Technical Report

Tightening the CRPD Bound for Multilevel non-Inclusive Caches

Syed Aftab Rashid*

Geoffrey Nelissen

Eduardo Tovar*

*CISTER Research Centre

CISTER-TR-211009

2021

Tightening the CRPD Bound for Multilevel non-Inclusive Caches

Syed Aftab Rashid*, Geoffrey Nelissen, Eduardo Tovar*

*CISTER Research Centre

Polytechnic Institute of Porto (ISEP P.Porto)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail: syara@isep.ipp.pt, gnn@isep.ipp.pt, emt@isep.ipp.pt

https://www.cister-labs.pt

Abstract

Tasks running on microprocessors with cache memories are often subjected to cache related preemption delays (CRPDs). CRPDs may significantly increase task execution times, thereby, affecting their schedulability. Schedulability analysis accounting for the impact of CRPD has been extensively studied over the past two decades for systems with a single level of cache. Yet, the literature on CRPD for multilevel non-inclusive caches is relatively scarce. Two main challenges exist when analyzing multilevel caches: (1) characterization of the indirect effect of preemption, i.e., capturing the increase in cache interference at lower cache levels (e.g., L2 cache) due to the evictions of cache content from a higher cache level (e.g., L1 cache), and (2) upper bounding the maximum CRPD suffered by tasks at lowercache levels (e.g., L2 cache), i.e., determining the cache content of tasks that can be evicted from lower cache levels in case of preemptions. Existing analysis that focus on bounding CRPD for multilevel non-inclusive caches overestimate the values of (1) and (2) leading to pessimistic worst-case response time (WCRT) estimations. In this work, we reducethe excessive pessimism of the state-of-the-art CRPD analysis for multilevel non-inclusive caches by (i) introducing the notion of multi-level useful cache blocks, i.e., cache blocks that can cause CRPD at different cache levels, and use it to compute a tighter bound on the indirect effect of preemption of tasks; and (ii) deriving a new analysis to compute tighter bounds on the CRPD of tasks at lower cache levels (e.g., L2 cache). We performed a thorough experimental evaluation using benchmarks to compare the performance of our proposed CRPD analysis against the state-of-the-art CRPD analysis. Experimental results show that our proposed CRPD analysis dominates the existing analysis and improves task set schedulability by up to 20% percentage points

© 2021 CISTER Research Center
www.cister-labs.pt

Tightening the CRPD Bound for Multilevel non-Inclusive Caches

Syed Aftab Rashid[a,b], Geoffrey Nelissen[c], Eduardo Tovar[a]

[a] *CISTER, ISEP, Polytechnic Institute of Porto, Porto, Portugal*
[b] *VORTEX CoLab, Porto, Portugal*
[c] *Eindhoven University of Technology, Eindhoven, The Netherlands*

Abstract

Tasks running on microprocessors with cache memories are often subjected to *cache related preemption delays* (CRPDs). CRPDs may significantly increase task execution times, thereby, affecting their schedulability. Schedulability analysis accounting for the impact of CRPD has been extensively studied over the past two decades for systems with a single level of cache. Yet, the literature on CRPD for multilevel non-inclusive caches is relatively *scarce*. Two main challenges exist when analyzing multilevel caches: (1) characterization of the *indirect effect of preemption*, i.e., capturing the increase in cache interference at lower cache levels (e.g., L2 cache) due to the evictions of cache content from a higher cache level (e.g., L1 cache), and (2) upper bounding the maximum CRPD suffered by tasks at lower cache levels (e.g., L2 cache), i.e., determining the cache content of tasks that can be evicted from lower cache levels in case of preemptions. Existing analysis that focus on bounding CRPD for multilevel non-inclusive caches overestimate the values of (1) and (2) leading to pessimistic worst-case response time (WCRT) estimations. In this work, we reduce the excessive pessimism of the state-of-the-art CRPD analysis for multilevel non-inclusive caches by (i) introducing the notion of multi-level useful cache blocks, i.e., cache blocks that can cause CRPD at different cache levels, and use it to compute a tighter bound on the indirect effect of preemption of tasks; and (ii) deriving a new analysis to compute tighter bounds on the CRPD of tasks at lower cache levels (e.g., L2 cache). We performed a thorough experimental evaluation using benchmarks to compare the performance of our proposed CRPD analysis against the state-of-the-art CRPD analysis. Experimental results show that our proposed CRPD analysis dominates the existing analysis and improves task set schedulability by up to 20% percentage points.

1. Introduction

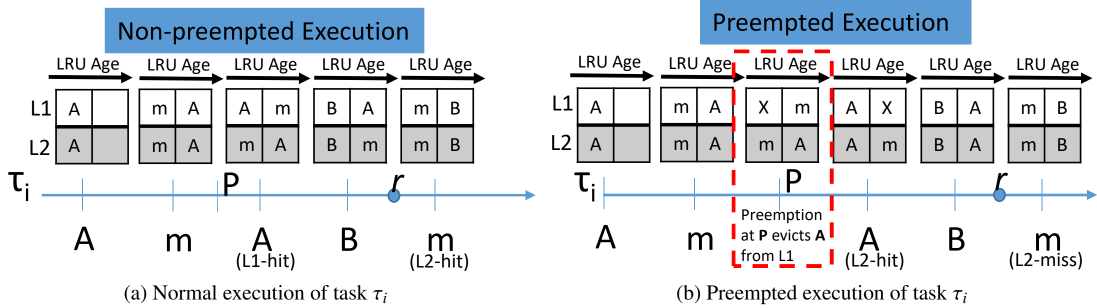
Modern processors use a hierarchy of cache memories to reduce average main memory access time. This hierarchy may comprise two or more cache levels, providing a trade-off between cost and speed. The cache level closest to the processor (also referred to as level-one or L1 cache) is the fastest with the least capacity. The level-two cache (also referred to as L2 cache) is usually slower than the L1 cache but has a larger capacity. Some processors also use a level-three cache to further expand caching capacity. Although, the presence of a cache hierarchy improves the average performance by reducing task's accesses to the main memory, it also causes large variations in the worse-case execution time (WCET) and worst-case response time (WCRT) of tasks. These variations strongly depend on the availability of task's data and instructions at different cache levels (i.e., the number of cache hits and cache misses).

In systems that use preemptive scheduling strategies, the use of a cache hierarchy poses additional challenges. These challenges stem from cache sharing between tasks at different cache levels with the execution of one task potentially evicting memory blocks previously loaded into one or more cache levels by other tasks. These cache evictions may lead to additional execution delays referred to as, *Cache Related Preemption Delays* (CRPDs). CRPDs are delays suffered by the preempted tasks in reloading *useful cache blocks* (UCBs) (memory blocks cached before the preemption and potentially reused after) that were evicted from the cache during the execution of preempting tasks.

CRPDs can significantly affect task set schedulability. Considering the same, CRPD for single-level caches and its impact on schedulability has been extensively studied over the last two decades [1–5]. However, as mentioned in [6, 7], existing analysis methods used to compute CRPD for single-level caches cannot be easily extended to multilevel caches. This is mainly because when considering only a single cache level, e.g., L1 cache, the CRPD any task τ_i can suffer due to preemptions solely depends on the inter-task cache conflicts between τ_i and tasks with

$\mathcal{ID}_{r,p} = \{m | m$
$\wedge (\mathcal{CU}$

Overestimation in the indirect effect of preemption



(a) Normal execution of task τ_i (b) Preempted execution of task τ_i

Figure 1: Illustration of the indirect effect of preemption during the execution of a task τ_i. L1 and L2 caches are assumed to be two-way set-associative and all memory blocks used by task τ_i, i.e., A, B and m, are mapped to the same L1/L2 cache set. The cache replacement policy is Least-Recently-Used (LRU), i.e., blocks are placed in the cache from most- to least-recently-used (from left to right) and on a cache miss the least-recently-used block is replaced.

higher priorities than τ_i. However, when considering more than one cache level, the CRPD of a task τ_i does not only depend on its inter-task cache conflicts but also on the increase in τ_i's intra-task cache conflicts (i.e., between different memory blocks of τ_i) at lower cache levels, e.g., L2 cache.

In architectures with multiple cache levels, a task τ_i may access a lower cache level (e.g., L2 cache) only after preemption , i.e., in case of a L1 cache miss after preemption. This additional access to L2 cache only during the preempted execution of τ_i may change the position of existing memory blocks in the L2 cache, thereby, generating additional cache conflicts. To illustrate, consider the sequence of memory references (from left to right) during the execution of a task τ_i depicted in Fig. 1. Fig. 1a and 1b respectively shows the contents of the L1 and L2 cache during a non-preempted and a preempted execution of τ_i, with a preemption at point P. Fig. 1a shows that during the non-preempted execution of τ_i, the second reference to memory block m is a L2 cache hit. However, the same reference to m results in a L2 cache miss during the preempted execution of τ_i (see Fig. 1b). This is mainly because, memory block A is evicted from the L1 cache due to preemption, and is accessed from the L2 cache when τ_i resumes after preemption. This additional access to A in the L2 cache changes the position of memory block m in L2, which eventually leads to a L2 cache miss for the second reference to m after preemption.

The phenomenon suffered by memory block m in the above example is referred to as the *indirect effect of preemption* [6]. Few existing works [6, 7] that focus on the CRPD analysis of multilevel caches show that a sound CRPD estimate can only be obtained by accurately quantifying the indirect effect of preemption that can be suffered by every memory block used by tasks. However, these existing approaches [6, 7] overestimate the number of memory blocks that can contribute to the indirect effect of preemption of memory blocks used by a task τ_i. Similarly, the existing analysis in the state-of-the-art [6, 7] also overestimate the CRPD task τ_i can suffer due to memory blocks that are used from lower cache levels. For example, when computing the CRPD due to a memory block $m_{x,i}$ of task τ_i that has multiple references categorized as L2 cache hits after a program point P in τ_i, the existing analysis [6, 7] assume that all references to $m_{x,i}$ after P can contribute to the CRPD of τ_i. Clearly, this assumption is pessimistic, considering that multiple references to memory block $m_{x,i}$ after a preemption may result in L2 cache hits but, only those references of $m_{x,i}$ that are impacted directly or indirectly due to preemption can contribute to the CRPD.

In this work, we first identify the source of pessimism in the SoA CRPD analysis for multilevel caches. Then, we provide solutions to reduce that pessimism. The main contributions of this paper are as follows: (1) We define the notion of useful cache blocks (UCBs) for multilevel caches based on the cache level from which those UCBs may be re-used. We also show how these UCBs can be determined; (2) using this notion of multilevel UCBs, we reduce the overestimation in the computation of the indirect effect of preemption; (3) we present a tighter analysis to compute the CRPD of tasks at lower cache levels, e.g., L2. At the L2 cache, tasks can suffer CRPD due to memory blocks that have one or more references categorized as L2 cache hit(s). Our approach identifies how many references to such memory blocks can be impacted directly or indirectly due to preemptions and therefore may contribute to the CRPD; (4) we perform an extensive experimental evaluation that compares the performance of our proposed CRPD analysis against an existing analysis from the state-of-the-art using a set of benchmarks. Results show that our analysis improves task set schedulability by up to 20 percentage points in comparison to the state-of-the-art analysis.

2. System Model and Assumptions

We focus on a single-core processor with a two-level *non-inclusive* cache hierarchy (i.e., with a L1 and L2 cache). Non-inclusive cache hierarchy implies that the content in the L1 cache may or may not be duplicated in the L2 cache. We only consider instruction references and assume that L1 and L2 are set-associative LRU caches. W_1 and W_2 respectively denote the number of cache *ways* or cache *associativity* of L1 and L2 cache. Note that LRU caches conceptually assign each cached memory block an *age* indicating its position in the cache, i.e., the most-recently used element in the cache set of level Lx has an age 0 and the least-recently used element will have an age $W_x - 1$. The set of all L1/L2 cache sets is denoted by $\mathbb{S}_1/\mathbb{S}_2$. The total number of cache sets in the L1 and L2 are given by $|\mathbb{S}_1|$ and $|\mathbb{S}_2|$, respectively. Specifically, we focus on architectures with a cache configuration such that, $|\mathbb{S}_1| \leq |\mathbb{S}_2|$ and $W_1 \leq W_2$.

We consider a task set Γ composed of n sporadic tasks $\{\tau_1, \tau_2, ...\tau_n\}$. Each task $\tau_i \in \Gamma$ is defined using a triplet, (C_i, T_i, D_i). C_i denote the worst-case execution time (WCET) of task τ_i, T_i is its minimum inter-arrival time and D_i is the relative deadline of each job of τ_i. We assume tasks have constrained deadlines, i.e., $D_i \leq T_i$ and each task has a unique priority. R_i denote the WCRT of τ_i, i.e., the longest time between the arrival and the completion of any job of τ_i. Tasks can be scheduled using any fixed-priority preemptive scheduling (FPPS) algorithm such as Rate or Deadline Monotonic [8]. Furthermore, we use hp(i) and lp(i) to denote the set of tasks with priorities higher, respectively lower, than that of τ_i. We use hep(i) and aff(i, j) as short notations for hep(i) = hp(i) $\cup$ $\{\tau_i\}$ and aff(i, j) = hep(i) $\cap$ lp(j). The latter denoting the set of intermediate tasks that may preempt τ_i but may themselves be preempted by some higher priority task τ_j.

The set of all memory blocks used by a task τ_i during its execution is given by $\mathbb{M}_i = \{m_{1,i}, m_{2,i}, ..., m_{z,i}\}$. For any memory reference, the L1 cache is always accessed. If a memory block is not available in the L1 cache but it is available in the L2 cache (i.e., a L1 miss but a L2 hit), that memory block will be first loaded into L1 from the L2. The time needed to load that block from L2 to L1 is given by d_{L1}. If the required memory block is not present in both levels, it will be loaded from the main memory to both cache levels. The time needed to load that block from the main memory to both cache levels is given by $d_{L1} + d_{L2}$, where d_{L2} denote the L2 cache miss penalty.

3. Background

In this section, we provide definitions for a number of key concepts and summarize the existing CRPD analyses for multilevel non-inclusive caches, which we later build upon.

Useful Cache Blocks (UCBs) [1]: A memory block $m_{x,i}$ of task τ_i is a UCB w.r.t a program point P, if $m_{x,i}$ is cached at P and $m_{x,i}$ may be reused at a program point Q that can be reached from P without eviction of $m_{x,i}$.

Evicting Cache Blocks (ECBs) [2]: All cache blocks used by a task during its execution are called its ECBs.

For single-level caches, CRPD is usually computed using the set of UCBs and ECBs of tasks.

Cache Related Preemption Delay (CRPD): CRPD is the additional execution time incurred by task τ_i in reloading its UCBs that may be evicted from the cache due to preemptions by higher priority tasks in hp(i). CRPD suffered by a task τ_i due to preemptions by a higher priority task $\tau_j \in$ hp(i) is usually denoted by $\gamma_{i,j}$.

3.1. State-of-the-art CRPD Analysis for Multilevel non-Inclusive Caches

To the best of our knowledge, the only existing work that focus on the CRPD analysis of multilevel non-inclusive caches is by Chattopadhyay et al. [6].

Chattopadhyay et al. [6] argued that due to the indirect effect of preemption the traditional UCB concept used to analyze CRPD for single-level caches is hard to use in case of multilevel caches. Consequently, they introduced an updated notion of UCBs in the context of two-level caches and used those UCBs to analyze CRPD for multilevel non-inclusive caches.

Definition 1 (Useful Cache Blocks (UCBs) in a two-level cache [6]). For a two-level cache, a memory block $m_{x,i}$ of task τ_i is considered a UCB w.r.t a program point P if (i) $m_{x,i}$ is cached at P in either L1, L2 or both and (ii) $m_{x,i}$ is reused at a program point Q that must be reached from P without eviction of $m_{x,i}$ from both L1 and L2 caches.

Based on the above definition, the CRPD analysis of [6] computes the set of UCBs of tasks using the Must-cache analysis [9, 10] and a backward flow analysis. Must-cache analysis (see [9, 10]) determines the set of memory blocks

that are in the cache at any given program point under all circumstances, i.e., the reference to such memory blocks will always be cache hits w.r.t that program point. The result of their UCB analysis is a tuple $CU^{P}_{m_{x,i}} = (CU^{P,1}_{m_{x,i}}, CU^{P,2}_{m_{x,i}})$ that captures the fixed-point on the *maximum LRU-age* of a memory block $m_{x,i}$ of task τ_i at a program point P in both L1 and L2 cache.

Maximum LRU-age [3]: The maximum LRU-age of memory block $m_{x,i}$ w.r.t a program point P (and cache level-l) is the sum of the maximal number of distinct accesses to l; from the last use of $m_{x,i}$ to program point P and from program point P to the next hit to $m_{x,i}$, under the constraint that $m_{x,i}$ is not evicted from level l cache before its reuse. If memory block $m_{x,i}$ is not present in the level-l cache at program point P then its maximum-LRU age $CU^{P,l}_{m_{x,i}} = \infty$.

To compute the set of ECBs of a preempting task τ_j, the analysis in [6] use the May-cache analysis [10]. The May-cache analysis (see [9, 10]) determines all memory blocks that may be in the cache at a given program point, i.e., it over-approximate the content of the cache w.r.t a program point. For any cache set S, the set of ECBs of the preempting task τ_j in S is computed by applying the May-cache analysis at the end point e of task τ_j. The May-cache state of cache set S at e will always include all possibly accessed memory blocks by τ_j in S. The output of the May-cache analysis w.r.t a cache set S is given by the tuple $May_{e,j}(S) = (May_{e,j,1}(S), May_{e,j,2}(S))$, where $May_{e,j,1}(S)/May_{e,j,2}(S)$ contain all memory blocks that may be cached in a L1/L2 cache set S during the execution of task τ_j.

If the mapping of a memory block $m_{x,i}$ of task τ_i in L1(L2) cache is defined by the tuple $S_{m_{x,i},1}(S_{m_{x,i},2})$ such that $S_{m_{x,i},1}$ (resp. $S_{m_{x,i},2}$) denote the cache set where $m_{x,i}$ is mapped in the L1 (resp. L2), then, the number of ECBs of a higher priority task $\tau_j \in \text{hp}(i)$ that may overlap with $m_{x,i}$ in L1(L2) cache are given by

$$|ECB_j^{S_{m_{x,i},1}}| = |May_{e,j,1}(S_{m_{x,i},1})| \quad \text{and} \quad |ECB_j^{S_{m_{x,i},2}}| = |May_{e,j,2}(S_{m_{x,i},2})| \tag{1}$$

where $ECB_j^{S_{m_{x,i},1}}(ECB_j^{S_{m_{x,i},2}})$ is the set of ECBs of task τ_j that map to the same L1(L2) cache set as $m_{x,i}$.

Using the maximum LRU-age of memory block $m_{x,i}$ and the set of ECBs of task $\tau_j \in \text{hp}(i)$ that may overlap with $m_{x,i}$ in the L1(L2) cache, the CRPD analysis in [6] determines if $m_{x,i}$ will be evicted from the L1(L2) cache in case of a preemption of τ_i by τ_j at a program point P. Formally, if

$$CU^{P,l}_{m_{x,i}} + |ECB_j^{S_{m_{x,i},l}}| \geq W_l \tag{2}$$

then memory block $m_{x,i}$ will be evicted from the level-l cache due to a preemption of task τ_i by any task $\tau_j \in \text{hp}(i)$ at a preemption point P.

It is shown in [3, 11] that in case of nested/multiple preemption of task τ_i by different tasks in hp(i), the CRPD cost can be computed by simulating nested preemptions, i.e., when computing the CRPD due to a single preemption of task τ_i by any higher priority task $\tau_j \in \text{hp}(i)$, it is assumed that τ_j has itself already been preempted by all higher priority tasks in hp(j). Therefore, to evaluate if a memory block $m_{x,i}$ will remain cached after nested/multiple preemptions of task τ_i by higher priority tasks in hp(i), Eq. (2) is adapted as follows

$$CU^{P,l}_{m_{x,i}} + \Big| \bigcup_{\forall h \in \text{hep}(j)} ECB_h^{S_{m_{x,i},l}} \Big| \geq W_l \tag{3}$$

where $\Big| \bigcup_{\forall h \in \text{hep}(j)} ECB_h^{S_{m_{x,i},l}} \Big|$ is the union of set of ECBs of all tasks in hep(j) that map to the same cache set as $m_{x,i}$.

3.1.1. Computing the Indirect Effect of Preemption

As we explained in the introduction, the indirect effect of preemption is an increase in the intra-task cache conflicts at a lower cache level, e.g., L2, due to eviction of memory blocks from a higher cache level, e.g., L1, in case of preemptions. Formally,

Definition 2 (**Indirect Effect of Preemption**). The indirect effect of preemption is the maximum increase in the age of a memory block $m_{x,i} \in \mathbb{M}_i$ in the level $l+1$ cache, that results from the eviction(s) of one or more memory block(s) from the level l cache due to preemptions.

From the analysis in [6], the indirect effect of preemption any memory block $m_{y,i} \in \mathbb{M}_i$ can suffer due to preemption of task τ_i by a higher priority task $\tau_j \in \text{hp}(i)$ at a preemption point P is given by $|ID^{r,P}_{m_{y,i}}|$, where $ID^{r,P}_{m_{y,i}}$ is a set comprising all memory blocks $m_{x,i} \in \mathbb{M}_i \setminus m_{y,i}$ that satisfy the following expression, i.e.,

$$ID^{r,P}_{m_{y,i}} = \Big\{ m_{x,i} | m_{x,i} \in D_{f,r} \wedge (S_{m_{x,i},2} == S_{m_{y,i},2}) \wedge CU^{P}_{m_{x,i}} \neq (\infty, \infty) \wedge CU^{P,1}_{m_{x,i}} + \Big| \bigcup_{\forall h \in \text{hep}(j)} ECB_h^{S_{m_{x,i},1}} \Big| \geq W_1 \Big\} \tag{4}$$

Eq. (4) states that any memory block $m_{x,i} \in \mathbb{M}_i$ can contribute to the indirect effect of preemption of another memory block $m_{y,i} \in \mathbb{M}_i \setminus m_{x,i}$ (accessed at any program point r after the preemption point P), if $m_{x,i}$ satisfies all conditions in Eq. (4). These conditions are explained below in detail.

- By definition only the memory blocks that are accessed from the L1 cache in absence of preemption, i.e., L1-hits, can cause the indirect effect of preemption. Therefore, for any memory block $m_{x,i}$ to contribute to the indirect effect of preemption of another memory block $m_{y,i}$, $m_{x,i}$ must have at least one reference categorized as a L1-hit before an access to $m_{y,i}$ at program point r, i.e., $m_{x,i} \in D_{f,r}$. Where $D_{f,r}$ is a set that contains all memory blocks of task τ_i that have at least one reference categorized as a L1-hit along any execution path of τ_i starting from the entry node of τ_i and ending at r. In [6] $D_{f,r}$ is computed using a forward-flow analysis and the Must-cache analysis [9]. The analysis starts from the entry point of task τ_i and ends at r, performing a set union of all memory blocks with L1-hits along the path.
- $S_{m_{x,i},2} == S_{m_{y,i},2}$ implies that any memory block $m_{x,i} \in \mathbb{M}_i \setminus m_{y,i}$ can only contribute to the indirect effect of preemption of any other memory block $m_{y,i} \in \mathbb{M}_i$ if both $m_{x,i}$ and $m_{y,i}$ are mapped to the same L2 cache set. This is intuitive, since in a set-associative cache each cache set can be analyzed independently. Therefore, any reference to memory block $m_{x,i}$ can only impact the ages of other memory blocks that are mapped to the same cache set as $m_{x,i}$.
- $CU^{P}_{m_{x,i}} \neq (\infty, \infty)$ states that any memory block $m_{x,i}$ can contribute to the indirect effect after a preemption at any program point P, if $m_{x,i}$ is categorized as a UCB at P, i.e., $m_{x,i}$ must be re-used after P. Moreover, $m_{x,i}$ must also be evicted from the L1 cache due to preemption, i.e., $CU^{P,1}_{m_{x,i}} + |ECB_j^{S_{m_{x,i},1}}| \geq W_1$. Again, this is intuitive, since $m_{x,i}$ can only impact the ages of other memory blocks (including $m_{y,i}$) in the L2 cache if $m_{x,i}$ is accessed either from L2 or the main memory after the preemption. Finally, the set $ID^{r,P}_{m_{y,i}}$ contains all memory blocks $m_{x,i} \in \mathbb{M}_i \setminus m_{y,i}$ that satisfy all the above mentioned conditions.

Considering that a memory block $m_{y,i}$ can be accessed at several different program locations (i.e., r) that are reachable from the preemption point P, the worst-case indirect effect of preemption that $m_{y,i}$ can suffer is computed by maximizing Eq. (4) over all program points reachable from P and where $m_{y,i}$ may be accessed. Let $\mathbb{R}$ denote the set of all such program locations then, the worst-case indirect effect $m_{y,i}$ may suffer due to a preemption at program point P is given by $ID^{max}_{m_{y,i},P}$, where

$$ID^{max,P}_{m_{y,i}} = \max_{\forall r \in \mathbb{R}} |ID^{r,P}_{m_{y,i}}| \tag{5}$$

3.1.2. CRPD Computation

Under the analysis of [6], the CRPD task τ_i may suffer due to a preemption by any higher priority task $\tau_j \in \text{hp}(i)$ at any arbitrary preemption point P is made of four components; $CRT^{P,1}_{i,j}$, $CRT^{P,2}_{i,j}$, $ICRT^{P,1}_{i,j}$, and $ICRT^{P,2}_{i,j}$.

$CRT^{P,1}_{i,j}$ captures the CRPD due to all those memory blocks of τ_i that are evicted from the L1 cache due to preemption by τ_j at program point P but may still be available in the L2 cache, i.e.,

$$CRT^{P,1}_{i,j} = d_{L1} \times |\mathbb{M}_{i,L_1}| \tag{6}$$

where $\mathbb{M}_{i,L_1}$ is a set of memory blocks defined as

$$\mathbb{M}_{i,L_1} = \Big\{ m_{x,i} | CU^{P}_{m_{x,i}} \neq (\infty,\infty) \wedge CU^{P,1}_{m_{x,i}} + \Big| \bigcup_{\forall h \in \text{hep}(j)} ECB_h^{S_{m_{x,i},1}} \Big| \geq W_1 \wedge CU^{P,2}_{m_{x,i}} + \Big| \bigcup_{\forall h \in \text{hep}(j)} ECB_h^{S_{m_{x,i},2}} \Big| + ID^{max,P}_{m_{x,i}} < W_2 \Big\} \tag{7}$$

Similarly, $CRT^{P,2}_{i,j}$ accounts for the CRPD due to all those memory blocks of τ_i that are evicted from both L1 and L2 caches due to preemption, i.e.,

$$CRT^{P,2}_{i,j} = (d_{L1} + d_{L2}) \times |\mathbb{M}_{i,L_1 L_2}| \tag{8}$$

where

$$\mathbb{M}_{i,L_1 L_2} = \Big\{ m_{x,i} | CU^{P}_{m_{x,i}} \neq (\infty,\infty) \wedge CU^{P,1}_{m_{x,i}} + \Big| \bigcup_{\forall h \in \text{hep}(j)} ECB_h^{S_{m_{x,i},1}} \Big| \geq W_1 \wedge CU^{P,2}_{m_{x,i}} + \Big| \bigcup_{\forall h \in \text{hep}(j)} ECB_h^{S_{m_{x,i},2}} \Big| + ID^{max,P}_{m_{x,i}} \geq W_2 \Big\} \tag{9}$$

All memory blocks in $\mathbb{M}_i$ with one or more references categorized as L2-hits during normal execution of task τ_i are not classified as UCBs according to Definition 1. Therefore, to account for the CRPD due to all such memory blocks,

the analysis in [6] checks all program locations with L2-hits after the preemption point P. The set of all program locations with L2 cache hit that are reachable from preemption point P is denoted by $\mathbb{P}_2$.

The analysis assumes that any reference to a memory block $m_{y,i} \in \mathbb{M}_i$ at any program point $r \in \mathbb{P}_2$, can suffer multiple L2-misses due to preemptions. The first reference to $m_{y,i}$ after preemption may result in a L2-miss due to the combined affect of the ECBs of the preempting task τ_j that map to the same cache set as $m_{y,i}$, and the indirect effect of preemption suffered by $m_{y,i}$ at that program point $r \in \mathbb{P}_2$, i.e., $|ID^{r,P}_{m_{y,i}}|$. Consequently, $ICRT^{P,1}_{i,j}$ captures the resulting CRPD cost for the first reference to memory block $m_{y,i}$ after preemption, i.e.,

$$ICRT^{P,1}_{i,j} = \sum_{r\in\mathbb{P}_2} \begin{cases} 0 & \text{if } MustAge(m_{y,i}, r, 2) + \left|\bigcup_{\forall h\in \text{hep}(j)} ECB_h^{S_{m_{y,i},2}}\right| + |ID^{r,P}_{m_{y,i}}| < W_2 \\ d_{L2} & \text{otherwise.} \end{cases} \tag{10}$$

where $MustAge(m_{y,i}, r, 2)$ is the LRU-age of memory block $m_{y,i}$ in the L2 cache immediately before the program point $r \in \mathbb{P}_2$ and is computed using the Must-cache analysis [10]. The CRPD analysis in [6] assume that the same memory block, i.e., $m_{y,i}$, can also be evicted solely due to the indirect effect of preemption. Consequently, $ICRT^{P,2}_{i,j}$ captures the CRPD cost due to references to $m_{y,i}$ that may suffer an L2 cache miss penalty after preemption only due to the indirect effect of preemption, where $ICRT^{P,2}_{i,j}$ is given as

$$ICRT^{P,2}_{i,j} = IL2_{ind} \times \sum_{r\in\mathbb{P}_2} \begin{cases} 0 & \text{if } MustAge(m_{y,i}, r, 2) + |ID^{r,P}_{m_{y,i}}| < W_2 \\ d_{L2} & \text{otherwise.} \end{cases} \tag{11}$$

In Eq. (11), $IL2_{ind}$ denotes an upper bound on the number of L2 cache misses to any memory reference solely due to the indirect effect of preemption. It is proved in [6] that if the cache configuration is such that $|\mathbb{S}_1| \leq |\mathbb{S}_2|$ and $W_1 \leq W_2$ then the value of $IL2_{ind}$ is upper bounded by 1, i.e., $IL2_{ind} \leq 1$ (see Theorem 5.3 of [6]).

Finally, the worst-case CRPD suffered by task τ_i due to a preemption by task $\tau_j \in \text{hp}(i)$ is given by maximizing Eq. (6)-(11) over the set of all program point $\mathbb{P}$, i.e.,

$$\gamma^H_{i,j} = \max_{P\in\mathbb{P}} \left(CRT^{P,1}_{i,j} + CRT^{P,2}_{i,j} + ICRT^{P,1}_{i,j} + ICRT^{P,2}_{i,j} \right) \tag{12}$$

Readers are directed to [6] for details on the formulation of Eq. (1)-(12).

4. Multi-level Useful Cache Blocks

The state-of-the-art definition of UCBs for multilevel caches(i.e., Definition 1) states that any memory block $m_{x,i}$ of task τ_i can only be categorized as a UCB w.r.t a program point P, if $m_{x,i}$ is not evicted from both L1 and L2 before being reused at a later program point Q. However, considering that multilevel non-inclusive caches do not strictly enforce content inclusion, it is likely that memory block $m_{x,i}$ can be available in only one cache level (e.g., L1 or L2) at program point Q and hence will be re-used from that cache level. For example, in Fig. 1a both memory blocks A and m are cached in L1 and L2 cache at program point P, however, only memory block A remains cached in both L1 and L2 before its next reuse. Therefore, when using Definition 1 only memory block A will be categorized as a UCB at program point P. However, we can see in Fig. 1a that memory block m is only evicted from L1 cache and is later reused from the L2 cache. In a non-inclusive multilevel cache, a memory block can be available in any of the cache levels. Therefore, using this insight, we can re-define the notion of UCBs for multilevel caches based on the cache level from which a memory block might be reused. Specifically, in a memory hierarchy with two cache levels a memory block $m_{x,i}$ can be categorized as a L1- or L2-UCB as follows:

Definition 3 (L1-Useful Cache Blocks (L1-UCBs)). A memory block $m_{x,i}$ of task τ_i is a L1-UCB w.r.t a program point P if (i) $m_{x,i}$ is certainly cached in L1 at P and (ii) $m_{x,i}$ is reused at a program point Q that must be reachable from P without eviction of $m_{x,i}$ from the L1 cache, i.e., the reference to $m_{x,i}$ at program point Q should be categorized as a L1 hit. The set of memory blocks of task τ_i categorized as L1-UCBs w.r.t a program point P is given by $UCB^P_{i,1}$.

Definition 4 (L2-Useful Cache Blocks (L2-UCBs)). A memory block $m_{y,i}$ of task τ_i is a L2-UCB w.r.t a program point P if (i) $m_{y,i}$ is certainly cached at P in L2, (ii) $m_{y,i}$ is reused at a program point Q that must be reachable from P without eviction of $m_{y,i}$ from the L2 cache, i.e., the reference to $m_{y,i}$ at program point Q is a L2 cache hit, and (iii) $m_{y,i}$

is not a L1-UCB w.r.t P, i.e., if the reference to $m_{y,i}$ at program point Q is always a cache hit in both L1 and L2, then, $m_{y,i}$ is not a L2-UCB but a L1-UCB instead. The set of memory blocks of task τ_i categorized as L2-UCBs w.r.t P is denoted by $UCB^P_{i,2}$.

It is argued in the existing works [6, 7] that the concept of UCBs is difficult to use for the analysis of CRPDs for multilevel cache. However, by categorizing UCBs based on the cache level from which they might be re-used, the UCB concept can be used to compute CRPD for multilevel caches.

4.1. Finding L1/L2-UCBs

The set of L1- and L2-UCBs of a task τ_i w.r.t a program point P can be determined by using the UCB analysis proposed in [6], i.e., by using the Must-cache analysis [10] along with a backward flow analysis. As mentioned earlier, the result of the UCB analysis in [6] is a tuple $CU^P_{m_{x,i}} = (CU^{P,1}_{m_{x,i}}, CU^{P,2}_{m_{x,i}})$ that captures the fixed-point on the maximum LRU-age of a memory block $m_{x,i}$ of task τ_i w.r.t a program point P. Consequently, the set of L1-UCBs of a task τ_i w.r.t a program point P can be computed as follows

$$UCB^P_{i,1} = \left\{ m_{x,i} | m_{x,i} \in \mathbb{M}_i \wedge CU^{P,1}_{m_{x,i}} \neq \infty \right\} \tag{13}$$

Eq. (13) implies that given a program point P, all memory blocks $m_{x,i} \in \mathbb{M}_i$ with a maximum LRU-age in L1 $\neq \infty$ are L1-UCBs of τ_i w.r.t P.

Similarly, the set of L2-UCBs of task τ_i w.r.t a program point P can be computed as follows

$$UCB^P_{i,2} = \left\{ m_{y,i} | m_{y,i} \in \mathbb{M}_i \wedge \left(CU^{P,1}_{m_{y,i}} = \infty \wedge CU^{P,2}_{m_{y,i}} \neq \infty \right) \right\} \tag{14}$$

Eq. (14) states that any memory block $m_{y,i} \in \mathbb{M}_i$ is a L2-UCBs w.r.t a program point P, if $m_{y,i}$ is evicted from L1 cache along at least one path from P to some other program point Q accessing $m_{y,i}$, i.e., $CU^{P,1}_{m_{y,i}} = \infty$, but it remains cached in L2 along any path to such program point Q, i.e., $CU^{P,2}_{m_{y,i}} \neq \infty$.

To compute the number of ECBs of a higher priority task $\tau_j \in \text{hp}(i)$ that may map to the same L1(L2) cache set as the L1/L2-UCBs of task τ_i, we use the May-cache analysis [10] and Eq. (1).

5. Tightening the Bound on the Indirect Effect of Preemption

The existing approach to calculate the indirect effect of preemption (i.e., Eq. (4)) is sound. However, it may be pessimistic due to an over-approximation of the set of memory blocks that can cause the indirect effect of preemption.

Example 1. *Fig. 2 shows a sequence of memory references (from left to right) during the non-preempted (left) and preempted (right) execution of task τ_i. We assume that L1 and L2 are two-way set-associative LRU-caches, i.e., $W_1 = W_2 = 2$. All memory blocks used by task τ_i, i.e., A, B and m, are mapped to the same L1 and L2 cache set. For clarity, we only focus on the computation of indirect effect of preemption suffered by memory block m due to a preemption at program point P. We can see in Fig. 2 that the second reference to memory block m is a L2-hit in both*

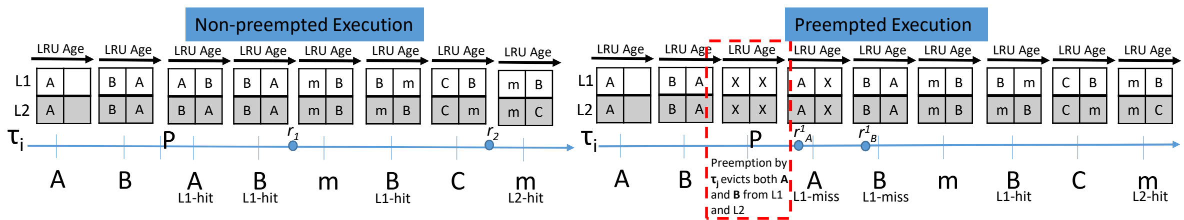


Figure 2: Highlighting the pessimism in the calculation of indirect effect of preemption by [6].

the non-preempted and preempted execution of τ_i. Now, if we use the analysis of [6] we get $D_{f,r_1} = D_{f,r_2} = \{A, B\}$ w.r.t program locations r_1 and r_2 where m is accessed after the preemption. Also, memory block A and B satisfy all the other constraints in Eq. (4)*. Therefore, the analysis in [6] concludes that memory blocks A and B can both cause an indirect*

effect of preemption on memory block m, i.e., $ID_m^{r_1,P} = ID_m^{r_2,P} = \{A, B\}$ *and* $ID_m^{max,P} = 2$*. Consequently, Eq.* (11) *concludes to the eviction of m from L2 due to the indirect effect of preemption, i.e.,* $MustAge(m, r_2, 2) + |ID_m^{r_2,P}| = 2 + 2 > W_2$*. However, we can see in Fig. 2 that this is not true. The second reference to memory block m remains a L2 cache hit even after preemption.*

Example 1 shows that [6] overestimates the indirect effect of preemption which may lead to pessimistic CRPD values. Therefore, we propose an improved analysis that computes a tighter bound on the indirect effect of preemption using Algorithm 1. The algorithm computes the indirect effect of preemption of every memory block $m_{y,i} \in \mathbb{M}_i$ of task τ_i due to a preemption at program point P by upper bounding the set of memory blocks that can cause the indirect effect of preemption given by $Ind^P_{m_{y,i}}$. We prove the correctness of Algorithm 1 using the following Lemma

Algorithm 1 Returns the set of memory blocks of task τ_i that can cause an indirect effect of preemption to a memory block $m_{y,i} \in \mathbb{M}_i$, when τ_j preempts τ_i at a program point P

Output: The indirect effect of preemption suffered by every memory block $m_{y,i} \in \mathbb{M}_i$ due to preemption at program point P, i.e., $Ind^P_{m_{y,i}}$.

1: **for** $\forall m_{y,i} \in \mathbb{M}_i$ **do**
2: $\quad Ind^P_{m_{y,i}} := \emptyset$
3: **end for**
4: **for** $\forall m_{x,i} \in UCB^P_{i,1}$ **do**
5: $\quad$ **if** $CU^{P,1}_{m_{x,i}} + \left|\bigcup_{\forall h \in \text{hep}(j)} ECB_h^{S_{m_{x,i},1}}\right| \geq W_1$ **then**
6: $\quad\quad FA^P_{m_{x,i}} := GetFirstAccess(m_{x,i}, P)$
7: $\quad\quad$ **for** $\forall m_{y,i} \in \mathbb{M}_i \setminus m_{x,i}$ **do**
8: $\quad\quad\quad$ **if** $((MustAge(m_{y,i}, FA^P_{m_{x,i}}, 2) \neq \infty) \wedge (S_{m_{y,i},2} == S_{m_{x,i},2}) \wedge (MustAge(m_{x,i}, FA^P_{m_{x,i}}, 2) > MustAge(m_{y,i}, FA^P_{m_{x,i}}, 2))$ **then**
9: $\quad\quad\quad\quad Ind^P_{m_{y,i}} := Ind^P_{m_{y,i}} \cup m_{x,i};$
10: $\quad\quad\quad$ **end if**
11: $\quad\quad$ **end for**
12: $\quad$ **end if**
13: **end for**

Lemma 1. The indirect effect of preemption that can be suffered by any memory block $m_{y,i}$ of task τ_i, when τ_i is preempted by a higher priority task $\tau_j \in \text{hp}(i)$ at any program point P is upper bounded by $|Ind^P_{m_{y,i}}|$.

Proof. We prove that the cardinality of the set $Ind^P_{m_{y,i}}$ (computed using Algorithm 1) is an upper bound on the indirect effect of preemption that can be suffered by any memory block $m_{y,i}$ due to a preemption by any task $\tau_j \in \text{hp}(i)$ at a program point P.
(1). By definition, the indirect effect of preemption is caused by memory blocks of task τ_i that are fetched from the L1 cache during the normal execution of τ_i but are fetched from the L2 cache or the main memory after preemption. Therefore, the set of memory blocks that can cause the indirect effect after a preemption of τ_i at point P is upper bounded by the set of L1-UCBs of τ_i at P, i.e., $UCB^P_{i,1}$. Thus, the external loop (i.e., lines 4 to 13) of Algorithm 1 only checks memory blocks in $UCB^P_{i,1}$.
(2). A L1-UCB $m_{x,i} \in UCB^P_{i,1}$ can cause an indirect effect of preemption on any other memory block $m_{y,i} \in \mathbb{M}_i$ at a preemption point P, only if $m_{x,i}$ is evicted from the L1 cache due the preemption at P. As shown in Equation 3, this may happen only if $CU^{P,1}_{m_{x,i}} + |\bigcup_{\forall h \in \text{hep}(j)} ECB_h^{S_{m_{x,i},1}}| \geq W_1$, which is checked at line 5 of Algorithm 1.
(3). In case of a preemption at program point P, the first reachable references to a memory block $m_{x,i} \in UCB^P_{1,i}$ after P determines whether $m_{x,i}$ is still in the L1 cache or not. When computing the indirect effect $m_{x,i}$ can cause on other memory blocks, only considering the first reference to $m_{x,i}$ after P is sufficient because of two possible scenarios; (i) if the first reference to $m_{x,i}$ after preemption is still a L1-hit, then, by definition, $m_{x,i}$ will not cause any indirect effect of preemption; and (ii) if the first reference to $m_{x,i}$ after preemption results in a L1-miss, then, $m_{x,i}$ will be reloaded in the L1 either from L2 or main memory. Therefore, only in case (ii), $m_{x,i}$ can contribute to the indirect effect of other memory blocks. In Algorithm 1, the function $GetFirstAccess(m_{x,i}, P)$ (line 6) is used to determine the first reachable reference to every $m_{x,i} \in UCB^P_{1,i}$, i.e., given by program point $FA^P_{m_{x,i}}$.
(4). If the access to $m_{x,i}$ at $FA^P_{m_{x,i}}$ is a L1-miss, then, $m_{x,i}$ will be fetched either from L2 or main memory. However, in both cases, $m_{x,i}$ can cause an indirect effect of preemption only on memory blocks that are already in the L2 cache

at $FA^P_{m_{x,i}}$, i.e., reloading $m_{x,i}$ from the main memory or from the L2 cache can only increase the LRU-age of memory blocks that are already in L2 cache at $FA^P_{m_{x,i}}$. This is considered by the nested loop (lines 7 to 11) in Algorithm 1. The nested loop determines all memory blocks $m_{y,i} \in \mathbb{M}_i \setminus m_{x,i}$ that can suffer an indirect effect of preemption due to $m_{x,i}$. The computation is performed using the following three conditions:

(4.1). $m_{x,i}$ can only cause an indirect effect of preemption on any other memory block $m_{y,i} \in \mathbb{M}_i \setminus m_{x,i}$, if $m_{y,i}$ is in the L2 cache at $FA^P_{m_{x,i}}$. This is determined using the Must-age [10] of $m_{y,i}$ at program point $FA^P_{m_{x,i}}$, i.e., $m_{y,i}$ can only suffer an indirect effect of preemption due to $m_{x,i}$ if ($MustAge(m_{y,i}, FA^P_{m_{x,i}}, 2) \neq \infty$). Otherwise, the additional access to $m_{x,i}$ in L2 after preemption does not increase the LRU-age of $m_{y,i}$ in L2.

(4.2). $m_{x,i}$ can cause an indirect effect of preemption on $m_{y,i}$ if both $m_{x,i}$ and $m_{y,i}$ map to the same L2 cache set, i.e., $S_{m_{x,i},2} == S_{m_{y,i},2}$. Otherwise, $m_{x,i}$ can not interfere with $m_{y,i}$.

(4.3). $m_{x,i}$ can only cause an indirect effect on $m_{y,i}$ if the access to $m_{x,i}$ at $FA^P_{m_{x,i}}$ is (i) a L2-miss or (ii) a L2-hit and the LRU-age of $m_{x,i}$ in L2 is greater than the LRU-age of $m_{y,i}$ in L2. The condition $MustAge(m_{x,i}, FA^P_{m_{x,i}}, 2) > MustAge(m_{y,i}, FA^P_{m_{x,i}}, 2)$ accounts for both these scenarios explained as follows. If the access to $m_{x,i}$ at $FA^P_{m_{x,i}}$ is a L2-miss, then, $MustAge(m_{x,i}, FA^P_{m_{x,i}}, 2) = \infty > MustAge(m_{y,i}, FA^P_{m_{x,i}}, 2)$. Hence, $m_{x,i}$ will be reloaded from the main memory to both L2 and L1 cache, which will increase the LRU-age of all existing memory blocks in the L2 cache including $m_{y,i}$. Similarly, if the access to $m_{x,i}$ at $FA^P_{m_{x,i}}$ is a L2-hit, then $m_{x,i}$ will be reloaded from L2 to L1 cache and the access to $m_{x,i}$ will only change the LRU-ages of memory blocks that are younger than $m_{x,i}$ in the L2 cache. So, $m_{y,i}$ will only suffer an indirect effect of preemption due to $m_{x,i}$ if $MustAge(m_{x,i}, FA^P_{m_{x,i}}, 2) > MustAge(m_{y,i}, FA^P_{m_{x,i}}, 2)$ holds.

If all the above conditions hold for any memory block $m_{y,i} \in \mathbb{M}_i \setminus m_{x,i}$ then, $m_{x,i}$ will be added to the set of memory blocks that can cause an indirect effect of preemption on $m_{y,i}$, i.e., $Ind^P_{m_{y,i}} \cup m_{x,i}$ (line 9). Finally, after iterating over all memory blocks $m_{x,i} \in UCB^P_{i,1}$, the set $Ind^P_{m_{y,i}}$ will contain all memory blocks that can cause the indirect effect of preemption on $m_{y,i}$ and the cardinality of the set $Ind^P_{m_{y,i}}$ will upper bound the maximum increase in the LRU-age of memory block $m_{y,i} \in \mathbb{M}_i$ due to the indirect effect of preemption at P. □

Note that the worst-case time complexity of Algorithm 1 is quadratic w.r.t the number of memory blocks used by tasks.

Example 2. *We use Algorithm 1 to compute the indirect effect of preemption of memory block m in Fig. 2. From the UCB analysis in Section 4, it results that both memory blocks A and B are L1-UCBs w.r.t program point P, i.e.,* $\{A, B\} \in UCB^P_{i,1}$*. Also, both A and B will be evicted from L1 due to preemption at P and they also map to the same L2 cache set as memory block m. However, m is not in the L2 cache at the first access of A (and B) after the preemption point P, i.e.,* $MustAge(m, FA^P_A, 2) = \infty$ *(and* $MustAge(m, FA^P_B, 2) = \infty$*). Therefore, Algorithm 1 concludes that the indirect effect of preemption of m is 0 , i.e.,* $|Ind^P_m| = 0$*, which can also be confirmed in Fig. 2.*

6. Improved CRPD Analysis for Multilevel caches

Having bounded the indirect effect of preemption in the previous section, in this section, we will demonstrate how to compute the CRPD for multilevel non-inclusive caches using the notion of multilevel UCBs.

6.1. CRPD due to the Eviction of L1-UCBs

To compute the CRPD due to the eviction of L1-UCBs of task τ_i in case of a preemption by any higher priority task $\tau_j \in \text{hp}(i)$ at any arbitrary program point P, we use a similar approach as presented in [6] (i.e., Eq. (6) and (8)).

L1-UCBs of task τ_i can be evicted from the L1 cache because of preemption but may still be available in the L2 cache. We use $\gamma^{P,L_1}_{i,j}$ to denote the CRPD cost due to all those L1-UCBs of task τ_i, where $\gamma^{P,L_1}_{i,j}$ is computed as follows:

$$\gamma^{P,L_1}_{i,j} = \left|\left\{m_{x,i} | m_{x,i} \in UCB^P_{i,1} \wedge CU^{P,1}_{m_{x,i}} + \Big|\bigcup_{\forall h \in \text{hep}(j)} ECB_h^{S_{m_{x,i},1}}\Big| \geq W_1 \wedge CU^{P,2}_{m_{x,i}} + \Big|\bigcup_{\forall h \in \text{hep}(j)} ECB_h^{S_{m_{x,i},2}}\Big| + |Ind^P_{m_{x,i}}| < W_2\right\}\right| \times d_{L1} \quad (15)$$

Eq. (15) states that all L1-UCBs of task τ_i that are evicted from the L1-cache due to preemption (i.e., their LRU-age in L1 plus the maximum interference by higher or equal priority tasks is larger than or equal to the L1-associativity) but are still available in the L2-cache (i.e., LRU-age in L2 plus interference is smaller than the L2-associativity) will only

incur a L1-miss penalty. Also, note that when computing the L2 cache conflicts to a L1-UCB $m_{x,i} \in UCB^{P}_{1,i}$, Eq. (15) also considers the indirect effect that may be suffered by $m_{x,i}$, i.e., $|Ind^{P}_{m_{x,i}}|$, computed using Algorithm. 1.

Similarly, some L1-UCBs of task τ_i might be evicted from both L1 and L2 caches due to preemptions. We use $\gamma^{P,L_{12}}_{i,j}$ to denote the CRPD cost due to all those L1-UCBs of task τ_i, where $\gamma^{P,L_{12}}_{i,j}$ is computed as follows:

$$\gamma^{P,L_{12}}_{i,j} = \left| \left\{ m_{x,i} | m_{x,i} \in UCB^{P}_{i,1} \wedge CU^{P,1}_{m_{x,i}} + \Big| \bigcup_{\forall h \in \text{hep}(j)} ECB^{S_{m_{x,i},1}}_{h} \Big| \geq W_1 \wedge CU^{P,2}_{m_{x,i}} + \Big| \bigcup_{\forall h \in \text{hep}(j)} ECB^{S_{m_{x,i},2}}_{h} \Big| + |Ind^{P}_{m_{x,i}}| \geq W_2 \right\} \right| \times (d_{L1} + d_{L2}) \quad (16)$$

Eq. (16) implies that all L1-UCBs of task τ_i that are evicted from both L1 and L2 cache due to preemption, will be loaded from the main memory to both cache levels. The total time to load one block from the main memory to both cache levels is given by $(d_{L1} + d_{L2})$.

6.2. CRPD due to the Eviction of L2-UCBs

L2 UCBs of task τ_i can be evicted from the cache in several ways; (i) directly due to interference by the preempting tasks or (ii) due to the indirect effect of preemption or (iii) due to a combination of both (i) and (ii). However, before presenting our analysis to compute the CRPD due to the eviction of L2-UCBs of tasks, we highlight a source of pessimism in the CRPD analysis of [6]. This pessimism lies in Eq. (10) and (11) that are used to compute the CRPD due to memory references that were L2 cache hits during the normal execution of a task τ_i but may become L2 cache misses after the preemption of τ_i. To illustrate, consider the following example.

Example 3. *We calculate the CRPD costs $ICRT^{P,1}_{i,j}$ and $ICRT^{P,2}_{i,j}$, i.e., respectively using Eq.* (10) *and* (11)*, for the example scenario shown in Fig. 3. We have $W_1 = 2$ and $W_2 = 3$ and we assume that all memory blocks used by task τ_i, i.e., A,B,C,D and m, map to the same L1/L2 cache set. We can see in Fig. 3 that three references to memory block m are L2 cache hits during the normal/non-preempted execution of task τ_i, i.e., the references at program points r_1, r_2 and r_3. Therefore, the set of program locations with L2 cache hits w.r.t to program point P, i.e., $\mathbb{P}_2$, is given by $\mathbb{P}_2 = \{r_1, r_2, r_3\}$. The Must-age of m at r_1, r_2 and r_3 is $MustAge(m, r_1, 2) = MustAge(m, r_2, 2) = MustAge(m, r_3, 2) = 2$. The number of ECBs of the preempting task, i.e., τ_j, that map to the same L2 cache set as m are given by $|ECB^{S_{m,2}}_{j}| = 2$. Similarly, using Eq.* (4) *to calculate the indirect effect of preemption suffered by m at r_1, r_2 and r_3, we get $|ID^{r_1,P}_{m}| = 0$ and $|ID^{r_2,P}_{m}| = |ID^{r_3,P}_{m}| = 1$.Note that since we assume $|\mathbb{S}_1| \leq |\mathbb{S}_2|$ and $W_1 \leq W_2$ we use $IL2_{ind} = 1$ in Eq.* (11) *(see Theorem 5.3 in [6]).*

Finally, by inserting all required values in Eq. (10)*, we will get $MustAge(m, r, 2) + |ECB^{S_{m,2}}_{j}| + |ID^{r,P}_{m}| > W_2$ for every $r \in \mathbb{P}_2$. Hence, the resulting value of $ICRT^{P,1}_{i,j}$ will be $3 \times d_{L2}$. Similarly, using the values of Must-age and the indirect effect of preemption of m in Eq.* (11) *we have $MustAge(m, r, 2) + |ID^{r,P}_{m}| > W_2$ for r_2 and r_3 in $\mathbb{P}_2$. Therefore, the resulting value of $ICRT_{P,2}$ will be $2 \times d_{L2}$. Consequently, the total CRPD cost due to L2 cache misses resulting from preemption is calculated to be $ICRT^{P,1}_{i,j} + ICRT^{P,2}_{i,j} = (3 + 2) \times d_{L2} = 5 \times d_{L2}$. However, we can see from the preempted execution scenario shown in Fig. 3 that this bound on the CRPD is very pessimistic and the actual CRPD cost due to all references to memory block m after preemption is only $2 \times d_{L2}$.*

The analysis of [6] overestimates the CRPD due to L2 cache misses resulting from preemptions because it assumes that all memory references that were L2 cache hit during the normal execution of a task may be impacted both directly and indirectly due to preemptions, i.e., it evaluates Eq. (10) and (11) for all program locations with L2 cache hits. Although, it is true that multiple references to the same memory block may result in L2 cache hits, e.g., if the memory block is accessed in a loop or for a particular execution scenario shown in Fig. 3, however, not all those references can be impacted both directly and indirectly due to preemptions.

To reduce the above mentioned pessimism in the existing analysis [6], the proposed analysis focuses on bounding the number of references to memory blocks that can be impacted directly/indirectly due to preemptions. We start by computing the set $\hat{UCB}^{P}_{i,2}$ of L2-UCBs of a task τ_i *reachable* from a program point P. $\hat{UCB}^{P}_{i,2}$ is the set of memory blocks that have at least one reference categorized as a L2 cache hit, starting from the program point under analysis, i.e., P, until the end point e of task τ_i. Formally,

$$\hat{UCB}^{P}_{i,2} = \bigcup_{\forall r \in \mathbb{P}_2} UCB^{r}_{i,2} \quad (17)$$

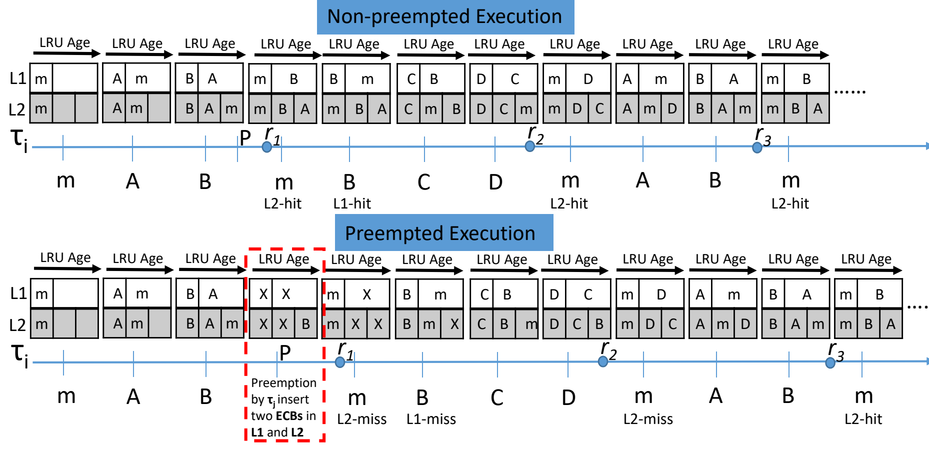


Figure 3: Example scenario to demonstrate the pessimism of [6] when calculating the CRPD due to L2 cache misses resulting from preemption.

where $\mathbb{P}_2$ is the set of all program locations between P and e with L2 cache hits.

Let $\hat{UCB}^P_{i,2} = \{m_{1,i}, m_{2,i}, m_{3,i}, ..., m_{n,i}\}$ be the result of Eq. (17), then for every memory block $m_{y,i}$ in $\hat{UCB}^P_{i,2}$ we define a set $\mathbb{R}^P_{m_{y,i}}$ that contains all program locations after the preemption point P where a reference to $m_{y,i}$ is a L2 cache hit, i.e., $\mathbb{R}^P_{m_{y,i}} = \{R^{1,P}_{m_{y,i}}, R^{2,P}_{m_{y,i}}, ..., R^{k,P}_{m_{y,i}}\}$. The rationale of defining $\mathbb{R}^P_{m_{y,i}}$ is to investigate how many references to memory block $m_{y,i}$ can be impacted directly or indirectly due to preemptions and therefore may contribute to CRPD. The analysis in [6] assumes that a memory block $m_{y,i} \in \hat{UCB}^P_{i,2}$ can be impacted both directly and indirectly at all program points in $\mathbb{R}^P_{m_{y,i}}$. However, in this work, we prove that in fact for any memory block $m_{y,i} \in \hat{UCB}^P_{i,2}$ it is only the first reference to $m_{y,i}$ after P, i.e., at $R^{1,P}_{m_{y,i}}$, that can be directly impacted due to preemptions. All subsequent references to $m_{y,i}$ after $R^{1,P}_{m_{y,i}}$, i.e., at $\{R^{2,P}_{m_{y,i}}, ..., R^{k,P}_{m_{y,i}}\}$, can only be impacted due to the indirect effect of preemption.

Lemma 2. Given an arbitrary preemption point P and a memory block $m_{y,i} \in \hat{UCB}^P_{i,2}$, it is only the first reference to $m_{y,i}$ after P, i.e., at $R^{1,P}_{m_{y,i}}$, that can be directly impacted due to preemption at P. All subsequent references to $m_{y,i}$ after $R^{1,P}_{m_{y,i}}$, i.e., at $\{R^{2,P}_{m_{y,i}}, ..., R^{k,P}_{m_{y,i}}\}$, can only be impacted due to the indirect effect of preemption suffered by $m_{y,i}$ w.r.t P.

Proof. By definition if $m_{y,i} \in \hat{UCB}^P_{i,2}$ then the reference to memory block $m_{y,i}$ at program point $R^{1,P}_{m_{y,i}}$ will be a L2 cache hit during the normal execution of task τ_i. So, after an access to $m_{y,i}$ at $R^{1,P}_{m_{y,i}}$, $m_{y,i}$ will be the youngest element in both L1 and L2 caches. Now, after the preemption of τ_i at P, the access to $m_{y,i}$ at $R^{1,P}_{m_{y,i}}$ may result in a L2 cache hit or miss. If the reference to $m_{y,i}$ at $R^{1,P}_{m_{y,i}}$ is still a L2 cache hit after preemption, $m_{y,i}$ will become the youngest element in L1 and L2 cache after $R^{1,P}_{m_{y,i}}$. Hence, the sate of L1 and L2 cache w.r.t $m_{y,i}$ will be the same as during the normal execution of τ_i. Similarly, even if the reference to $m_{y,i}$ at $R^{1,P}_{m_{y,i}}$ becomes a L2 cache miss after preemption, $m_{y,i}$ will be reloaded from the main memory into both L1 and L2 caches at $R^{1,P}_{m_{y,i}}$. Again, this will make $m_{y,i}$ the youngest element in both L1 and L2 caches after $R^{1,P}_{m_{y,i}}$. Therefore, the direct impact of preemption on $m_{y,i}$ will be neutralized after a L2 cache hit/miss at $R^{1,P}_{m_{y,i}}$ and all subsequent references to $m_{y,i}$ can only be impacted due to the indirect effect of preemption suffered by $m_{y,i}$ w.r.t the preemption point P. □

Using Lemma 2 we can compute the CRPD cost task τ_i may endure due to the eviction of one of it's L2-UCB, after a preemption by any higher priority task $\tau_j \in \text{hp}(i)$ at a program point P. We use two equations to compute that CRPD cost, i.e., Eq. (18) and (19). Eq. (18) computes the CRPD cost due to the first reference to memory block $m_{y,i} \in UCB^P_{i,2}$ after the preemption point P, whilst Eq. (19) computes the CRPD cost due to all subsequent reference to $m_{y,i}$

$$\gamma^{P,L2,first}_{m_{y,i},j} = \begin{cases} 0 & \text{if } MustAge(m_{y,i}, R^{1,P}_{m_{y,i}}, 2) + \left|\bigcup_{\forall h \in \text{hep}(j)} ECB^{S_{m_{y,i},2}}_h\right| + |Ind^P_{m_{y,i}}| < W_2 \\ d_{L2} & \text{otherwise.} \end{cases} \tag{18}$$

$$\gamma^{P,L2,next}_{m_{y,i},j} = \sum_{\forall r \in \mathbb{R}^{P}_{m_{y,i}} \setminus R^{1,P}_{m_{y,i}}} \begin{cases} 0 & \text{if } MustAge(m_{y,i}, r, 2) + |Ind^{P}_{m_{y,i}}| < W_2 \\ d_{L2} & \text{otherwise.} \end{cases} \tag{19}$$

It is straightforward to see that if any reference to a memory block $m_{y,i} \in \hat{UCB}^{P}_{i,2}$ is not in a loop, it can contribute only once to the CRPD. However, the problem emerges when one or more references to memory block $m_{y,i}$ are inside a loop, in which case we need to bound how many times each reference to $m_{y,i}$ can contribute to the CRPD.

Lemma 3. Any reference to a memory block $m_{y,i} \in UCB^{P}_{i,2}$ which is inside a loop, e.g., $Ref(m_{y,i})$, can contribute at most two L2 cache misses to the CRPD suffered by task τ_i due to preemption at a program point P.

Proof. Any reference to a memory block $m_{y,i} \in \hat{UCB}^{P}_{i,2}$ which is inside a loop, e.g., $Ref(m_{y,i})$, can be classified as a L2-hit in the absence of preemption w.r.t a program point P under two conditions; (1) if there are fewer than W_2 conflicting L2 memory blocks accessed between two references to $m_{y,i}$ w.r.t to P and (2) if at least W_2 conflicting L2 memory blocks are accessed between two access to $m_{y,i}$ w.r.t P, however there exist at least one memory reference which is a L1 cache hit, i.e., it does not generate an L2 cache conflict to $m_{y,i}$. If (1) is true, then the reference $Ref(m_{y,i})$ can only be a L2-miss after preemption if $m_{y,i}$ is directly evicted due to preemption at P, and $Ref(m_{y,i})$ can suffer at most one L2-miss after preemption. Also, if (2) holds then the reference $Ref(m_{y,i})$ may also become a L2-miss after preemption due to the indirect effect of preemption caused by memory references that were L1-hits in the absence of preemption but may access the L2 after preemption. However, it is proved in [6] that if the cache configuration is such that $|\mathbb{S}_1| \leq |\mathbb{S}_2|$ and $W_1 \leq W_2$, then, any memory reference that was a L2 cache hit in the absence of preemption, e.g., $Ref(m_{y,i})$, can lead to at most one L2-miss due to the indirect effect after preemption. Knowing that $Ref(m_{y,i})$ is in a loop so both (1) and (2) can be true along different paths between P to $Ref(m_{y,i})$. Therefore, if reference $Ref(m_{y,i})$ is inside a loop, it may cause up to two L2 cache misses after preemption. □

We also know from Lemma 2 that after a preemption at any program point P, it is only the first reference to memory block $m_{y,i}$ after P, i.e., at program point $R^{1,P}_{m_{y,i}}$, that can be directly impacted by the preemption. This leads to the following Lemma

Lemma 4. Any reference to a memory block $m_{y,i} \in \hat{UCB}^{P}_{i,2}$ which is inside a loop, i.e., $Ref(m_{y,i})$, can cause up to two L2 cache misses after a preemption at any program point P only when $Ref(m_{y,i})$ is the first reference to $m_{y,i}$ after P.

Proof. We prove this lemma by contradiction. Let us assume there exist a memory reference to $m_{y,i}$, e.g., $\hat{Ref}(m_{y,i})$, which is inside a loop and can cause up to two L2 cache misses after the preemption at P but $\hat{Ref}(m_{y,i})$ is not the first reference to $m_{y,i}$ after the preemption point P.

For $\hat{Ref}(m_{y,i})$ to cause two L2 cache misses after the preemption at P, there must be at least two paths reachable to $\hat{Ref}(m_{y,i})$ after P where $m_{y,i}$ will be evicted either directly or indirectly. However, we know from Lemma 2 that it is only the first reference of $m_{y,i}$ after preemption, i.e., at $R^{1,P}_{m_{y,i}}$, that can cause an L2 cache miss directly due to a preemption at P and all subsequent references to $m_{y,i}$ after the program point $R^{1,P}_{m_{y,i}}$ can only be evicted from the L2 cache due to the indirect effect of preemption. Moreover, it is proved in [6] that any reference to a memory block $m_{y,i}$ can lead to at most one L2 miss solely due to the indirect effect of preemption. Therefore, if $\hat{Ref}(m_{y,i})$ is not the first reference to $m_{y,i}$ after preemption it can cause at most one L2 cache miss due to the indirect effect of preemption suffered by $m_{y,i}$. Hence, we reach a contradiction. □

Finally, by using Lemmas 2, 3 and 4 we can bound the maximum number of times any memory block $m_{y,i} \in \hat{UCB}^{P}_{i,2}$ can contribute to the CRPD of task τ_i when τ_i is preempted by a task $\tau_j \in \text{hp}(i)$ at an arbitrary program point P.

Lemma 5. The contribution of a memory block $m_{y,i} \in \hat{UCB}^{P}_{i,2}$ to the CRPD suffered by task τ_i due to a preemption by any higher priority task $\tau_j \in \text{hp}(i)$ at an arbitrary program point P is upper bounded by $\min(k, |Ind^{P}_{m_{y,i}}| + 1) \times d_{L2}$. Where k is the cardinality of the set $\mathbb{R}^{P}_{m_{y,i}}$.

Proof. We prove that for a memory block $m_{y,i} \in \hat{UCB}^{P}_{i,2}$ both k and $|Ind^{P}_{m_{y,i}}| + 1$ are upper bounds on the number of additional L2 cache misses that can be generated due to preemption at a program point P. Therefore, $\min(k, |Ind^{P}_{m_{y,i}}| + 1) \times d_{L2}$ upper bounds the contribution of $m_{y,i}$ to the CRPD suffered by task τ_i at a program point P.

If memory block $m_{y,i}$ has k memory references classified as L2 cache hits in the absence of preemption after the program point P, then, in the worst-case all k references to $m_{y,i}$ may result in L2 cache misses due to preemption at P. Considering that the penalty of a single L2 cache miss is d_{L2}, therefore, the product $k \times d_{L2}$ upper bounds the contribution of $m_{y,i}$ to the CRPD due to preemption at a program point P.

From Lemma 2, we know that only the first reference to memory block $m_{y,i}$ after preemption may result in a L2 cache miss directly due to preemption and all subsequent reference to $m_{y,i}$ after the first reference can result in a L2 cache miss only due to indirect effect of preemption. Also, we proved in Section 5 that $|Ind^P_{m_{y,i}}|$ is an upper bound on the number of memory blocks that can cause an indirect effect of preemption on $m_{y,i}$ after a preemption at P. So, the worst-case scenario is when each memory block in $Ind^P_{m_{y,i}}$ is accessed between every two references to $m_{y,i}$ and every access to a memory block in $Ind^P_{m_{y,i}}$ leads to a L2 cache miss for $m_{y,i}$. Consequently, $m_{y,i}$ can suffer up to $|Ind^P_{m_{y,i}}|$ L2 cache misses due to the indirect effect of preemption caused by memory blocks in $Ind^P_{m_{y,i}}$. Furthermore, from Lemma 4, we know that if the first reference to memory block $m_{y,i}$ after preemption is in loop we can have one additional L2 cache miss after preemption. Consequently, a total of $|Ind^P_{m_{y,i}}| + 1$ L2 cache misses can be generated by the references to memory block $m_{y,i}$ after a preemption at P. Therefore, $(|Ind^P_{m_{y,i}}| + 1) \times d_{L2}$ is also an upper bound on the contribution of $m_{y,i}$ to the CRPD suffered by task τ_i due to a preemption at P. □

6.2.1. CRPD Computation

We use Algorithm 2 to compute the CRPD of task τ_i due to all its L2-UCBs that can be evicted when τ_i is preempted by any higher priority task $\tau_j \in \text{hp}(i)$. The working of Algorithm 2 is explained as follows:

The output of Algorithm 2 is the maximum CRPD cost $\gamma^{P,L_2}_{i,j}$ that can be suffered by task τ_i due to the eviction of all memory blocks in $UCB^P_{i,2}$. $\gamma^{P,L_2}_{i,j}$ is computed by first computing the CRPD $\gamma^{P,L_2}_{m_{y,i},j}$ for every memory block $m_{y,i} \in U\hat{C}B^P_{i,2}$ independently (line 3).

The external loop (lines 5 to 28) is used to compute CRPD for every L2-UCB in $U\hat{C}B^P_{i,2}$. The set of L2 cache hit locations for every memory block $m_{y,i} \in U\hat{C}B^P_{i,2}$ is computed using the function *GetHitLocations*(.) (line 6). The output of *GetHitLocations*(.) is the set $\mathbb{R}^P_{m_{y,i}}$. The algorithm then computes the CRPD of $m_{y,i}$ due to its first reference after preemption point P, i.e., at $R^{1,P}_{m_{y,i}}$ (lines 7 to 17). If the reference to $m_{y,i}$ at $R^{1,P}_{m_{y,i}}$ is in a loop (line 7), that reference may result in up to two L2 cache misses (see Lemmas 3 and 4). So, the algorithm checks for the eviction of $m_{y,i}$ from the L2 cache at $R^{1,P}_{m_{y,i}}$ using both Eq. (18) and (19) (lines 8 to 13) and adds the resulting CRPD to $\gamma^{P,L_2}_{m_{y,i},j}$. However, if the reference to $m_{y,i}$ at $R^{1,P}_{m_{y,i}}$ is not in a loop, it can only suffer a cache miss due to the combination of ECBs of the preempting task τ_j and the indirect effect of preemption. Hence, in this case, the algorithm only computes the CRPD using Eq. (18) (i.e., lines 14 to 17).

All reference to $m_{y,i}$ after $R^{1,P}_{m_{y,i}}$, i.e., at $\{R^{2,P}_{m_{y,i}}, ..., R^{k,P}_{m_{y,i}}\}$, can only result in L2-misses due to the indirect effect of preemption (see Lemma 2). Therefore, Algorithm 2 checks for the eviction of $m_{y,i}$ at all references except $R^{1,P}_{m_{y,i}}$ using only Eq. (19) (lines 19 to 21). Also, we know that the maximum CRPD task τ_i can suffer due to a memory block $m_{y,i} \in U\hat{C}B^P_{i,2}$ is upper bounded by $\min(|\mathbb{R}^P_{m_{y,i}}|, |Ind^P_{m_{y,i}}|+1) \times d_{L2}$ (from Lemma 5). This is considered in the last construct of Algorithm 2 (lines 24 and 26). Finally, the CRPD cost suffered by task τ_i due to a memory block $m_{y,i} \in U\hat{C}B^P_{i,2}$ is available in $\gamma^{P,L_2}_{m_{y,i},j}$, and the total CRPD cost task τ_i may suffer due to all its L2-UCBs is summed in $\gamma^{P,L_2}_{i,j}$ (line 27). Note that the worst-case time complexity of Algorithm 2 is $O(kn)$, where n is the cardinality of the set of L2-UCBs of tasks and k is the number of references to L2-UCBs.

6.3. Handling Nested/Multiple Preemptions

The CRPD analysis presented in [6] assumes that nested/multiple preemptions of a task τ_i by higher priority tasks in hp(i) can be handled by simulating nested preemptions. Consequently, the CPRD τ_j can cause on τ_i is computed by using the union of set of ECBs of all tasks in hep(j) instead of only using the set of ECBs of task τ_j (e.g., see Eq. (7)).

However, when computing CPRD for multilevel caches in the presence of multiple preemptions, only simulating nested preemptions of tasks may not be enough. This is mainly due to the indirect effect of preemption that exists in multilevel caches. Due to the indirect effect of preemption, multiple preemptions by the same or different task(s) can "collaborate" to cause more indirect effect than they would in "isolation". To illustrate, consider the following example:

Algorithm 2 Algorithm to compute the CRPD cost due to the eviction of all L2-UCBs of task τ_i

Output: The total CRPD cost, i.e., denoted by $\gamma_{i,j}^{P,L_2}$, that can be suffered by task τ_i due to the eviction of all its L2-UCBs in $\hat{UCB}_{i,2}^{P}$, in case of a preemption at program point P by any higher priority task $\tau_j \in \text{hp}(i)$.

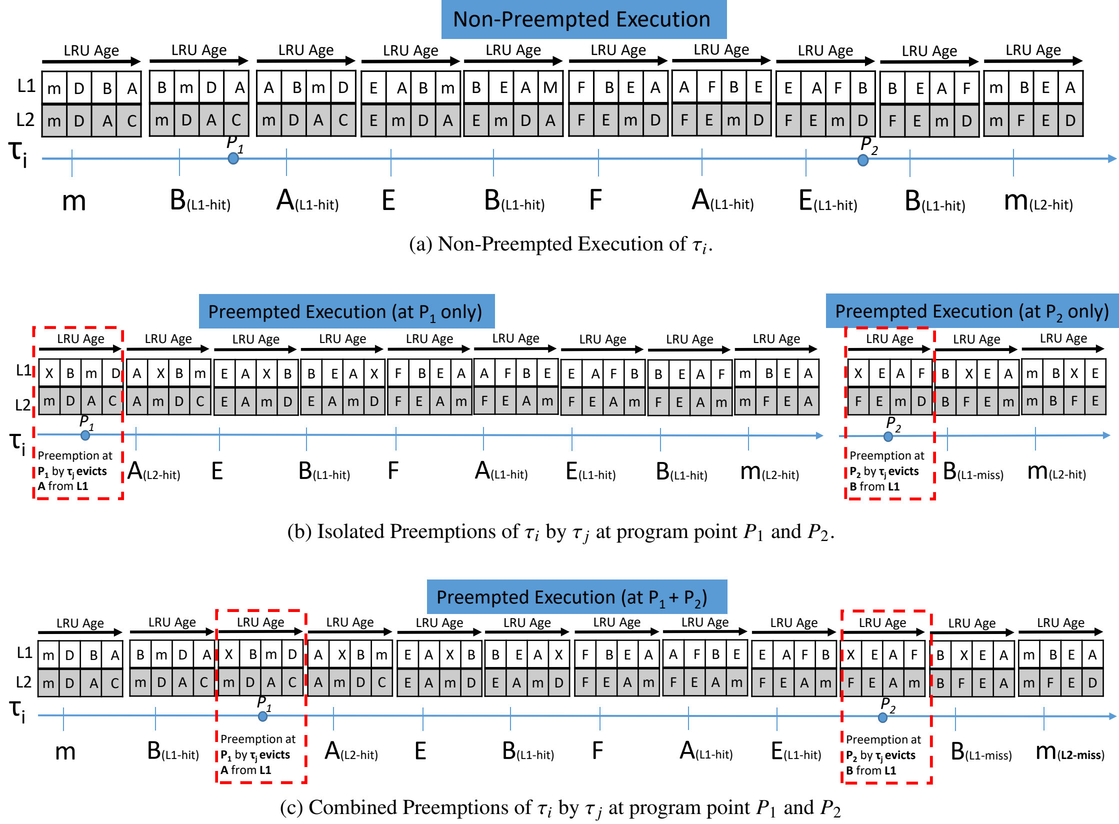
1: $\gamma_{i,j}^{P,L_2} := 0$
2: **for** $\forall m_{y,i} \in \hat{UCB}_{i,2}^{P}$ **do**
3: $\quad \gamma_{m_{y,i},j}^{P,L_2} := 0$
4: **end for**
5: **for** $\forall m_{y,i} \in \hat{UCB}_{i,2}^{P}$ **do**
6: $\quad \mathbb{R}_{m_{y,i}}^{P} = GetHitLocations(m_{y,i}, P)$
7: $\quad$ **if** $R_{m_{y,i}}^{1,P}$ is in loop **then**
8: $\quad\quad$ **if** $MustAge(m_{y,i}, R_{m_{y,i}}^{1,P}, 2) + \left|\bigcup_{\forall h \in \text{hep}(j)} ECB_h^{S_{m_{y,i},2}}\right| + |Ind_{m_{y,i}}^{P}| \geq W_2$ **then**
9: $\quad\quad\quad \gamma_{m_{y,i},j}^{P,L_2} := \gamma_{m_{y,i},j}^{P,L_2} + d_{L2}$
10: $\quad\quad$ **end if**
11: $\quad\quad$ **if** $MustAge(m_{y,i}, R_{m_{y,i}}^{1,P}, 2) + |Ind_{m_{y,i}}^{P}| \geq W_2$ **then**
12: $\quad\quad\quad \gamma_{m_{y,i},j}^{P,L_2} := \gamma_{m_{y,i},j}^{P,L_2} + d_{L2}$
13: $\quad\quad$ **end if**
14: $\quad$ **else**
15: $\quad\quad$ **if** $MustAge(m_{y,i}, R_{m_{y,i}}^{1,P}, 2) + \left|\bigcup_{\forall h \in \text{hep}(j)} ECB_h^{S_{m_{y,i},2}}\right| + |Ind_{m_{y,i}}^{P}| \geq W_2$ **then**
16: $\quad\quad\quad \gamma_{m_{y,i},j}^{P,L_2} := \gamma_{m_{y,i},j}^{P,L_2} + d_{L2}$
17: $\quad\quad$ **end if**
18: $\quad$ **end if**
19: $\quad$ **for** $\forall r \in \mathbb{R}_{m_{y,i}}^{P} \setminus R_{m_{y,i}}^{1,P}$ **do**
20: $\quad\quad$ **if** $MustAge(m_{y,i}, r, 2) + |Ind_{m_{y,i}}^{P}| \geq W_2$ **then**
21: $\quad\quad\quad \gamma_{m_{y,i},j}^{P,L_2} := \gamma_{m_{y,i},j}^{P,L_2} + d_{L2}$
22: $\quad\quad$ **end if**
23: $\quad$ **end for**
24: $\quad$ **if** $\gamma_{m_{y,i},j}^{P,L_2} > \min(|\mathbb{R}_{m_{y,i}}^{P}|, |Ind_{m_{y,i}}^{P}| + 1) \times d_{L2}$ **then**
25: $\quad\quad \gamma_{m_{y,i},j}^{P,L_2} := \min(|\mathbb{R}_{m_{y,i}}^{P}|, |Ind_{m_{y,i}}^{P}| + 1) \times d_{L2}$
26: $\quad$ **end if**
27: $\quad \gamma_{i,j}^{P,L_2} := \gamma_{i,j}^{P,L_2} + \gamma_{m_{y,i},j}^{P,L_2}$
28: **end for**

Example 4. *Fig. 4 shows a sequence of memory references during the execution of a task τ_i considering different preemption scenarios. We assume that L1 and L2 are 4-way set-associative LRU caches and all memory blocks used by task τ_i and the preempting task τ_j map to the same L1 and L2 cache set. Note that we only focus on computing the indirect effect of preemption that can be suffered by memory block m due to preemptions at program points P_1 and P_2.*

During the non-preempted execution of τ_i (see Fig. 4a), first reference to memory block A after P_1 and the first reference to memory block B after P_2 are L1 cache hits. Moreover, the second reference to memory block m is also a L2 cache hit. We assume that task τ_i can be preempted by a higher priority task $\tau_j \in$ hp(i) at program points P_1 and P_2 such that τ_j only has one L1 cache conflict with τ_i, i.e., τ_j only loads ECB X in the L1 cache. When considering preemptions of τ_i by τ_j at P_1 and P_2 separately or in isolation (i.e., Fig. 4b), we can see that by just considering the preemption at P_1, only the first reference to memory block A will be impacted and it will result in a L1 cache miss (but a L2 cache hit). Similarly, a preemption at P_2 in isolation can only impact the first reference to memory block B after P_2 which results in a L2 cache miss. Furthermore, since both preemptions, i.e., at P_1 and P_2, can only evict one L1-UCB, the maximum indirect effect (computed using Algorithm 1) that memory block m may suffer due to a preemption at P_1 or P_2 will be 1. Consequently, we can see in Fig. 4b that the second reference to memory block m will remain a L2 cache hit when considering both preemptions in isolation.

However, when considering consecutive preemptions of task τ_i by task τ_j (see Fig. 4c) we can see that the preemption at program point P_1 can collaborate with the preemption at P_2. This collaboration generates an indirect effect of 2 on memory block m, leading to the eviction of memory block m from the L2 cache after P_2 (see Fig. 4c).

Example 4 shows that Algorithm 1 can underestimate the indirect effect of preemption of memory blocks in the



(a) Non-Preempted Execution of τ_i.

(b) Isolated Preemptions of τ_i by τ_j at program point P_1 and P_2.

(c) Combined Preemptions of τ_i by τ_j at program point P_1 and P_2

Figure 4: Multiple preemption scenarios with collaborating and isolated preemptions. The indirect effect of preemption suffered by memory block m due to consecutive preemptions, i.e., at P_1 and P_2, is higher than the indirect effect caused by individual preemptions.

presence of multiple preemptions. This is mainly because Algorithm 1, computes the indirect effect of preemption that can be caused at a preemption point P by only considering the number of evicted L1-UCBs of tasks w.r.t P. However, as we just demonstrated in Example 4, two consecutive accesses to the memory block under analysis, e.g., $m_{y,i}$, may enclose two or more preemption points (e.g., P_1 and P_2 in Fig. 4). So, all L1-UCBs that may be evicted between two accesses to $m_{y,i}$ can contribute to the indirect effect of preemption of $m_{y,i}$. Therefore, to have a sound estimate on the indirect effect that can be caused due to multiple preemptions w.r.t a program point P, we need to consider all L1-UCBs that may be evicted from the L1 cache between the preemption point under analysis and the program point where a memory block is first accessed after P. For example, if $m_{y,i}$ is first accessed at program point r after the preemption point P. Then, the indirect effect of preemption that $m_{y,i}$ can suffer due to one or more preemptions is upper bounded by the maximum number of distinct L1-UCBs of task τ_i that can be evicted from the L1 cache along any path between P to r.

We use Algorithm 3 to compute the indirect effect of preemption in the presence of nested/multiple preemptions. Algorithm 3 is similar to Algorithm 1 but with some key differences explained as follows; The main difference between Algorithm 1 and 3 is the function $GetProgamPoints(P, FA^{P}_{m_{y,i}})$ which computes $\mathbb{P}_1$ (line 4). $\mathbb{P}_1$ is a set that contains all program locations between the preemption point under analysis, i.e., P, and the program point where memory block $m_{y,i}$ is first accessed after P, i.e., $FA^{P}_{m_{y,i}}$. Note that $FA^{P}_{m_{y,i}}$ is computed using the same $GetFirstAccess(.)$ function as in Algorithm 1. Lines 7 to 10 in Algorithm 3 then computes the indirect effect of preemption for all program locations $P^{'} \in \mathbb{P}_1$ using the exact same steps as used in Algorithm 1 (see Section 5 for details).

The second difference between Algorithm 1 and 3 is the additional condition $m_{x,i} \notin Ind^{mul,P}_{m_{y,i}}$ (line 7). This condition ensures that every memory block $m_{x,i} \in UCB^{P^{'}}_{i,1}$ that can be evicted from the L1 cache at any program point $P^{'} \in \mathbb{P}_1$ is considered only once in the indirect effect of preemption of $m_{y,i}$. This is mainly because, if any access to memory

Algorithm 3 Calculating the indirect effect of preemption in the presence of nested/multiple preemptions

Output: Upper bound on the indirect effect of preemption that can be suffered by every memory block $m_{y,i}$ of task τ_i w.r.t a preemption point P, when considering multiple preemptions of τ_i by a higher priority task $\tau_j \in$ hp(i), i.e., $|Ind^{mul,P}_{m_{y,i}}|$.

1: **for** $\forall m_{y,i} \in \mathbb{M}_i \setminus m_{x,i}$ **do**
2: $\quad Ind^{mul,P}_{m_{y,i}} := \emptyset$
3: $\quad FA^{P}_{m_{y,i}} := GetFirstAccess(m_{y,i}, P)$
4: $\quad \mathbb{P}_1 := GetProgamPoints(P, FA^{P}_{m_{y,i}})$
5: $\quad$ **for** $\forall P' \in \mathbb{P}_1$ **do**
6: $\quad\quad$ **for** $\forall m_{x,i} \in UCB^{P'}_{i,1}$ **do**
7: $\quad\quad\quad$ **if** $((CU^{P',1}_{m_{x,i}} + |\bigcup_{\forall h \in \text{hep}(j)} ECB^{S_{m_{x,i},1}}_{h}| \geq W_1) \wedge (m_{x,i} \notin Ind^{mul,P}_{m_{y,i}}))$ **then**
8: $\quad\quad\quad\quad FA^{P'}_{m_{x,i}} := GetFirstAccess(m_{x,i}, P')$
9: $\quad\quad\quad\quad$ **if** $((MustAge(m_{y,i}, FA^{P'}_{m_{x,i}}, 2) \neq \infty) \wedge (S_{m_{y,i},2} == S_{m_{x,i},2}) \wedge (MustAge(m_{x,i}, FA^{P'}_{m_{x,i}}, 2) > MustAge(m_{y,i}, FA^{P'}_{m_{x,i}}, 2)))$ **then**
10: $\quad\quad\quad\quad\quad Ind^{mul,P}_{m_{y,i}} := Ind^{mul,P}_{m_{y,i}} \cup m_{x,i};$
11: $\quad\quad\quad\quad$ **end if**
12: $\quad\quad\quad$ **end if**
13: $\quad\quad$ **end for**
14: $\quad$ **end for**
15: **end for**

block $m_{x,i}$ at any program location in $\mathbb{P}_1$ results in a L1 cache miss then, $m_{x,i}$ will be reloaded to the L1 cache from the L2 cache or from the main memory. In both cases, $m_{x,i}$ will become the youngest element in the L1 and L2 cache. Therefore, $m_{x,i}$ cannot contribute to the indirect effect of preemption of $m_{y,i}$ more than once.

Considering that every memory block $m_{x,i} \in UCB^{P'}_{i,1}$ that satisfies all conditions in line 9 can contribute to the indirect effect of preemption of $m_{y,i} \in \mathbb{M}_i$, the set $Ind^{mul,P}_{m_{y,i}}$ is used to holds all such memory blocks. Consequently, the indirect effect of preemption of a memory block $m_{y,i}$ of task τ_i can suffer due to one or more preemptions by any higher priority task $\tau_j \in$ hp(i) is upper bounded by $|Ind^{mul,P}_{m_{y,i}}|$.

6.4. Computing total CRPD and the WCRT Analysis

In this section, we explain how the total CRPD task τ_i can suffer due to a preemption by any higher priority task $\tau_j \in$ hp(i) is computed and incorporated into the WCRT analysis.

We know that the sum of Eq. (15) and (16) upper bound the CRPD any task τ_i may suffer due to evictions of its L1-UCBs by any higher priority task $\tau_j \in$ hp(i). Similarly, Algorithm 2 computes an upper bound on the CRPD of τ_i due to eviction of its L2-UCBs. Therefore, the CRPD task τ_i can suffer due to a preemption by task τ_j is obtained by maximizing Eq. (15), (16) and Algorithm 2 over all program points in τ_i, i.e.,

$$\gamma^{\overline{H}}_{i,j} = \max_{P \in \mathbb{P}} (\gamma^{P,L_1}_{i,j} + \gamma^{P,L_{12}}_{i,j} + \gamma^{P,L_2}_{i,j}) \tag{20}$$

Note that when computing $\gamma^{\overline{H}}_{i,j}$ the indirect effect of preemption is computed using Algorithm 3.

Considering that a higher priority task $\tau_j \in$ hp(i) can preempt any task $\tau_k \in$ aff(i, j) during the response time of task τ_i Therefore, to ensure that the maximum CRPD cost is considered for every preemption of τ_i by τ_j, Eq. (20) is further maximized over all tasks in aff(i, j), i.e.,

$$\gamma^{\overline{H},max}_{i,j} = \max_{\forall k \in \text{aff}(i,j)} \gamma^{\overline{H}}_{i,j} \tag{21}$$

Eq. (21) is built using the same principle as the ECB-union approach presented in [4] (see Eq.10 of [4]). It accounts for both nested preemptions (i.e., τ_i preempted by τ_k which is preempted by τ_j) and consecutive preemptions (of τ_i by τ_k and τ_j). The CRPD cost of a direct preemption of task τ_i by task $\tau_j \in$ hp(i) is accounted for in the term $\gamma^{\overline{H},max}_{i,j}$, whereas the indirect CRPD cost task τ_j may generate due to preemption of any task $\tau_k \in$ aff(i, j) during the response time of τ_i (i.e., a nested preemption) will be accounted for in the term $\gamma^{\overline{H},max}_{i,k}$, i.e., due to the use of union of ECBs of all tasks hep(k) when computing $\gamma^{\overline{H}}_{i,k}$. Thus, for every preemption of task τ_i by task τ_j, taking the maximum CRPD cost over all tasks in aff(i, j) ensures that the highest number of cache evictions are considered.

6.4.1. WCRT Computation

The WCRT analysis for fixed priority preemptive systems (FPPS) that accounts for CRPDs was proposed in [12]. We incorporate $\gamma_{i,j}^{\overline{H},max}$ into the same WCRT formulation proposed in [12] to compute the WCRT R_i of task τ_i, i.e.,

$$R_i = C_i + \sum_{\forall j \in \text{hp}(i)} \left\lceil \frac{R_i}{T_j} \right\rceil \times \left(C_j + \gamma_{i,j}^{\overline{H},max} \right) \tag{22}$$

where $\gamma_{i,j}^{\overline{H},max}$ represent the CRPD cost due to each job of a higher priority preempting task $\tau_j \in \text{hp}(i)$ executing within the response time of task τ_i. Note that Eq. (22) is recursive. However, a solution can be found using simple fixed-point iteration on R_i by initializing R_i to C_i. The iteration stops as soon as R_i converges or $R_i > D_i$, in which case the task is deemed unschedulable.

7. Experimental Evaluation

This section details how our proposed CRPD analysis for non-inclusive multilevel caches compares against the state-of-the-art analysis of [6]. First, we explain how the input quantities required by the analyses are computed and then detail experiments that were conducted to compares the performance of both analyses.

7.1. Deriving Parameters for the Analyses

We have used the Heptane [13] static WCET analysis tool to derive parameters needed to compare our CRPD analysis against the analysis of [6]. Heptane is an open source WCET analysis tool that supports multilevel non-inclusive caches and implements the WCET analysis presented in [10]. However, currently the output of Heptane is the WCET of the analyzed benchmark and few cache statistics. Therefore, we had to modify Heptane[1] to compute all the required parameters.

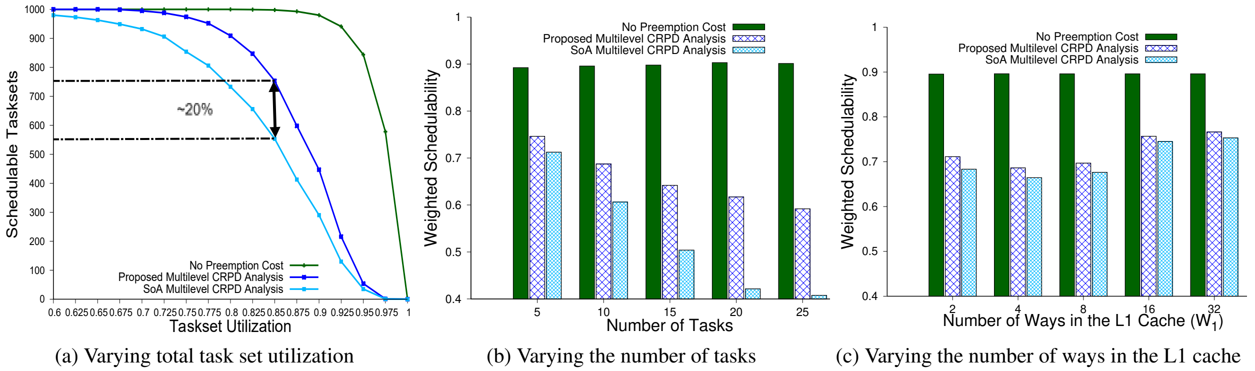
We have added a new module named `MultiCRPDAnalysis` to Heptane that enables us to compute different parameters needed for the analysis. The set of L1- and L2-UCBs of tasks are computed using the Must-cache analysis along with a backward flow analysis on the control flow graph. The backward flow analysis computes the abstract cache state at the exit of a basic block by using the join operation on all the abstract cache states at the entry of its successors. For every memory block $m_{x,i}$ used by task τ_i the analysis starts by assuming $CU^P_{m_{x,i}} = (\infty, \infty)$. Then for every program point P, the analysis checks the accessed memory block and update the abstract cache state using the Must-update and Must-join operations defined in [10]. A memory block $m_{x,i}$ is considered a L1-UCB at program point P if it satisfies Eq. (13). Similarly, all memory blocks that satisfy Eq. (14) w.r.t a program point P are considered L2-UCBs at that program point. Note that our analysis to derive the set of L1-UCBs for multilevel caches is similar to the UCB analysis in [6] however, we additionally derive the set of L2-UCBs w.r.t every program point P. The set of ECBs of task τ_i are computed using the May-cache analysis [10] that determines the set of all memory blocks used by task τ_i at each cache level.

To compute the indirect effect of preemption, we use a forward flow analysis along with the Must-cache analysis [9]. Knowing that the indirect effect of preemption is caused by memory blocks that were L1 cache hits in the absence of preemption but may be accessed from the L2 cache or main memory after preemption, the forward flow analysis (along with Must-case analysis) determines all memory blocks with L1 cache hits. For the analysis in [6], the forward flow analysis is performed starting from the entry point of the program and ending at program point r, where the memory block under analysis, e.g., $m_{x,i}$, can be accessed. For our analysis, the forward flow analysis is performed for each pair of program locations between two accesses to the memory block $m_{x,i}$. In both cases, the largest set of memory blocks is used when computing the indirect effect of preemption of $m_{x,i}$. Since Heptane allows to analyze each cache level separately, other parameters needed for the implementation of Eq. (4) and Algorithm 1 are extracted using the Must-cache analysis [10]. Similarly, the `cfglib` library used by Heptane allows to compute loop bound for each basic block. This information is then used in Algorithm 2 to compute the CRPD due to the eviction of L2-UCBs. Note that our proposed `MultiCRPDAnalysis` is based on the cache analysis presented in [10], that uses Abstract Interpretation (AI) based cache analysis [9]. It is demonstrated in [9], that the domain of the AI based cache analysis is finite and the fixpoint iterations terminate in a reasonable computation time.

[1]The Modified version of Heptane is available on demand by contacting the first author.

Table 1: Benchmarks parameters from the Mälardalen Benchmark Suite [14] used during the experiments.

Name	C_i	L1-ECBs	L2-ECBs	L1-UCBs	L2-UCBs	Name	C_i	L1-ECBs	L2-ECBs	L1-UCBs	L2-UCBs
bs	4020	11	6	11	2	bsort100	5811344	20	10	19	3
crc	1782419	43	22	43	10	expint	647343	19	11	18	3
fibcall	14023	8	4	8	3	insertsort	52245	16	8	16	4
lcdnum	8640	12	6	12	1	matmult	1795585	28	14	28	3
ns	129598	20	10	20	5	qurt	111554	53	33	53	24
fir	96215	22	11	22	4	prime	248299	17	9	17	7
select	145160	26	14	26	3	sqrt	20159	26	13	25	5
minmax	4435	17	9	16	6	ud	145170	75	38	74	6
minver	58155	167	84	167	58	fft	1252363	141	71	140	13
statemate	229575	261	133	254	37	fdct	101944	106	53	106	2
jfdctint	100331	96	48	96	2	ludcmp	341583	98	49	98	8
nsichneu	515015	1377	512	1377	422						



(a) Varying total task set utilization (b) Varying the number of tasks (c) Varying the number of ways in the L1 cache

Figure 5: Task set schedulability by varying the total task set utilizations, the total number of tasks in a task set, and the L1 cache size

7.2. Experimental Setup

All experiments were performed using the Mälardalen benchmark suite [14]. Mälardalen benchmark suite comprise of different benchmarks that are representative of typical workloads used in real-time applications. For every benchmark, parameters such as the WCET, set of L1- and L2-UCBs, set of L1- and L2-ECBs, maximum LRU-ages of memory blocks, total number of references, number of references in loops etc., were extracted using Heptane. The target architecture was a single core MIPS R2000/R3000 processor with a two level instruction cache hierarchy such that, L1 cache is 2-way set-associative with 32 sets and line size of 32-bytes, and L2 cache is 4-way set-associative with 64 sets and line size of 64-bytes. The L1 cache miss penalty was 10 processor cycles, i.e., $d_{L1} = 10$, and the L2 cache miss penalty was 100 processor cycles. Table 1 shows the benchmark parameters used in the experiments.

The other task set parameters were randomly generated as follows. The default number of tasks in each task set were 10 with task utilization generated using UUnifast [15]. Each task was randomly assigned values of one of the benchmark in Table 1. Task deadlines were implicit with priorities assigned in a deadline monotonic order. Task periods were set such that $T_i = C_i/U_i$.

7.3. Experiments

We performed several experiments by varying different parameters and used task set schedulability, i.e., the total number of task sets deemed schedulable, as the performance metric. The WCRT formulation in Eq. (22) is used to compute the WCRT of tasks. For all the analyzed approaches, a given task set Γ is deemed schedulable only if for every task $\tau_i \in \Gamma$, the WCRT R_i is less than the deadline of τ_i, i.e., $R_i \leq D_i$. Otherwise, Γ is deemed unschedulable.

1. Task set Utilizations: In this experiment, we varied the total task set utilization from 0.025 to 1 in steps of 0.025 and randomly generated 1000 task sets per utilization point. Fig. 5a shows the number task sets that were deemed schedulable using the “SoA Multilevel CRPD analysis [6]” and our “Proposed Multilevel CRPD analysis”. The

green line marked as “No Preemption cost” provides an upper bound on the number of task sets that were deemed schedulable without considering any CRPD cost. For clarity, we only show a cropped version of the plot in Fig. 5a starting from a utilization of 0.6. All approaches produce identical results below this point. Fig. 5a shows that our proposed approach performs significantly better in comparison to the SoA analysis. The proposed analysis dominates the SoA analysis mainly due to two reasons: (i) it provides a tighter bound on the indirect effect of preemption and (ii) it accurately estimates the CRPD of tasks at the L2 cache. Although, the major improvement in the CRPD computation results from the treatment of memory blocks with L2 cache hits, however, we can see that the number of L2-UCBs of tasks (see Table 1) is very small in comparison to the number of L1-UCBs. However, our proposed analysis still results in scheduling more task sets and improving task set schedulability by up to 20% percentage points over the existing analysis.

2. Number of Tasks: In this experiment, we varied the total number of tasks per task set between 5 to 25 in steps of 5. For all other parameters default values were considered. Since we vary both the total task set utilizations and the number of tasks, we have used the weighted schedulability [16, 17] measure to generate the plot shown in Fig. 5b. The weighted schedulability measure reduces what would otherwise be a 3-dimensional plot to 2-dimensions by eliminating the axis of task set utilization.

Intuitively, increasing the number of tasks tends to decrease task set schedulability. This is due to an increase in the number of preemptions, which also leads to an increase in the overall CRPD cost. Fig. 5b also confirms this decrease in task set schedulability. However, we can see that our proposed CRPD analysis always dominates the SoA CRPD analysis. In fact, for higher number of tasks per task set (e.g., for 20 or 25 tasks per task set) the difference between the weighted schedulability of both approaches tends to increase. This is due to the excessive pessimism in the SoA CRPD analysis that can count the evictions of same memory blocks several times. This pessimism is reduced by our analysis by bounding the number of times each memory block can contribute to the CRPD.

3. Number of Ways in the L1 Cache (W_1): In this experiment, we varied the L1 cache associativity, i.e., W_1, and evaluated its impact on the performance of all the analyzed approaches. All other parameters were set to their default values. However, since we focus on a cache configuration where L1 cache associativity is always less than or equal to the L2 cache associativity, i.e., $W_1 \leq W_2$. Therefore, for this experiment, we also fixed the L2 cache associativity to 32, i.e., $W_2 = 32$. We then varied the number of ways in the L1 cache between 2 to 32 and plotted the weighted schedulability for both approaches as shown in Fig. 5c. Note that increasing the number of ways in the L1 cache will also increase the size of the L1 cache.

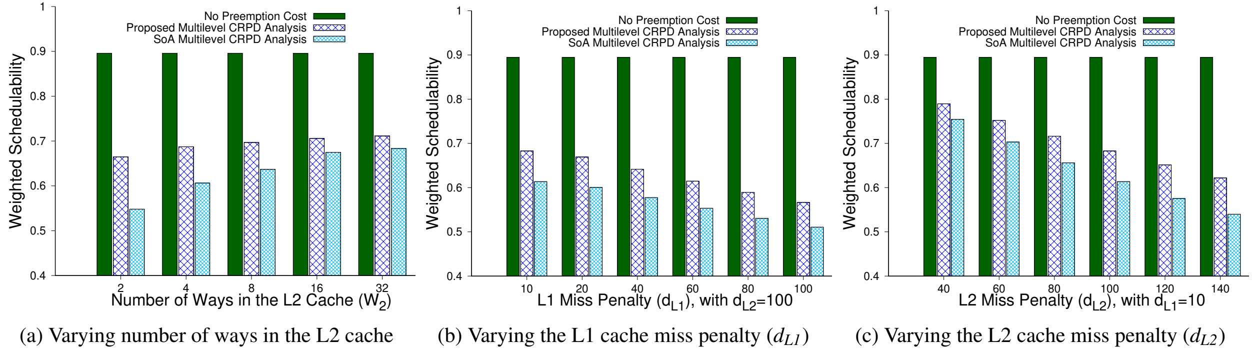
We can see in Fig. 5c that by varying the number of ways in the L1 cache (i.e., L1 cache size), both approaches produce similar results with the proposed approach marginally improving task set schedulability. This is mainly because, for both approaches the CRPD analysis for the L1 cache is very similar except for the computation of the indirect effect of preemption. Moreover, with the number of ways in the L2 cache set to 32, the L2 cache size becomes relatively larger w.r.t the analyzed benchmarks, which leads to almost no CRPD due to the L2 cache. Therefore, we can observe that increasing the number of ways in the L1 cache has a similar effect on both analyses.

4. Number of Ways in the L2 Cache (W_2): We also performed an experiment by varying the number of ways in the L2 cache, i.e., W_2, between 2 to 32. Default values were used for all the other parameters. The results are shown in Fig. 6a. Note that increasing the number of ways in the L2 cache also increases its size.

Fig. 6a shows that when varying the number of ways in the L2 cache (i.e., increasing the L2 cache size), the difference between the performance of both analyses becomes more prominent. This is because our analysis provides much tighter bound on the CRPD of task at the L2 cache than the existing analysis. We can see in Fig. 6a that when the L2 cache is smaller, i.e., the potential CRPD due to the L2 cache is higher, our approach performs significantly better than the existing analysis. However, by increasing the number of ways in the L2 cache, the difference between the performance of both analysis tends to reduce. This is mainly due to an overall reduction in the L2 CRPD due to a larger L2 cache. Note that we also performed experiments by varying the number of sets in the L1/L2 cache and respectively observed a similar trend as shown in Fig. 5c and 6a.

5. L1 and L2 Cache Miss Penalties (d_{L1} and d_{L1}): We conducted two more experiments by varying the L1 and L2 cache miss penalties. Default values were used for all other parameters.

In the first experiment, we varied the L1 cache miss penalty between 10 to 100 processor cycles and the resulting weighted schedulability measure is shown in Fig. 6b. For most architectures, L1 cache miss penalty is much smaller than the L2 cache miss penalty, therefore, for this experiment, we set $d_{L2} = 100$ cycles. Fig. 6b shows that by increasing the L1 cache miss penalty the weighted schedulability for both approaches decreases. However, the difference



(a) Varying number of ways in the L2 cache (b) Varying the L1 cache miss penalty (d_{L1}) (c) Varying the L2 cache miss penalty (d_{L2})

Figure 6: Weighted schedulability results by varying the size of L1 and L2 caches

between the performance of both analysis remains nearly constant due to a similar L1 CRPD analysis.

For the second experiment, we varied the L2 cache miss penalty between 40 to 140 processor cycles. Default value was used for L1 cache miss penalty. The results are shown in Fig. 6c. We can see that for lower values of L2 cache miss penalty the difference between the weighted schedulability of both approaches is smaller. However, by increasing the value of L2 miss penalty the difference between the performance of both approaches also increases, which is due to a tighter bound on the L2 CRPD by the proposed analysis.

Analysis Time: The proposed CRPD analysis is performed in two steps. In the first step, the WCET of tasks and the set of L1/L2 ECBs and UCBs of tasks is determined using Heptane as discussed in Section 7.1. In the second step, the WCRT of tasks is computed by first computing the CRPD of tasks using our proposed analysis and the state-of-the-art CRPD analysis [6].

For a two-level cache hierarchy, Heptane can compute the WCET of the benchmarks given in Table 1 in less than one minute on an Intel core i-7 processor (3.4GHz) with 16GB RAM. With the addition of our `MultiCRPDAnalysis` module to Heptane the total time taken to analyze the benchmarks in Table 1 increases on average by 20%. The WCRT analysis however, is much more compute intensive. On an Intel core i-7 processor (3.4GHz) with 16GB RAM, it takes on average 25 minutes to produce the plot shown in Fig. 5a. However, the difference between the run-time of our proposed analysis and [6] is negligible since we essentially built on the CRPD analysis in [6] and use additional information to improve the CRPD bounds.

8. Related Work

Many different approaches have been proposed in the state-of-the-art to bound CRPDs considering single-level caches. Lee et al. [1] introduced the notion of UCBs and used the number of UCBs of the preempted tasks to bound CRPD. Tomiyama and Dutt [2] formally introduced the notion of ECBs and used the number of ECBs of the preempting tasks to bound CRPD. Tan and Mooney [18] presented an approach to bound CRPD using UCBs of the preempted tasks and ECBs of the preempting tasks. Staschulat et al. [19] and Altmeyer et al. [4, 20] also used both the UCBs of the preempted tasks and ECBs of the preempting tasks in order to come up with more precise bounds on the CRPD cost. Building on the analysis in [4], Markovic et al. [21] have recently proposed an analysis that dominate all the existing approaches for CRPD calculation for single-level cache. Rashid et al. [22–24] have also presented several approaches that use the notion of cache persistence along with CRPD to improve the bounds on preemption related cache overheads for single-level caches.

Due to the added complexity of analyzing cache conflicts at multiple cache levels only few existing approaches [6, 7] have focused on the CPRD analysis for multilevel caches. The work of Zhang et al. [7] focus on the computation of CRPD for multilevel inclusive caches. Their analysis identifies challenges in the computation of CRPD due to cache inclusion policy and use the notion of useful positive reference (UPR), i.e., references that are positively classified by the intra-task cache analysis (e.g., cache hits), to compute CRPD. The analysis derives the set of positive references that can be considered as UPRs and bound the number of times each of them may act as a UPR. This information is then used to upper bound the CRPD. However, the analysis in [7] only use the UPRs of the preempted task to bound

CRPD and does not consider the preempting tasks, which might lead to overly pessimistic CRPD bounds. Zhang et al. [25] also presented a WCET analysis that focuses on the write-back mechanism of multilevel caches.

The only existing work that focus on the CRPD analysis for multilevel non-inclusive cache is presented in [6]. It relies on the cache behavior analysis for non-inclusive cache hierarchies that is proposed in [10]. The analysis in [6] identifies challenges in the computation of CRPD for multilevel inclusive caches, i.e., the indirect effect of preemption, and provide solutions to overcome those challenges. However, as we have shown in this work, the analysis in [6] may lead to pessimistic CRPD bounds by overestimating the indirect effect of preemption and the CRPD suffered by tasks due to L2 cache hits.

9. Conclusion and Future Work

In this paper, we presented an analysis to compute CRPDs for multilevel non-inclusive caches. We redefined the notion of UCBs for multilevel caches, showed how to find those UCBs and used them to compute the CRPD. We showed that a tighter bound on the indirect effect of preemption can be obtained by calculating the indirect effect of preemption that can be caused instead of calculating the indirect effect of preemption that can be suffered by memory blocks. We then presented a new analysis to compute the CRPD due to memory blocks that were categorized as L2 cache hits in the absence of preemptions but may become L2 cache misses after preemptions. Our analysis provides a tighter CRPD bound than the existing analysis by identifying how many references to a memory block can be impacted due to preemptions and therefore may contribute to the CRPD.

We evaluated the performance of our proposed CRPD analysis against an existing analysis from the state-of-the-art. Experiments were performed by varying different parameters with most values taken from the Mälardalen benchmarks. Experimental results show that our proposed CRPD analysis dominates the existing analysis and results in up to 20% percentage points higher task schedulability.

As future work, we aim to extend this analysis to consider shared last-level cache in multicore platforms.

Acknowledgment

This work was partially supported by National Funds through FCT/MCTES (Portuguese Foundation for Science and Technology), within the CISTER Research Unit (UIDP/UIDB/04234/2020); also by the Operational Competitiveness Programme and Internationalization (COMPETE 2020) under the PT2020 Partnership Agreement, through the European Regional Development Fund (ERDF), and by national funds through the FCT, within project PREFECT (POCI-01-0145-FEDER-029119); also by the European Union's Horizon 2020 - The EU Framework Programme for Research and Innovation 2014-2020, under grant agreement No. 732505. Project "TEC4Growth - Pervasive Intelligence, Enhancers and Proofs of Concept with Industrial Impact/NORTE-01-0145-FEDER000020" financed by the North Portugal Regional Operational Programme (NORTE 2020), under the PORTUGAL 2020 Partnership Agreement.

References

[1] C. G. Lee, J. Hahn, Y. M. Seo, S. L. Min, R. Ha, S. Hong, C. Y. Park, M. Lee, C. S. Kim, Analysis of cache-related preemption delay in fixed-priority preemptive scheduling, Computers, IEEE Transactions on 47 (6) (1998) 700–713.

[2] H. Tomiyama, N. D. Dutt, Program path analysis to bound cache-related preemption delay in preemptive real-time systems, in: CODES, 2000, pp. 67–71.

[3] S. Altmeyer, C. Maiza, J. Reineke, Resilience analysis: Tightening the crpd bound for set-associative caches, in: LCTES, ACM, 2010, pp. 153–162.

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

[5] M. Lv, N. Guan, J. Reineke, R. Wilhelm, W. Yi, A survey on static cache analysis for real-time systems, Leibniz Transactions on Embedded Systems 3 (1) (2016) 05–1.

[6] S. Chattopadhyay, A. Roychoudhury, Cache-related preemption delay analysis for multilevel noninclusive caches, ACM Transactions on Embedded Computing Systems (TECS) 13 (5s) (2014) 1–29.

[7] Z. Zhang, X. Koutsoukos, Cache-related preemption delay analysis for multi-level inclusive caches, in: Proceedings of the 13th International Conference on Embedded Software, 2016, pp. 1–10.

[8] C. L. Liu, J. W. Layland, Scheduling algorithms for multiprogramming in a hard-real-time environment, JACM 20 (1) (1973) 46–61.

[9] H. Theiling, C. Ferdinand, R. Wilhelm, Fast and precise wcet prediction by separated cache and path analyses, Real-Time Systems 18 (2-3) (2000) 157–179.

[10] D. Hardy, I. Puaut, Wcet analysis of multi-level non-inclusive set-associative instruction caches, in: 2008 Real-Time Systems Symposium, IEEE, 2008, pp. 456–466.

[11] S. Altmeyer, Analysis of preemptively scheduled hard real-time systems, epubli GmbH, 2013.

[12] J. Busquets-Mataix, J. J. Serrano, R. Ors, P. Gil, A. Wellings, Adding instruction cache effect to schedulability analysis of preemptive real-time systems, in: RTAS, 1996, pp. 204–212.

[13] D. Hardy, B. Rouxel, I. Puaut, The heptane static worst-case execution time estimation tool, in: 17th International Workshop on Worst-Case Execution Time Analysis (WCET 2017), Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

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

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

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

[17] A. Burns, R. I. Davis, Adaptive mixed criticality scheduling with deferred preemption, in: 2014 IEEE Real-Time Systems Symposium, IEEE, 2014, pp. 21–30.

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

[19] J. Staschulat, S. Schliecker, R. Ernst, Scheduling analysis of real-time systems with precise modeling of cache related preemption delay, in: ECRTS, 2005, pp. 41–48.

[20] S. Altmeyer, R. I. Davis, C. Maiza, Cache related pre-emption delay aware response time analysis for fixed priority pre-emptive systems, in: RTSS, 2011, pp. 261–271.

[21] F. Marković, J. Carlson, S. Altmeyer, R. Dobrin, Improving the accuracy of cache-aware response time analysis using preemption partitioning, in: 32nd Euromicro Conference on Real-Time Systems (ECRTS 2020), Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[22] S. A. Rashid, G. Nelissen, D. Hardy, B. Akesson, I. Puaut, E. Tovar, Cache-persistence-aware response-time analysis for fixed-priority preemptive systems, in: ECRTS, 2016, pp. 262–272.

[23] S. A. Rashid, G. Nelissen, S. Altmeyer, R. I. Davis, E. Tovar, Integrated analysis of cache related preemption delays and cache persistence reload overheads, in: RTSS, IEEE, 2017, pp. 188–198.

[24] S. A. Rashid, G. Nelissen, E. Tovar, Bounding cache persistence reload overheads for set-associative caches, in: 2020 IEEE 26th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), IEEE, 2020, pp. 1–10.

[25] Z. Zhang, Z. Guo, X. Koutsoukos, Handling write backs in multi-level cache analysis for wcet estimation, in: Proceedings of the 25th International Conference on Real-Time Networks and Systems, 2017, pp. 208–217.