}$ in (20).

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

**Theorem 3.** The total memory reload overhead $\mu_i$ during $\tau_i$’s response time is upper-bounded by

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

**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 $\tau_j$ is upper-bounded by the smallest positive solution to

$$R_i = C_i + \sum_{\forall j \in \text{hp}(i)} \left( \gamma_{i,j} + \min \left\{ \left[ R_i \over T_j \right] C_j ; \left[ R_i \over T_j \right] P D_j + \left( M_{ecb}(R_i) + \delta_{j,i}^{\text{mul}} \right) \right\} \right)$$

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

**Proof.** By Theorem 3 and substituting $\delta_{j,i}^{\text{mul}}$ for $\rho_{j,i}$ in (13).
the same cache blocks of $\tau_j$ as the tasks with higher priority (i.e., in $\text{hip}(j)$). 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.

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 ($d_{\text{mem}}$): In this experiment, we analyze the impact of block reload time $d_{\text{mem}}$ on the performance of all the approaches by varying it between $2\mu$s to $20\mu$s, 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 $d_{\text{mem}}$, the integrated and separate CRPD-CPRO analyses produce similar results.

For smaller values of $d_{\text{mem}}$ (i.e., between $2\mu$s and $4\mu$s) 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 $d_{\text{mem}}$ (i.e., $d_{\text{mem}} > 15\mu$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 $d_{\text{mem}}$ between $8\mu$s to $12\mu$s 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
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 Acknowledgments. This paper is supported by NWO Veni Project, “The

REFERENCES


We increased the memory footprint (i.e., number of ECBs) of using UUnifast algorithm. Each task was assigned parameters the highest priority task \( \tau_1 \) and analyzed its impact on the total memory reload overhead \( \mu_i^{\text{sep}} \) of task \( \tau_i \), i.e., \( \mu_i^{\text{sep}} \), by increasing the number of ECBs of the highest priority task \( \tau_1 \).

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

**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 \( \tau_1 \) increases with the \( \tau_1 \)'s memory footprint.

This behavior can be explained as follows. If one of the higher priority tasks (e.g., \( \tau_1 \)) 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 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.

**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 (\( \tau_1 \) to \( \tau_6 \), where \( \tau_1 \) has the highest priority). We increased the memory footprint (i.e., number of ECBs) of the highest priority task \( \tau_1 \) and analyzed its impact on the total memory reload overhead \( \mu_i \) of the medium priority task \( \tau_4 \). 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 ludecmp benchmark7. Task periods were set such that

---

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